MASCOT: Microarchitecture synthesis of control paths

A J W M ten Berg

This paper presents MASCOT (MicroArchitecture Synthesis of ConTrol paths). This synthesis system constructs the optimal microarchitecture for a control path of an instruction set processor. Input to the system is the behavioural specification of a control path. This specification is in finite state machine form which is mapped initially onto a single programmed logic array (PLA) microarchitecture. The synthesis strategy then applies a sequence of decompositions on this initial microarchitecture. This strategy follows a decision scheme until all design objectives are met. It transforms the initial microarchitecture into a complex microarchitecture of several PLAs and ROMs. Where it is impossible to meet the design objectives, the system constructs a microarchitecture which comes as close as possible to given design objectives. Design objectives are allowed on floorplan dimensions and delay. Our strategy integrates a number of known optimization methods for specific microarchitectures. Therefore this synthesis method explores a larger part of the design space than do other control path synthesis methods. Other methods are mostly bound to one microarchitecture which they optimize. Our system is not only very flexible in microarchitecture construction but also open for extension by other optimizations.

Keywords: control structures, control design styles, logic design

Most of today's control path synthesis systems are restricted to one target microarchitecture, mostly a programmed logic array (PLA) or microprogrammed microarchitecture. This application of standard microarchitectures, with only minor variations allowed, leaves large parts of the design space for control path implementations unexplored. Furthermore, most automated systems optimize the design with one objective only, mostly the design size. Logic synthesis systems[1,2] and systems for state-assignment[3–5] are examples of such systems. Some systems[6] take delay into account, but a floorplanning objective is in general not included. A partial cause for such one-parameter optimizations may be found in the extensive use of benchmarks for comparison, as for example in Reference 7. For ease and clarity of comparison the competition stresses just one design parameter, for example on the number of literals or product terms. For design methods which optimize to a collection of design parameters, however, such comparisons are far more difficult to perform. This is because the user determines to which combination of design parameter values the system must optimize, instead of to a fixed entity as minimal size. Because industry does not need systems which are highly optimal for one design parameter, but requires a design balanced to a set of design parameters we developed a system which explores the design space for control paths and also incorporates several well known optimizations. Our system accepts floorplan and delay objectives and is optimized for the design of control paths of instruction set processors (ISP). Instruction set processors contain control functions which are largely independent of the data processing, as opposed to digital signal processors (DSP) where the data flow determines the control function. For the DSP data processing part a wide variety of synthesis methods already exists[8,9]. MASCOT accepts a behavioural description which must be of the finite state machine (FSM) type and delivers a control path microarchitecture in a netlist type description with additional specifications for the personality matrices of the PLAs. MASCOT's strategy is a decomposition sequence, which is guided by floorplanning, delay and size evaluation. It starts with coarse grain decompositions which have an overall impact and ends up with fine grain decompositions which have an impact on details. This ensures the highest possible design regularity, a parameter recognized earlier[10] as an important measure for design quality.

Computer Science Department, University of Twente, PO Box 217, 7500 AE Enschede, The Netherlands
Paper received: 9 December 1991

CONTROL PATH SYNTHESIS AND DECOMPOSITION

In control path synthesis a number of design levels can be distinguished. Each has its specific function with respect to functionality and structure. These levels are:

- FSM synthesis level
- Microarchitecture or structure level
- Logic level

Each level concentrates on different design aspects of control path implementation. The FSM level synthesis deals with the specification of minimal finite state machines out of the behavioural specification of the processor given in for example VHDL. This level includes decisions on whether or not to apply more interacting FSMs instead of a single FSM. Multiple FSM control can be profitable in cases of semi-distributed data processing. The FSM specification has a strong interaction with design choices for the data path implementation. The design at this level abstracts from the implementation structures available for FSMs. These are constructed at the next level of design, which we call the microarchitecture or structure level. This level does the design of an optimal configuration of PLAs, registers and multiplexers which implements the FSM specified at the previous level. Analysis of the functional structure in the FSM, expressed by for example the state-graph and the output variable interdependency, provides a basis for decisions on microarchitecture alternatives. This microarchitecture level is analogous to the register transfer level (RTL) in data path synthesis. The RTL level describes the network of combinatorial blocks connected by buses, registers and multiplexers which implement the data processing. The last level in *Figure 1*, the logic level, implements the Boolean equations contained in each combinatorial function. The main question at this level is whether standard-cell based multi-level logic implementations should be preferred over the more general PLA based implementations.

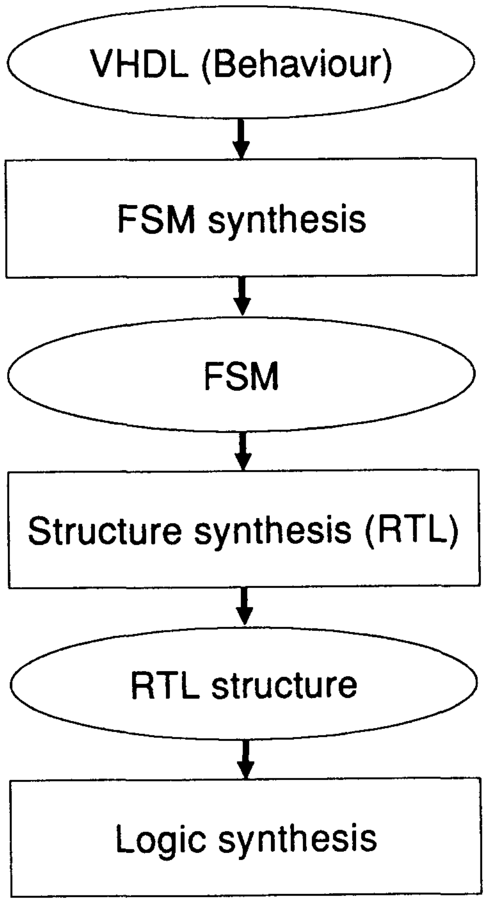


Figure 1 Design flow

The option to implement the combinatorial part of the FSM initially in one PLA allows decomposition as a method for synthesis. Our view on the synthesis process is that of stepwise refinement of the microarchitecture. Decomposition gradually works more details from the FSM function into the FSM microarchitecture. Initially the microarchitecture shows no details of the FSM function as it consists of a single PLA. Only some coarse parameters of the FSM such as the number of product terms or number of output variables are visible in the PLA dimensions. The initial single PLA hides both the FSM state-transition structure and the logic structure completely. On the other hand, the ultimate form of decomposition, a multi-level logic implementation, shows the exact structure of the logic, because all elementary logic functions and their relations are visible as cells and wiring. The microarchitecture resembles the FSM functional structure better after each decomposition. Decomposition stops when the implementation structure fulfils the design objectives given, if possible. This principle of microarchitecture refinement by a sequence of decompositions is also feasible from another point of view. There is no need to consider all design details or primitives together at one decision level. This reduces the computational complexity of synthesis.

