FSM Partitioning and Synthesis Targeting an Embedded Autonomous FSM

Abstract: *This paper concerns the synthesis of complex finite state machines (FSM) by a novel partitioning and encoding approach. The target architecture is a generalization of FSM implementations with embedded loadable counters. Starting with a subgraph extraction constraints driven partitioning generates three parts, a sequencing, a command and an autonomous logic block. By solving the encoding problem simultaneously for all blocks the total area of the partitioned circuits is minimized. Experimental results demonstrate the efficiency of the proposed approach.*

Keywords: Finite state machine, autonomous automaton, partitioning, state encoding

1 Introduction

Starting at a high level description synthesis programs produce large finite state machines (FSM) as a RT-level specification of the control part. The controller is now synthesized and tuned to a suitable target architecture [KNRR88, ZSRM90, GeSa92, Sauc93, DeMi94]. In this paper a novel partitioning approach is proposed for the synthesis of complex finite state machines.

A number of FSM partitioning methods have been published concerning algebraic [HaSt66], general [Deva89, AsDN92,YaOI93] and linear [Putk91, JoKo91, BaBr93, KGFF94] partitioning which are based on different communication structures of the subFSMs. In a different manner of partitioning a PLA-counter architecture is used [AmBa89]. A subgraph is extracted there by splitting off unforked chains of states which are realized by an embedded loadable counter. The remaining subgraph forms the so- called sequencing PLA. In [Paul89] this approach is applied to a generalized sequencer architecture. Using a special associative programmable array structure in [KoWa93] the design costs are drastically diminished but the realization costs are insignificantly enlarged in comparison with the conventional approach.

The partitioning approach of this paper based on a target architecture using an autonomous automaton and PLAs is a generalisation of FSM partitioning with embedded loadable counters [AmBa89]. The partitioning by splitting off an autonomous automaton and the simultaneous state encoding of the communicating subFSMs allows us to exploit an enlarged optimization potential and leads to significant improvements of the synthesized structures.

The method starts with a subgraph extraction algorithm using a modified state transition graph generated from the given state transition table. Subsequently, constraints driven partitioning separates an autonomous automaton and a remaining FSM. This results in generating three symbolical descriptions corresponding to a sequencing, a command and a autonomous logic block. The functional models are transformed to a special common state representation. This enables us to solve the encoding problem for all logic blocks simultaneously. Thus the following logic optimization minimizes the total costs. Experimental results demonstrate the efficiency of the proposed approach.

The paper is organized as follows. In the next section we discuss the target architecture. Section 3 describes the partitioning method. The simultaneous state encoding and logic optimization is outlined in section 4. The approach is illustrated by a small example. Finally, section 5 underlines the advantages of this approach by experimental results.

2 Counter based Architecture and Extension using Autonomous FSM

By adapting the controller structure to properties of the state transition graph we can exploit an increased optimization potential for the FSM realization. One encoding and optimization approach based on the replacement of the state register by a loadable counter (Figure 1a) was proposed in [AmBa87]. The state transition function is realized there by the sequencing block cooperating with a loadable counter. The output function is separately implemented by the command block. A PLA cost reduction can be achieved because each state transition, which can be carried out by counting, saves one or more rows of the original sequencing PLA. While counting mode the signal load equals zero and the next state outputs are „don‘t care“. The counter based FSM realization is well suited for state transition graphs with a relatively small number of forks. Taking into account the restricted encoding constraints [DeMi85] for assignment of states covered by the counter and of states remaining in the sequencing block a considerable PLA cost reduction is possible [AmBa89].

We replace the loadable counter (Figure 1a) by an autonomous block (automaton) containing multiplexor, register and PLA (Figure 1b). This leads to an increased optimization potential. The autonomous block which is independent of input signals corresponds to a subgraph of the given state transition graph which contains not only unforked chains of states but also arbitrarily encoded state chains, joins and selected loops. To exploit the potential of this structure a novel design approach is presented in the next sections.

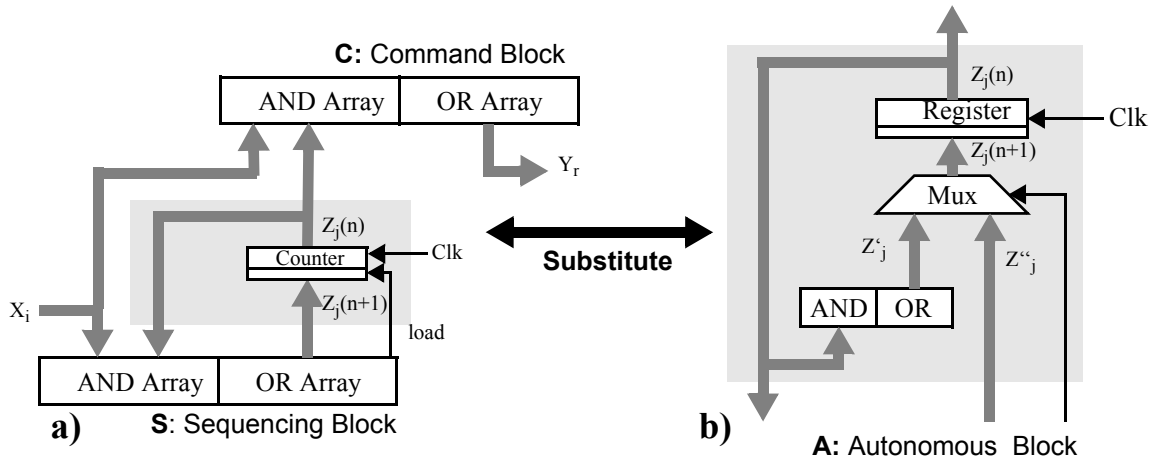


Figure 1. Structural approach using autonomous FSM instead of loadable counter

3 A Partitioning Approach based on Autonomous FSM Model

3.1 Preliminary Definitions

We recall some basic definitions helpful for the subsequent problem formulations.

(1) A finite state machine (FSM), M, is a tuple (X, Y, Z, f, g, Z^0) where X is a finite input alphabet, Y is the output alphabet, Z is a finite set of states, f is the transition function defined by $X \times Z \rightarrow Z$, Z^0, $Z^0 \subseteq Z$, is the set of initial states, g is the output function, where M is called Moore and Mealy FSM if g: $Z \rightarrow Y$ and g: $X \times Z \rightarrow Y$, respectively.

(2) An autonomous FSM, A, is a FSM, where $X = \varnothing$, f: $Z \rightarrow Z$, and g: $Z \rightarrow Y$.

(3) The state transition table (STT) is a tabular representation of M. Each row can be described as a 4-tupel, $(in_i, ps_i, ns_i, out_i) \mid i \in \{1,2,...,k\}$, where k is the number of rows, in_i and out_i are assignments of the input and output variables, respectively, ps_i and ns_i are symbolic present state (ps) and next state (ns), respectively.

(4) The directed graph G(V,E,W), called state transition graph (STG), is a graphical representation of M, where V, V=Z, is the set of vertices, E, $E \subseteq Z \times Z$, is the set of edges $e_i = (ps_i, ns_i)$ and $W = \{(in(e_i), out(e_i)) \mid e_i \in E\}$ is the set of weights of the edges.

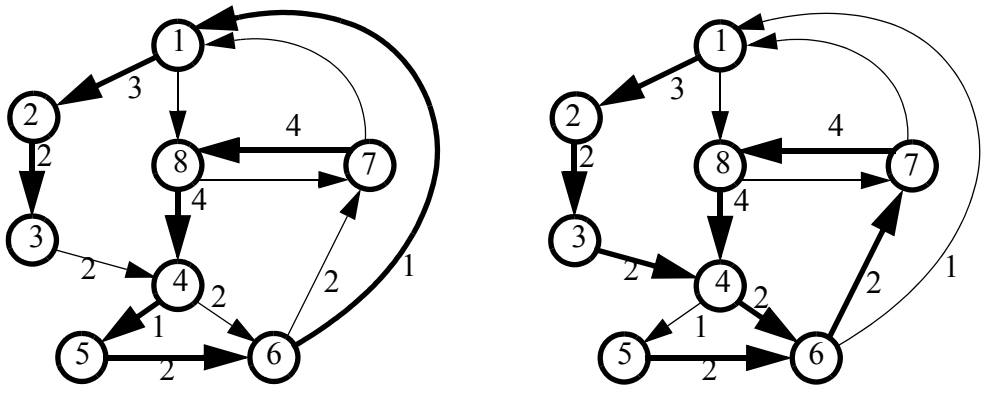
(5) State z_i is called predecessor of z_j, $z_i = Pre(z_j,)$ if $(z_i, z_j) \in E$ and successor of z_j, $z_i = Post(z_j)$ if $(z_j, z_i) \in E$.