This synthesis strategy results in a high regularity factor of the microarchitecture[10]. The lower the number of individually designed components and the higher the repetition factor for each of them, the more regular the microarchitecture. A high design regularity improves the predictability of layout parameters during microarchitecture synthesis. One of the main reasons for the difficult estimation of the area of microarchitectures with a low regularity is the large amount of interconnect found in them. Interconnect area is difficult to estimate. Also delay is harder to estimate in an irregular microarchitecture, because the critical path is harder to identify. MASCOT's synthesis strategy also allows incorporation of known optimization algorithms[3, 5, 11] dedicated to specific microarchitectures. All decompositions defined in our synthesizer inherit the floorplanning aspect[12, 13]. Floorplanning, formerly part of the placement and routing design phases, is integrating more and more into high level synthesis methods. This significantly reduces the placement and routing problem which follows microarchitecture synthesis, because the parts of the microarchitecture will better fit together.

Related work

Microprogram development methods[14-16] have a special position in the synthesis process, as was recognized first in Reference 17. These methods are based on a fixed microarchitecture and focus completely on the problem of generating an optimal microprogram to cooperate with the data path. These methods therefore specify the behaviour of the control path based on knowledge of one default microarchitecture. This problem is quite different from the synthesis problem tackled here, which concerns construction of the optimal microarchitecture for a previously defined control path behaviour. Therefore microprogramming methods can be regarded as FSM synthesis methods (*Figure 1*) for dedicated microarchitectures.

At the microarchitecture level a large number of minimization methods exist. Some systems focus on PLA implementations, with minimization techniques varying from state-assignment, PLA folding and PLA partitioning[3, 5, 11, 18]. Based on this, Reference 19 describes a system which combines the state-assignment of Reference 3 with a method for minimizing the number of transition related product terms in a partitioned FSM. No system incorporates both delay and floorplan considerations. The target of these methods is to obtain maximal size reductions for a specific microarchitecture to which they are dedicated. However, some of the microarchitectures covered are also useful alternatives for MASCOT. Therefore, MASCOT incorporates several of the more general types among these optimization algorithms. Other recent synthesis methods[20, 21] perform useful decompositions or optimizations related to the previous methods but suffer from equal limitations. The first extensive comparison between control path microarchitectures was made[10] by means of reverse engineering. This comparison showed large differences in delay and size between the different microarchitectures. However, no synthesis method was included to generate efficient microarchitectures. At the logic level there are many successful systems[1]. We refer the reader to the literature. Several systems at this level are able to handle a delay parameter[6].

To conclude, microprogramming methods are outside the scope of control path microarchitecture synthesis. A number of FSM synthesis methods have been developed over the years. However, these are restricted to specific microarchitectures and focus only on the size minimization problem, which limits their applicability. At the logic level, current systems can handle delay parameters. Unfortunately, these systems do not incorporate floorplanning.

DESIGN DOMAIN

Before we discuss the MASCOT details, we define a frame of constraints and assumptions for the synthesis. In fact, this frame represents domain specific heuristic knowledge to limit the design space by cutting off the less feasible microarchitectures. This keeps the computational complexity manageable. The general part of the domain knowledge is discussed in this section. Details of specific decomposition phases are found throughout the remaining sections.

Finite state machine

The cardinality of a set of elements is denoted by a #. For example; #I is the number of symbols in the input alphabet. A binary coded symbol is called a word. A finite state machine is defined by the quintuplet FSM = <I, O, S, OUT, TRS> given by:

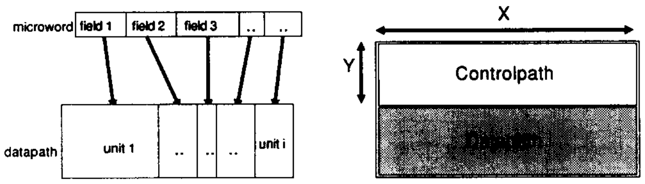


Figure 2 Control fields and general floorplan

$I = \{i_1..i_k\}$	Input alphabet where i_x is a binary coded symbol
$O = \{o_1..o_m\}$	Output alphabet where o_x is a binary coded symbol
$S = \{s_1..s_l\}$	State alphabet
O := OUT (*S*, *I*)	Mealy output function
O := OUT (*S*)	Moore output function
S := TRS (*S*, *I*)	Transition function

A second specification of a FSM is given by a set PTS $= \{pt_1..pt_n\}$ of four tuples <input word, state, next state, output word> denoted by $pt = < i, s, s', o >$ which are called product terms. A subset of PTS is the set of transitions TRS $= \{trs_1..trs_t\}$ formed by the tuples $trs = < s, s' >$. This *trs* denotes one edge in the state graph spanned by TRS. Therefore #TRS < #PTS; because of the binary coded input word more product terms *pt* may be needed to implement one transition *trs*. In addition to this we can write the transition function as TRS $= \{ptrs_1..ptrs_n\}$ where $ptrs = < i, s, s' >$ and the output function as OUT $= \{pout_1..pout_n\}$ where $pout = < s, o >$. This is the Moore type definition; for Mealy $pout = < i, s, o >$.

The output vector is divided into fields $o_i = < f_1 \ldots f_d >$. Each field corresponds with exactly one functional unit in the data path as shown in *Figure 2.* A control field contains the operational commands to control its data path functional unit. This ordering ensures a minimal wiring area for command lines in the control path.

Design objectives

To allow incorporation of a floorplan design objective in the microarchitecture synthesis we define a global floorplan. The global control path floorplan has a length *X* equal to that of the data path. Its desired height *Y* relates to the control path area *Ar* by $Ar = X * Y$. Thus, the floorplan dimensions *X* and *Y* define the control path bounding box by its shape $AsR = Y/X$ and total size *Ar*. For the delay specification we take a specification for the control throughput. The delay parameter specifies the period of time $D = t(o_i) - t(o_{i-1})$ between successive output words. The control synthesis is then free to select any clock scheme and this definition also hides, and therefore allows, internal pipelining of the control path.