3.2 Problem Formulation and Algorithmic Solution

The intended subgraph extraction corresponds to a cut through nodes of the given STG. Partitioning by splitting off an autonomous automaton aims at finding a bipartition of the given FSM description which minimizes the overall hardware costs of the communicating logic blocks. The partitioning problem is formalized as follows. We start with the unweighted Graph G = (V,E) derived from the given STT. $G'(V', E')$ with $V' \subseteq V$ and $E' \subseteq E \cap (V' \times V')$ is a subgraph of G. We investigate the set Γ of all subgraphs G' with $\forall z(z \in V' \rightarrow |Post(z)| \leq 1)$, i.e. each vertex of G' has at most one successor. Let r(e) be a weight of the edge e, $e \in E$ where r(e) is used to describe the potential decreasing of the costs of the remaining (sequencing) subFSM by splitting off this edge e. The weight for a whole subgraph G' is given by $R(G'(V', E')) = \Sigma(r(e) \mid e \in E')$. The partitioning task consists in:

Find a Subgraph G', $G' \in \Gamma$, with a maximal weight $R(G'(V', E'))$

The depicted algorithm *Subgraph* generates successively by local decisions a subgraph which approximately satisfies the above partitioning task. The measure r is based on heuristics proposed in [AmBa89] for selecting state chains. We extend the partitioning procedure and r with regard to splitting of an autonomous automaton involving subtrees (joining edges) and loops. As an example, the emphasized subgraph in Figure 2b holds these properties.
We compare the obtainable subgraphs for the two different architectures of Figure 1a and b. The emphasized subgraph Figure 2a of a given STG is computed for the case of an embedded loadable counter structure Figure 1a. The subgraph in Figure 2b for an embedded autonomous automaton contains a bigger number of edges. Therefore and the number of products covered by the autonomous automaton is enlarged in this example from 17 to 21. Additionally each join (3,4),(8,4) and (4,6),(5,6) can be covered by one product of the autonomous block.



a) Subgraph (emphasized) for the loadable counter architecture

b) Subgraph (emphasized) for the autonomous automaton architecture

Figure 2. Comparison of the obtainable subgraphs for the two different architectures of Figure 1a and b

Algorithm *Subgraph*(G):

$G = (V, E, R)$; $V' = \emptyset$; $E' = \emptyset$;
forall($z \in V$ **){**
  if ($|S(z)| > 1$ **) {** **// branch**
    find $z' \in S(z)$ **| such that**
      forall $(z'' \in S(z) \setminus \{z'\} \rightarrow r(z, z'') \leq r(z, z'))$);
    $V' = V' \cup \{z, z'\}$;
    $E' = E' \cup \{(z, z')\}$;
  }
  elseif $|S((z))| = 1$ **{** **// chain**
    $z' = S(z)$;
    $V' = V' \cup \{z, z'\}$;
    $E' = E' \cup \{(z, z')\}$;
  }
  else ;
}
$G' = (V', E')$; **// autonomous block**
$G'' = (V'', E'') | E'' = E \setminus E'$ and $V'' = \bigcup(\{z, z'\} | ((z, z') \in E''))$; **// sequencing block**

The implemented algorithm *subgraph* [FeMu93] uses two main heuristics influencing the edge selection as follows:

Heuristic A: Give preference to the encoding constraints of the sequencing block.

Heuristic B: To minimize the autonomous block give preference to joining edges with respect to face embedding constraints [DeMi85].

Figure 3 illustrates the proposed approach for a small FSM example (STT in Figure 3a used in [AmBa89]). The selected subgraph G′= ({1,2,3,0},{(1,2),(2,3),(3,0),(0,0)} forms the autonomous block A (Figure 3c) and saves rows g and i of the sequencing block, includes self loop b and takes into account the face encoding constraint (3,0) as option for the subsequent simultaneous state encoding.

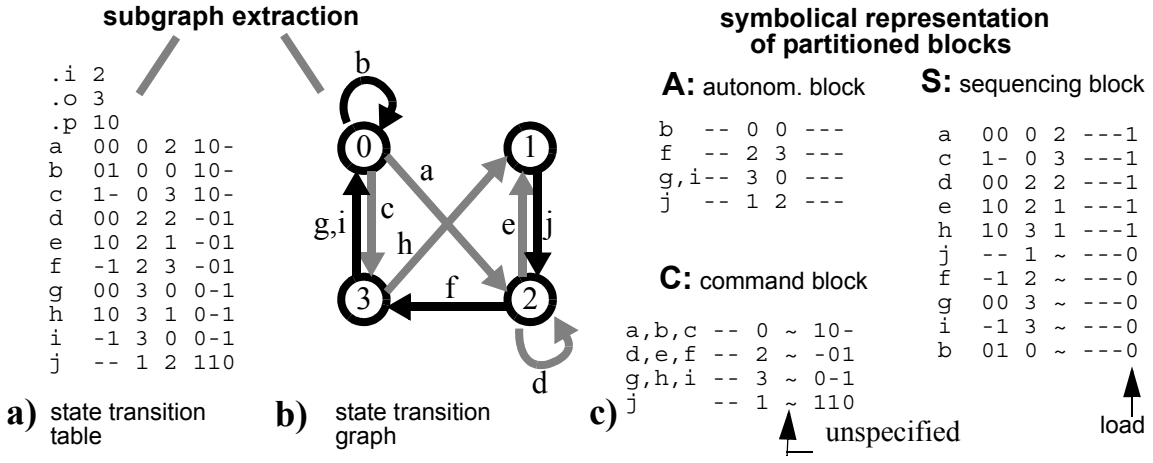


Figure 3. Illustration of subgraph extraction

4 Simultaneous State Encoding and Logic Optimization

4.1 Common State Encoding of Partitioned Blocks

The structural approach proposed above implies the property that the sequencing block S (Figure 1 and Figure 3c), the autonomous block A and the command block C are fed by one common register. Therefore the state encoding of partitioned blocks influences the needed area for combinational logic significantly. For minimizing the overall cost we have to solve the following state assignment problem:
There are given the FSMs S, A and C with the state sets Z_S, Z_A and Z_C, respectively. The state sets Z_S, Z_A and Z_C are identical or at least overlapping. The task is to find a code for the set $Z_S \cup Z_A \cup Z_C$ such

that after logic minimization the overall costs for the three blocks S, A and C are minimal.
Focusing the PLA based controller structures the two level logic optimization task can be reduced to satisfy a set of encoding constraints EC (M) for a state transition table STT of a FSM M [DeMi85]. The computed partition of M into S, A and C allows to group the set of encoding constraints in the form:

$$EC(M) = EC(S) \cup EC(A) \cup EC(C) \cup EC(S, A) \cup EC(S, C) \cup EC(A, C)$$

and to reduce the set by deleting the interacting constraints. We get the remaining set of constraints:

$$EC(S) \cup EC(A) \cup EC(C) \quad .$$

To solve this reduced encoding problem we insert the temporary splitting tupels 100, 010 and 001 into the input part of block S, A and C, respectively. We now obtain a new simultaneous state encoding model illustrated for our example in Figure 4a in KISS notation [SSLM92]. Because the inserted splitting tupels are one hot encoded they exclude the encoding constraints $EC(S, A) \cup EC(S, C) \cup EC(A, C)$ exceeding the logic block boundaries. Thus the simultaneous state encoding model can be handled by common state assignment tools like NOVA [ViSa89]. Involving all encoding constraints inside S, A, C in the state assignment process leads to a minimized total combinational cost for all blocks. For our example, the results of encoding are depicted in Figure 4b.

```
# encoding task:
 .i 5
 .o 4
 .s 4
# autonomous block A
 --100 3 0 ----
 --100 0 0 ----
 --100 2 3 ----
 --100 1 2 ----
# sequencing block S
 00010 0 2 ---1
 1-010 0 3 ---1
 00010 2 2 ---1
 10010 2 1 ---1
 10010 3 1 ---1
# command block C
 00001 0 ~ 10--
 01001 0 ~ 10--
 1-001 0 ~ 10--
 00001 2 ~ -01-
 10001 2 ~ -01-
 -1001 2 ~ -01-
 00001 3 ~ 0-1-
 10001 3 ~ 0-1-
 -1001 3 ~ 0-1-
 --001 1 ~ 110-
# state assignment:
 .code 3 11
 .code 0 00
 .code 2 10
 .code 1 01
```