Microarchitecture optimality

Finally we define the notion of optimality of a microarchitecture as the distance of the *X*, *Y* and *D* parameters of the microarchitecture to the *X*, *Y* and *D* goals defined by

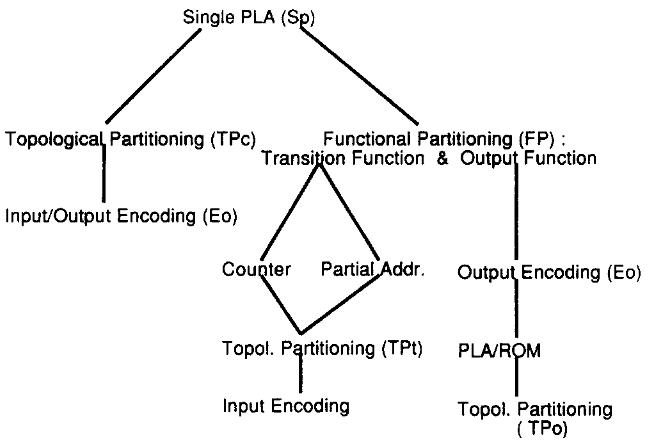

Single PLA (Sp)
Topological Partitioning (TPc)
Functional Partitioning (FP) : Transition Function & Output Function
Input/Output Encoding (Eo)
Counter
Partial Addr.
Output Encoding (Eo)
Topol. Partitioning (TPt)
PLA/ROM
Input Encoding
Topol. Partitioning (TPo)

Figure 3 Decomposition decision scheme

the user. If these goals cannot be met with the design actions incorporated, the microarchitecture with parameters closest to the goal parameters is the most optimal one. This is not necessarily identical with the smallest or fastest microarchitecture.

DECOMPOSITION PROCESS

This section discusses the design space with the microarchitecture alternatives considered by MASCOT. We developed the decomposition decision scheme of *Figure 3*. This scheme shows the search order for feasible decompositions. It will be explained in detail in the following sections. The philosophy underlying this scheme is that of gradual decomposition as discussed above. Thus, coarse decompositions such as topological or functional partitioning are closer to the root than the fine grain methods such as partial addressing or output encoding. The scheme is travelled from root to leaves and from left to right. MASCOT follows this scheme as far as needed to fulfil the design objectives, as far as possible. Where these objectives are not fulfilled after the scheme has been fully executed, a warning message is returned to the designer. MASCOT decides on microarchitecture alternatives by both analysis of the FSM structure and the order of alternatives in the scheme. This analysis concerns the transition or state graph structure, the cardinality of input alphabet #I, output alphabet #O, state set #S, transitions #TRS and product terms #PTS. The exact descriptions of the analysis are found throughout the remaining sections.

Behavioural description

The first action of the synthesizer is to compile a description of the control path behaviour into a FSM transition table. This behaviour can be specified in either a microprogram format or a PLA format by means of the set PTS of four tuples *pt*. The behaviour compiler replaces branch constructs in the microprogram, which are typical for sequential programming, by completely specified transition sets. Both a Moore type and a Mealy type transition table

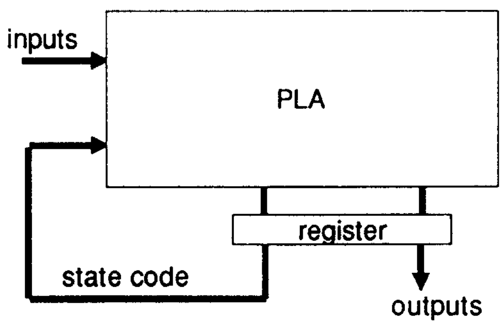


Figure 4 Single PLA microarchitecture

are constructed. For future versions we plan a compiler for the translation of processor architectures described in VHDL to FSM descriptions.

Initial configuration

The simplest possible microarchitecture (Sp) contains one PLA and a state register (*Figure 4*). The alternatives at this point in synthesis are either a Mealy type or Moore type of specification. These differ in the size of the combinatoric. For several reasons the Mealy specification was selected. It contains in the first place the lowest number of product terms ($\#PTS_{Mealy} < \#PTS_{Moore}$) as shown in *Figure 5*. Because $\#TRS_{Mealy} = \#S_{Moore}$, each transition can generate a unique output vector (thus :$\#TRS = \#O$), whereas in the Moore type a complete state is needed for each unique output vector. Hence, the Moore specification contains more states ($\#S = \#O$) and therefore more transitions and product terms. This difference is related strongly to the degree of connection of states in the state graph. This number is high for typical ISP processor state graphs. Secondly, the conceptual input-to-output delay is twice as large for Moore PLA combinatoric as for the Mealy PLA combinatoric. Therefore the Mealy specification is the default for the single PLA. Note that, due to the register on the output variables, the external behaviour is of the Moore type.

The system then checks the single PLA parameters with the floorplan and delay objectives. Before this check is performed, the size reduction caused by state-assignment and logic minimization is estimated. This estimation[22] is based on a polynomial curve fit on recent results of state-assignment algorithms[23]. The parameters in this curve are the number of output variables and the average number of product terms/state of the FSM. If the single PLA does not

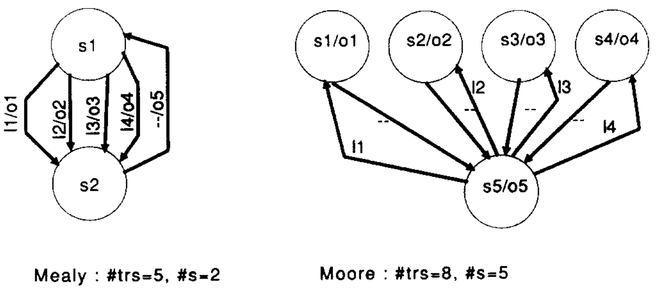


Figure 5 Mealy and Moore FSM specifications

fulfil the design objectives, the synthesizer goes to the next phase in the design path according to *Figure 3*. Otherwise it actually performs state-assignment[23] followed by logic minimization[2], which finishes the synthesis.

First level decomposition alternatives

This point in the scheme contains two decomposition alternatives: topological partitioning (TPc) and functional partitioning (FP). TPc selects strongly related input/output subsets and implements their subfunction in separate PLAs[11, 18]. This may require duplicate input/output variables or product terms, but reduces delay and area. The other alternative, FP, separates both functions of the FSM and implements them in separate PLAs[3, 19, 20]. One PLA implements the transition function and the other PLA implements the output function. By this, minimization of both functions becomes independent. Therefore we can apply methods dedicated to either the transition or the output function, which gives more flexibility for microarchitecture optimization. This is expressed in *Figure 3* by the two independent subgraphs under the FP node.

The selection between the partitioning alternatives is based on the distance between the PLA size and delay parameters and the required design objectives on delay and floorplan. In general, if size is no problem or in the case of relatively small machines, TPc is sufficient to correct a too large delay or a floorplan misfit. Therefore TPc is executed first by default. If the results are not accepted by the user the system proceeds with FP. A previously generated microarchitecture is not discarded by the system until a better one is generated. This guarantees that the optimal solution remains available in case later design actions do not deliver more optimal microarchitectures.

Topological partitioning

Several types of topological partitioning have been distinguished[11]. These differ in the way they allow multiple occurrence of input and output variables in the partitioned PLAs. We briefly describe the properties for this input and output augmentation of each type. A more detailed discussion is found elsewhere[11]. The TP types are:

- input partitioning
- output partitioning
- parallel partitioning.

Presume n sets of variables (PLAs) after partitioning and $\mathrm{SUM}_n(X_i)$ is the cumulative sum of a certain entity X_i over $i = 1 \ldots n$. Then input partitioning (TPi) partitions the PLA AND-plane and creates duplicate inputs. It disregards the output variables and is therefore applied best to the transition function of a functional partitioned FSM. The total number of inputs #I and product terms #PTS of the PLA collection are described by: $\#I <= \mathrm{SUM}_n(\#I_i)$ and $\#PTS = \mathrm{SUM}_n(\#PTS_i)$.

Output partitioning (TPo) partitions the PLA OR-plane and allows product term and/or output duplication. With product term duplication, this type provides only optimization for the single PLA microarchitecture if independent groups of output variables exist (for example outputs to different data paths). We do not apply the output duplication variant because it increases the output interconnect. The TPo variant is applied best on the output function after functional partitioning. Its characteristics are: $\#O = \mathrm{SUM}_n(\#O_i)$ and $\#PTS <= \mathrm{SUM}_n(\#PTS_i)$.

Parallel partitioning (TPc) allows both input and output duplicates and no product term duplicates. Because it regards inputs as well as outputs, this partitioning type is applied only on the initial single PLA. Its characteristics are: $\#I <= \mathrm{SUM}_n(\#I_i)$ and $\#O <= \mathrm{SUM}_n(\#O_i)$ and $\#PT = \mathrm{SUM}_n(\#PT_i)$.

We have previously presented[24] topological partitioning with constraints on floorplan and/or delay. In its original form[11, 18], TP optimizes only to a minimal cumulative area of the PLAs. Our version, called TPc, searches a solution in which a maximum to the number of product terms in a PLA exists, while retaining a maximal area reduction profit. This method therefore does not use the product term counts as a pure limit, but anticipates it by performing clustering of state-PLAs in two phases, both with different targets. The first phase concentrates on pure size optimization until the largest PLA grows to more than 80–90% of the product term limit. The second phase then clusters with respect to size equality of the PLAs. In fact, it tries to reduce the empty area in the circumscribing rectangle of the partitioned PLA collection.

At this first level of decomposition the parallel variant of TP is applied because it delivers the largest optimization to size and floorplan flexibility. Both other variants of TP are found in later stages of the design process.

Functional partitioning

Functional partitioning (FP) provides a separate PLA implementation for the transition and the output function. Therefore each PLA contains a different logic structure. The output function PLA has, for typical ISP control paths, a much higher number of output variables than the transition function PLA, because $\#O > {}^2\log \#TRS$. The reverse statement holds for the number of input variables, ${}^2\log \#TRS < \#I$. Thus, as stated before, each of both PLAs requires its own specific minimization methods. This provides for some FSMs a larger optimization potential than application of topological partitioning. There is a difference in area between an FP configuration and a single PLA. The number of output vectors is in the single PLA equal to the number of product terms and in the FP configuration equal to the number of transitions. This saves area in FP because $\#PTS/\#TRS \geqslant 1$. Furthermore, FP changes the floorplan fit problem and it enlarges the floorplanning flexibility of the configuration. However, FP does not always minimize the area of the configuration. The cause is the extra area needed for the input plane of the output function PLA.

Two alternatives to FP exist, one which has only a state code (*Figure 6*), and the other which has separate state and transition codes (*Figure 7*).

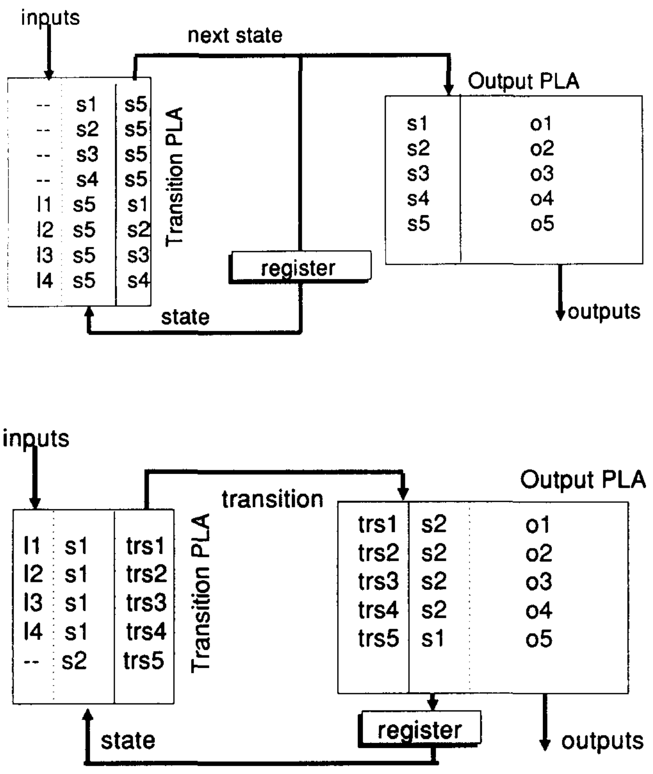


Figure 7 Functional partitioning with double code

Single code: $ptrs = < i, s, s' >$ and $pout = <, s, o >$
Double code: $ptrs = < i, s, trs >$ and $pout = < trs, s', o >$

Figure 7 illustrates the double code FP microarchitecture. The left PLA takes the inputs and the state codes and generates transition codes. The state code is input to the transition PLA. In the single code version, this state code is also input to the right PLA which generates the appropriate output vector. In the case of the double coded machine the right PLA generates the next state code and the transition PLA generates a transition code which selects the output vector. Note that the number of transition codes in the double code version is equivalent to the number of state codes in the single code version. Thus $\#TRS_{double} = \#S_{single} = \#S_{Moore}$. This is also equivalent to the state set cardinality in the Moore type specification. Also, $\#S_{double} = \#S_{Mealy}$. The double coded FP microarchitecture has for this reason the smallest transition PLA and is the default.