a) State encoding model with additional inserted splitting tupel 100, 010 and 001

```
# autonomous block A'
 .i 2
 .o 2
 11 00
 00 00
 10 11
 01 10
# sequencing block S'
 .i 4
 .o 3
 0000 101
 1-00 111
 0010 101
 1010 011
 1011 011
 0011 --0
 0100 --0
 -11- --0
 --01 --0
# command block C'
 .i 4
 .o 3
 0000 10-
 0100 10-
 1-00 10-
 0010 -01
 1010 -01
 -110 -01
 0011 0-1
 1011 0-1
 -111 0-1
 --01 110
```

b) Truth tables (.type fr) of encoded blocks (deleted splitting tupel)

```
# autonomous block A"
 .i 2
 .o 2
 .s 2
 01 10
 10 11
# sequencing block S"
 .i 4
 .o 3
 .p 3
 00-0 101
 101- 011
 1-00 111
# command block C"
 .i 4
 .o 3
 .p 3
 --01 110
 --1- 001
 ---0 100
```

c) PLA description (.type f) of optimized blocks

Figure 4. Example of simultaneous state encoding and updating logic optimization task

4.2 Updating Logic Optimization Task

After encoding we have to update the optimization task by the steps:

1. Substitute symbolic states in the blocks S, A and C by computed binary codes.
2. Remove all splitting tupels in S‘, A‘ and C‘.
3. Include OFF set in S‘ defined by load = 0,
   i.e. block A operates autonomously (Figure 3c block S).

Figure 4a shows the simultaneous encoding model including inserted splitting tupels and the resulting overall state assignment using NOVA [ViSa89]. The tasks for the subsequent logic optimization defined by truth tables S‘, A‘ and C‘ are illustrated in Figure 4b. PLA descriptions of the minimized blocks obtained by means of ESPRESSO [BMHS84] for overall two level minimization are depicted in Figure 4c.

For our example, Table 1 summarizes characteristics of the FSM partitioning compared to conventional single FSM and vertical partitioning using the PLA cost function $area_{PLA} = (2*(\#I) + (\#O))*(\#PT)$ with number of inputs #I, outputs #O and product terms #PT, respectively.

FSM architecture	**single FSM**	**vertical partitioning [Paul89]**	**horizontal partitioning using autonomous block**
structure	conventional single PLA	sequencing + command PLA	sequ. + autonom. + com. PLA
$area_{PLA}$	91	50 + 21 = 71	12 + 21 + 33 = 66

TABLE 1. Characteristics of FSM partitioning for our small example

5 Experiments and Application Results

This partitioning and optimization approach is implemented in the program **CNT**. It is embedded in a experimental environment using logic synthesis tools NOVA, ESPRESSO and SYNOPSYS as shown in Figure 5.