Optimizations for the FP microarchitecture

We implemented three optimizations for the FP microarchitecture, two relating to the transition function and the third to the output function:

- application of a counter
- application of partial addressing
- output encoding (per field or complete).

The incorporation of a counter removes the countable transitions from the transition function PLA, as explained in the next subsection. Partial addressing splits the next state code into two sections, by which the largest part is common to all next states for a state. This reduces the number of output variables of the transition PLA. Furthermore, equal 'forks' of transitions merge and reduce the number of product terms as explained below. Output encoding can reduce the area of the output function PLA by replacing sparse codes by more dense codes followed by a decoder. This is also explained briefly below. Input encoding adds an encoder before the PLA to reduce the width of its AND-plane. This is presently not implemented because it is in general not very effective for instruction set processor control paths, due to the usually very dense encoding of instructions.

Impact of a counter

The application of a counter in a functional decomposed FSM has been studied extensively[3]. The counter implements the countable transitions, which makes these transitions redundant in the transition PLA and therefore removable. The PLA generates the signal for counter activation by default in case an input pattern does not match any of its product terms. The counter must be loadable to accept codes generated by the PLA in case of non-countable transitions. The counter is implemented on the transition function output, and thus for the double code FP microarchitecture only the TRS code is affected. There are no side effects for the state or S code.

The optimization coupled to this implementation is the coding of transitions such that the countable transitions cover the highest possible number of product terms (PT). Thus, the algorithm must identify a collection of transition chains which cover the maximum number of product terms. Such an algorithm is given in Reference 3. This algorithm is dedicated to Moore type single coded machines and also incorporates other constraints for the code assignment, which make it less suitable for our purpose. Therefore we developed our own algorithm.

Countable transition chain Ct_i:

$$Ct_i = trs_1..trs_k \mid \mathrm{Code}(trs_{i+1}) = \mathrm{Code}(trs_i) + 1$$
$$\text{where } 1 \leqslant i \leqslant k$$

The total number of product terms implemented by a counter is then:

$$\mathrm{PTSC} = \sum_{i=1}^{n} \#\mathrm{pt}(\mathrm{Ct}_i)$$

where n is the number of countable chains.

The objective of the algorithm is to code TRS such that PTSC is maximal. Therefore all transitions have a weight assigned representing their number of product terms: $w_i = \#\mathrm{pt}(\mathrm{trs}_i)$. The algorithm first removes all transitions whose start state is equal to the next state, because transition cycles are not countable. Then the transitions are sorted according to decreasing weight. The algorithm then builds chains of transitions by taking the first unmarked transition and scanning the list for follow-up transitions from the head. Thus the transitions with the highest weight are selected first. This process proceeds until a cycle is met or no follow-up transition exists. The state which is the head of the chain is marked. With this way of marking a state may occur in more than one chain, but occurs only once as the head of a chain. This redundancy avoids local optima. New chains are generated as long as there are unmarked states which have unmarked successors. Then the chains are sorted according to their cumulative weights, in decreasing order. To get the final collection of countable chains, the algorithm takes chains from the head of the list. Chains containing states that also occur in already selected chains are rejected.

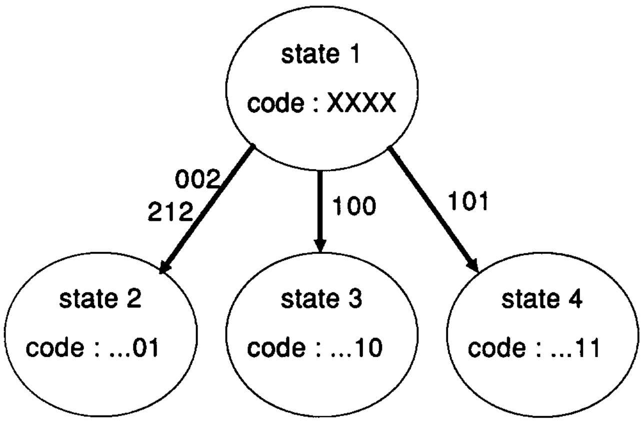
The algorithm codes the states according to the chains selected. The algorithm is restarted and iterates until no improvement occurs in the PTSC. *Table 1* shows the results for this algorithm on the MCNC FSM benchmark set. Column d.Cntrl shows the number of countable product terms before optimization, and d.Cntr2 after optimization.

Partial state codes

Partial state codes or addressing is not a common optimization type in control path synthesis systems. Many manual designs of processor control paths[25], however, apply this method to implement state transitions. It reduces the

Table 1 Improvements in countable transitions

	PT	d.Cntr1	d.Cntr2
bbara	60	11	15
bbsse	56	8	11
bbtas	24	10	11
beect	28	4	7
cse	91	4	14
dk14	56	5	7
dk15	32	3	5
dk16	108	11	26
dk17	32	7	9
dk27	14	3	5
dk512	30	7	12
donfl	96	7	13
ex1	138	3	21
ex2	72	6	18
ex3	36	3	9
ex4	21	11	13
ex5	32	4	8
ex6	34	3	9
ex7	36	6	10
keyb	170	12	44
lion	11	3	3
lion9	25	6	6
plan	115	6	50
s1	107	6	23
sand	184	2	82
scf	166	31	120
shift	16	4	8
styr	166	8	56
tav	49	2	23
tra04	14	5	6
tra11	25	1	8



Example 1 Local code difference for next states

number of product terms in case a number of states have identical transition structures. The transition function generates just a part of the next address (state code), often only the two least significant bits (LSB). The output function generates the other bits (MSB) of the next address. By this, these most significant bits (MSB) must be identical in all the next states of a certain state. This state code generation is feasible because states usually have only a small number of next states, as shown in *Figure 8*. This makes it possible to code the local selection of the next state in a few bits. Then the MSB bits of the state code can be equal among all the next states of a state. This makes it possible to generate the MSB part from the output function. If several states have identical sets of input vectors and transitions, they can share a set of product terms in the transition function. This reduces the number of product terms and output variables in the transition PLA. Note the cost of additional output variables for the MSB bits in the output function. The optimization problem is to find the maximal number of compatible transition sets. Transition sets are compatible when all inputs are identical and the sets of next states contain equal subsets of states. Furthermore these sets must contain each state only once to prevent multiple codes for a state. Note that when not all states are partially codable, the transition function decomposes into two PLAs (*Figure 8*), of which one generates the normal complete next state code for all state transitions which cannot be implemented by partial addressing.

In *Example 2* the transition set of state 7 is not identical

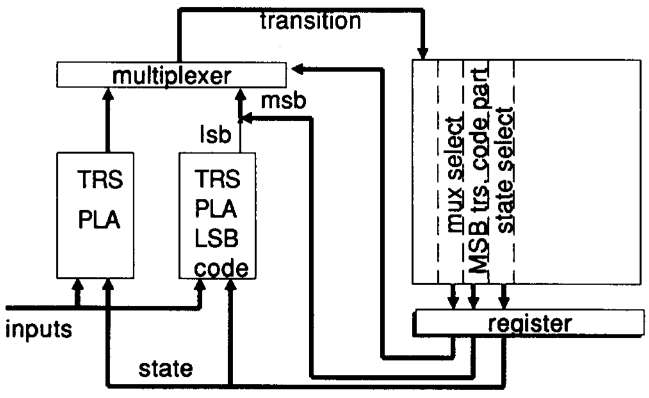
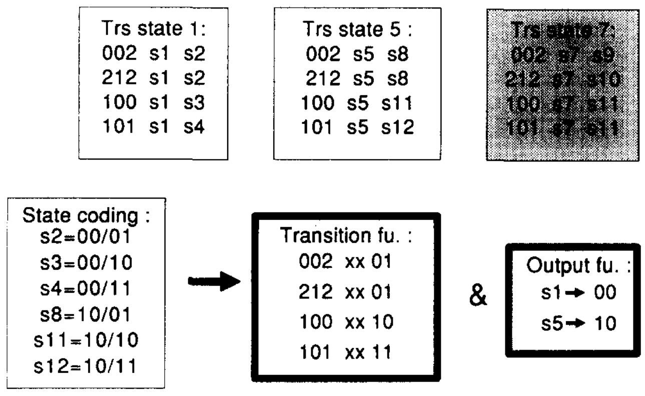


Figure 8 Partial address configuration



Example 2 Product term reduction by partial codes

with the other two sets because its next state structure has a different pattern. Both the input word set and the transitions must be identical as is the case for states 1 and 5 in *Example 2.* The state coding shows the reduction in product terms of the transition function; the transitions of states 1 and 5 are merged. Note that the transitions from state 7 cannot be implemented in the same PLA as the merged set of states 1 and 5 (*Figure 8*). The output function generates the static code part; in the case of state 1 it generates 00, of state 5: 10, and for state 7 this code is don't care: 22.

State detection

The next algorithm detects the number of redundant product terms in a FSM for a *d* bits wide LSB part of the state code. First it removes all transitions starting from states which have more than 2***d* bits of next states. Then it generates maximal clusters of states with identical input vectors. From these clusters the algorithm makes new ones which contain the next state collections in a matrix form with one row of next states for each state. For partial addressing a cluster should contain only unique states. If not, a maximal unique sub-matrix must be searched. The algorithm performs this search by discarding non-unique states one by one until a unique cluster remains. The reduction in transitions is then obtained by counting the number of states in the unique submatrix minus one row, because one row is actually implemented. From this follows the number of redundant product terms.

This method is best suited for control functions with a few transitions per state and disjoint next-state sets. Microprogramming methods[25] use that knowledge extensively to generate the control path function which can be implemented efficiently with this technique.

Output encoding

Encoding reduces the width of the output PLA in the case of sparse output codes[25, 26]. Two types of output encoding are incorporated. The first is field encoding, an algorithm to detect groups of output variables for which encoding gives a maximal reduction of size. To prevent area losses by extra wiring, the output variables remain ordered according to the physical terminal ordering in the data path. The second encoding type is vertical encoding, which encodes the complete output variable set. Note that the field encoding algorithm can also come up with the vertical coding solution. Both encoding types are executed and the one which gives microarchitecture results closest to the design objectives is selected. In the next section we present further decomposition of the optimized and functional partitioned microarchitecture.

Decomposition of the FP configuration

In case the optimized functional partitioned microarchitecture does not fulfil all design objectives on floorplan and delay, MASCOT continues with further decompositions. Now MASCOT applies the input and output partitioning variants of the TP algorithm[24] on the transition PLA or output PLA. The function with the largest area is partitioned first. This is mostly the output function. In case the output function fulfils the delay requirement, the system checks if the floorplan misfit (if any) can be removed by application of a ROM instead of the PLA. If this is not possible, TPo is applied.

Output PLA partitioning

All forms of topological partitioning in this and the next subsection are of both the floorplan and delay constrained type. Only two variants of topological partitioning are useful for the output function. The input partitioning variant is not suitable because of the few input variables to the output function (only the transition code). We then have to select between either an increase in the number of product terms, without output augmentation, or extra interconnect where output augmentation is allowed. The product term augmentation variant of TP (TPo) is the most effective here, because the number of input variables is small.

But before TPo is applied and only where there is no delay problem, MASCOT tries to meet the floorplan requirement by checking a ROM structure. The ROM is more flexible in its aspect ratio than a PLA because it can contain more columns of output words due to its regular input decoder. The system finds the optimal aspect ratio of the ROM by enumeration. Currently, this aspect ratio is open towards state-assignment. Therefore the enumeration includes only ROM column numbers which are a power of 2. Other column numbers cause missing state codes in the ROM address range, and thus give restrictions on the state codes. The aspect ratio of the ROM is thus variable with a factor of 4. Also, in the case of vertical output encoding, the system enumerates all aspect ratios of the ROM pair. The pair of ROMs always has such a topology that the decoder ROM borders the data path.

Transition PLA partitioning

If, after output PLA partitioning, the floorplan and/or delay objectives are still not fulfilled, the system also partitions

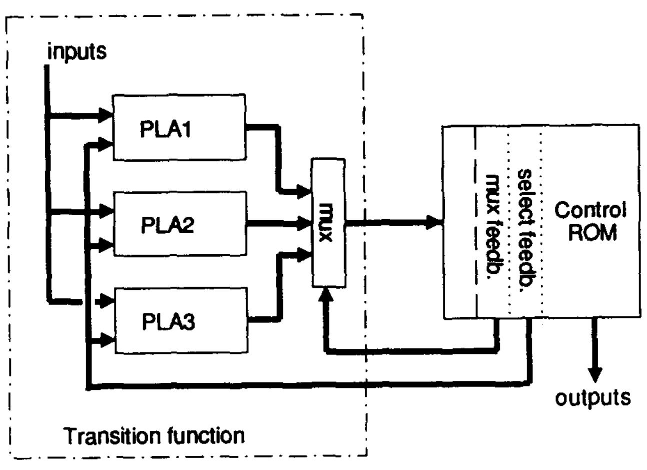


Figure 9 Partitioning for floorplan aspect ratio

the transition PLA. The transition PLA is split by TPt into smaller PLAs, which are multiplexed to the next-address register (*Figure 9*). The multiplexer takes a part of the state code input from the transition PLAs. To realize this, the state code feedback from the output ROM/PLA is split into two separate codes (*Figure 9*). One code controls the transition PLA multiplexer and the other is input to the transition PLAs. Together, both codes represent the original state code. The advantage of this is twofold: the delay of the transition function decreases and it gives a better fit to the floorplan. The number of transition PLAs determines the width of both codes. Thus, the floorplan shape has an impact on these codes and therefore on the state coding. The only logic constraint is that all transitions from one state have to be implemented in one PLA. This limits the maximal multiplexer width to the state set cardinality.

For this partitioning we use TP input partitioning with floorplan and delay constraints (TPt)[24]. This algorithm has the option to cluster transition sets of all states with equal input sets. This clustering reduces computation time, therefore it is the default. If the floorplan fit is not improved by clustered TPt, the system performs the next run without this clustering. This partitioning is likely to save area because the input set of an instruction set processor control path usually contains several input variable groups. For example, the instruction fields are such groups. These occur when, for example, the address modes and the opcodes are coded in separate fields or several instruction formats exist.

FLOORPLAN

The selection of FP decomposition influences the floorplan aspect. The combination of both PLAs has to meet the floorplan requirement. Therefore the initial FP floorplan area is divided simply in two sections as shown in *Figure 10*. Furthermore, the output function PLA/ROM is always positioned vertically, so that the often large number of control lines to the datapath are as short as possible. This minimizes the area needed for wiring. The floorplan is obtained by generation of a binary tree with the PLAs as leaves. This slicing tree leads to a shape function[12,13] of the

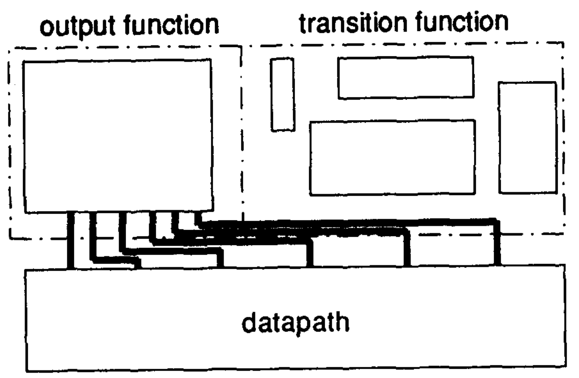


Figure 10 FP floorplan

floorplan, from which the point closest to the goal *X*, *Y* is taken. The floorplan subdivision of *Figure 10* is then represented by generation of two independent binary trees, which are clustered at the highest level.

The dimensions and delay of PLAs and ROM were obtained from PLA and ROM generator data sheets[27] for a 1.0μm process. By changing these functions, the system can easily be adapted to emerging technologies.

RESULTS

This section gives the results for some of the larger machines of the MCNC FSM benchmark set[7] and two in-house control paths. Their data is listed in *Table 2*. The complexity of eu1 or s80 is comparable with that of the Intel 8080 control path, from which the latter is derived. The results of this system can be expressed in several ways. Note that area or delay minimization is not a target of the system itself. The user determines the goal *X*, *Y* and delay values, and thereby if area minimization is needed. The target of the system is to generate a microarchitecture with *X*, *Y* and delay parameters as close as possible to the given goal parameters.

Therefore the flexibility of MASCOT's strategy can be best shown by defining different goals for one of the machines. This is done for machine scf in *Table 3*. This table contains eight different goals characterized by aspect ratio ($=X/Y$), denoted as AsR, and area ($=X * Y$), denoted as Ar, of the floorplan; also the delay value (*D*) is varied. This leads to the eight rows in *Table 3*. For the first two rows area and delay are equal to the initial configuration, and only the aspect ratio of the floorplan is varied. Then, the delay is reduced to 50% and enlarged to 200% of the

Table 2 Machine data

Machine	scf	ex1	planet	sand	bbsse	eu1	s80
#Inputs	27	9	7	11	7	11	17
#Outputs	56	19	19	9	7	25	40
#Products	166	138	115	184	56	91	210
#States	121	20	48	32	16	50	62
X_initial	766	318	313	280	210	391	546
Y_initial	1379	1178	987	1541	467	791	1766
D_initial	39	32	28	39	17	24	45

original, again for both aspect ratios. At last the area is reduced to about 66% of the original, again for both aspect ratios.

Table 3 shows, besides the *X/Y/D* triple obtained, the number of PLAs in the microarchitecture. The / denotes separation of the count for transition PLAs from the output PLAs. The design actions are also given in the shorthand notation of *Figure 3*. From *Table 3* we conclude that TPc is a rather powerful tool, because it adapts well to different aspect ratios. More complex microarchitectures are generated when delay must be reduced or size reduction is wanted. This is shown by the 4th, 7th and 8th rows in the table. Except for delay, MASCOT reached the goal in all cases. Apparently a delay reduction of 50% cannot be obtained for this machine within the initial area.

Table 4 shows microarchitecture *X/Y/D* triples for several points in the decision scheme. In this case the synthesis was forced to scan the complete scheme. The goals were defined so that minimizations were needed. The different columns give results for different positions in the decision scheme. The column denoted by +TPt includes the FP and counter (cntr) design actions; this occurs also for +TPo which includes TPt. The last two columns show which microarchitecture was considered optimal. OptKol gives the design actions or the position of the optimal microarchitecture in the scheme and #PLA again gives the number of PLAs for the microarchitecture. One can see that for the larger machines more complex microarchitectures are found. This is due to the relatively low impact of PLA control and power overhead area on the total area in the case of large machines. For small machines this overhead area prevents solutions with many PLAs.

Table 3 Different goals for machine scf

AsR, D, Ar	*X/Y/D* goal	*X/Y/D* obtained	#PLA	Des.Actions
1.0, 1x, 1x	1028/1028/39	990/1010/27	2	TP
0.2, 1x, 1x	2298/460/39	1710/450/27	15	TP
1.0, 0.5x, 1x	1028/1028/20	1030/1020/27	3	TP
0.2, 0.5x, 1x	2298/460/20	2280/460/24	15/16	FP/TPt/Eo/TPo
1.0, 2x, 1x	1028/1028/80	990/1010/27	2	TP
0.2, 2x, 1x	2298/460/80	1710/450/27	15	TP
1.0, 1x, 2/3x	840/840/39	780/830/38	3	Eo/TPo
0.2, 1x, 2/3x	1877/376/39	1870/380/27	1/15	FP/CNTR/TPo

X, Y in μm, *D* in ns

CONCLUSIONS

The MASCOT control path synthesis system has great flexibility in floorplan and delay constraints in comparison with other control path synthesis methods. Different goals in the design space for a machine can be met with this synthesis strategy. For microarchitecture optimization it includes a number of known methods from both FSM synthesis and microprogramming. The floorplan and delay requirements determine the decomposition. The user defines the goal and is therefore able to explore the design space of the control path.

Currently, MASCOT is able to synthesize PLA or ROM microarchitectures. In future versions we plan to incorporate cell based implementations as well. The greatest problem with such implementations is the estimation of the interconnect area. Also the translation of processor architectures described in VHDL to minimal control path finite state automata or machines requires further research.

REFERENCES

1 Brayton, R K 'Algorithms for multi-level logic synthesis and optimization' in DeMicheli, G and Sangiovanni-Vincentelli, A (Eds) *Design Systems for VLSI Circuits* Nyhoff (1987) pp 197–248

2 Brayton, R K *et al. Logic Minimization Algorithms for VLSI Synthesis* Kluwer (1984)

3 Amann, R 'Algorithmische entwurfsverfahren fuer kombinierte pla/rom-steuerwerke unter verwendung van zaehlern' *Dissertation*, VDI Verlag, Düsseldorf (1987)

4 DeMicheli, G 'Optimal state assignment for finite state machines' *IEEE Trans. CAD* Vol 4 (1985) pp 269–284

5 Devadas, S and Newton, R 'Exact algorithms for output encoding, state assignment, and four-level Boolean minimization' *IEEE Trans. CAD* Vol 10 No 1 (January 1991) pp 13–27

6 Bartlett, K *et al* 'Synthesis and optimization of multilevel logic under timing constraints' *IEEE Trans. CAD* Vol 5 No 4 (October 1986) pp 582–595

7 Lisanke, R 'Introduction of synthesis benchmarks' Int. Workshop on Logic Synthesis, North Carolina (1989)

8 De Man, H *et al* 'Architecture driven synthesis techniques for mapping digital signal processing algorithms into silicon' *Proc. IEEE* Vol 78 No 2 (February 1990) pp 319–336

9 Lippens, P *et al* 'PHIDEO: a silicon compiler for high speed algorithms' Proc. EDAC '91, Amsterdam (February 1991) pp 436–441

10 Obrebska, M 'Efficiency and performance comparison of different design methodologies for control parts of microprocessors' *Microprocessing Microprogr.* Vol 10 (1982) pp 163–178

11 DeMicheli, G and Santomauro, M 'Smile: a computer program for partitioning of programmed logic arrays' *Comput. Aided Des.*, Vol 15 No 2 (March 1983) pp 89–97

12 Otten, R 'Automatic floorplan design' in Proceedings of the 19th Design Automation Conf., ACM/IEEE (1982) pp 261–267

13 Stockmeyer, I 'Optimal orientations of cells in slicing floorplan designs' *Inform. Control* Vol 57 (June 1983) pp 91–101

Table 4 Several points in the decision scheme

Machine	*X/Y/D* goal	TPc	FP/cntr	+TPt	+TPo	OptKol	#PLA
scf	600/600/30	1070/730/24	920/1190/50	780/1190/47	1020/670/25	+TPo	2/15
sand	500/500/25	560/500/13	430/960/42	430/960/37	430/920/36	TPc/Eo	4
plan	400/400/20	590/590/14	490/830/41	890/420/35	1070/410/34	TPc	3
ex1	500/300/25	690/560/12	890/470/41	920/440/37	1130/400/35	Sp	1
bbsse	200/150/15	400/180/10	330/240/23	490/290/21	490/290/18	Sp	1
eu1	500/500/20	500/620/17	560/600/33	490/600/30	500/590/30	TPc	2
s80	700/700/35	1140/680/21	710/1610/67	760/1610/61	1280/930/43	TPc/Eo	8

14 Gessner, J *et al* 'Synthesis of control units in a design environment for chip architecture' *Microprocessing Microprogr.* Vol 27 (1989) pp465–472
15 Sheraga, R J and Gieser, J L 'Experiments in automatic microcode generation' *IEEE Trans. Comput.* Vol 33 No 6 (1983) pp557–569
16 Davidson, S *et al* 'Some experiments in local microcode compaction for horizontal machines' *IEEE Trans. Comput.* Vol 30 No 7 (1981) pp460–477
17 Tredennick, N 'The cultures of microprogramming' Proc. 15th Annual Workshop on Microprogramming – Micro-15 (1982) pp79–83
18 Henessy, J 'Partitioning programmable logic arrays summary' IEEE Proc. Int. Conf. on Computer Aided Design (1983) pp180–181
19 Mahler, H *et al* 'Processor control part synthesis using effective partitioning algorithms' *Microprocessing Microprogr.* Vol 23 (1988) pp 33–36
20 Paulin, P G 'Horizontal partitioning of PLA-based finite state machines' Proc. 26th ACM/IEEE Design Automation Conference (1989) pp333–338
21 Tarroux, G *et al* 'Optimization of micro-controllers by partitioning' Proc. EDAC 1991, Amsterdam (February 1991) pp368–373
22 Berg ten, A 'Curve-fitting of state-assignment minimization for PLAs' '*Memoranda Informatica* INF-91-44, Twente University, Enschede (1991)
23 Villa, T and Sangiovanni-Vincentelli, A 'NOVA: state assignment of finite state machines for optimal two-level logic implementation' *IEEE Trans. CAD* Vol 9 No 9 (September 1990) pp905–924
24 ten Berg, AJWM 'Floorplan optimized topological partitioning of programmed logic arrays' in Saucier, G and Trilhe, J (Eds) '*Synthesis for Control Dominated Circuits* IFIP Trans. **A-22,** Elsevier Science, Amsterdam (1993) pp 399–412
25 Husson, S S *Microprogramming Principles and Practices,* Prentice Hall (1970)
26 Stritter, S and Tredennick, N 'Microprogrammed implementation of a single chip microprocessor' Proc. 11th Annual Microprogramming Workshop (1978) pp8–16
27 *Solo 2000 Family Libraries,* European Silican Structures Limited (1990)

Ir. AJWM ten Berg completed his study in electrical engineering at the University of Twente in 1983. From 1983 to 1986 he worked at Philips in Nijmegen, the Netherlands on the development of routing and placement software for I2L IC technology. From 1986 to 1988 he was at Stork Brabant, Boxmeer, the Netherlands on the development of a management information system for use in textile finishing plants. In 1988 he joined the Faculty of Informatics at the University of Twente in the position of assistant professor in the field of computer architecture. Currently, his interests are architecture synthesis of digital systems, the application of AI methods in synthesis methods and the formal description of digital designs and synthesis processes.