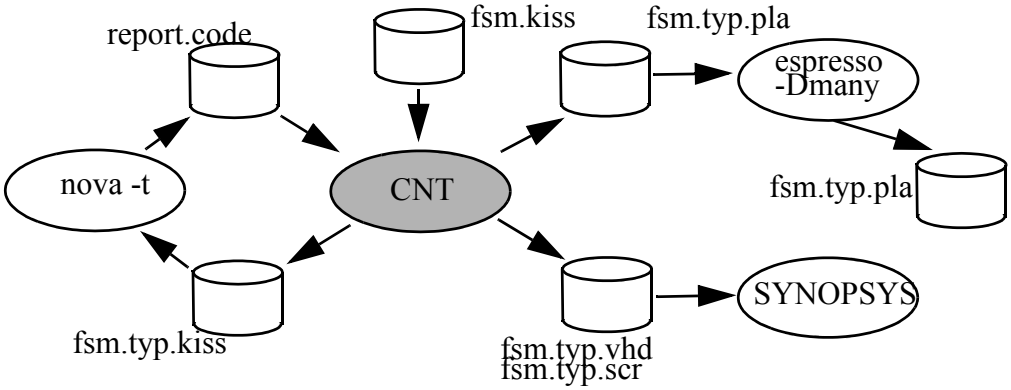
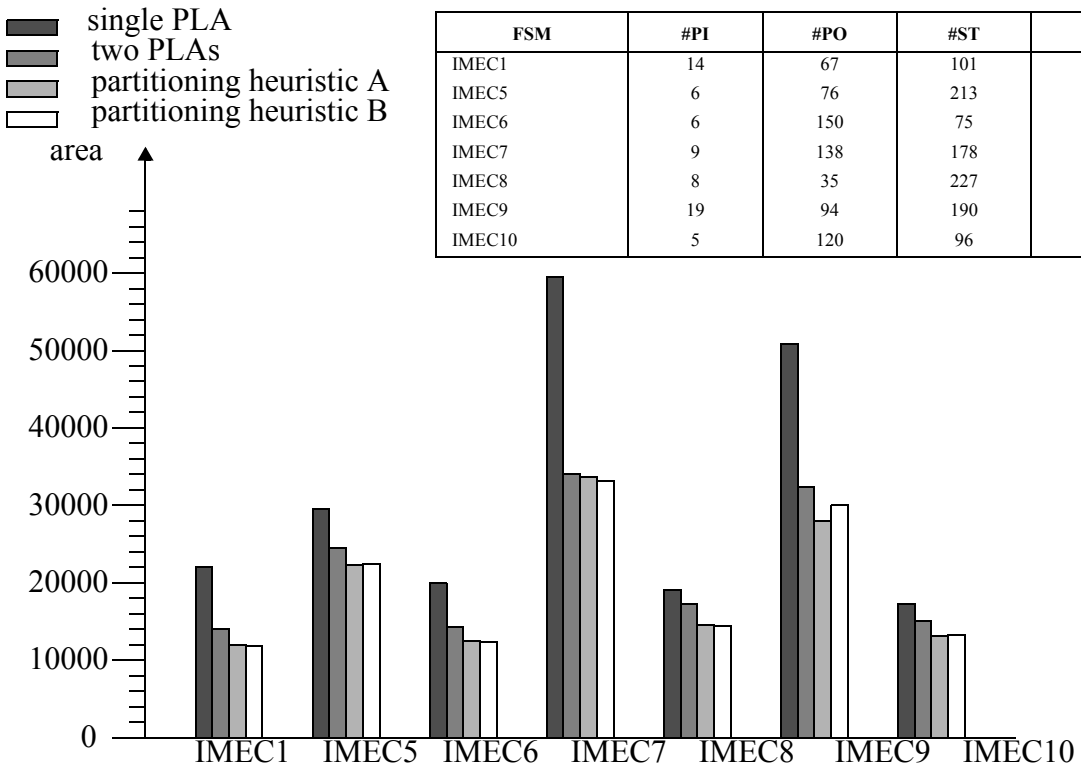


Figure 5. Interaction of implemented tool CNT to logic synthesis tools NOVA, ESPRESSO and SYNOPSYS

Table 2 presents experimental results obtained by CNT for MCNC FSM Benchmarks [Yang91]. In relation to $area_{PLA}$ of a single PLA realization design improvements are compared using heuristics A and B for the constraints driven partitioning of state transition graph. Significant area savings are achieved especially for Moore FSMs (marked by *), FSMs with mainly Moore behaviour (marked by #) and larger FSM by exploiting the heuristic B

FSM	#PI	#PO	#ST	#PT	single FSM	partitioning heuristic A	partitioning heuristic B
tav	4	4	4	49	198	**152**	**152**
bbara	4	2	10	60	550	**504**	606
ex4 #)	6	9	14	21	589	**418**	**418**
mark1 *)	5	16	15	22	684	**390**	402
tss #)	9	12	14	26	920	745	**738**
tma *)	7	6	20	44	986	819	**782**
bbsse	7	7	16	56	**990**	1096	1097
cse	7	7	16	91	1617	1854	**1417**
keyb #)	7	2	19	170	1705	2274	**1541**
pma *)	8	8	24	73	1925	1355	**1077**
styr	9	10	30	166	4257	**3412**	4150
tbk	6	3	32	1569	5310	4926	**4499**
s298 *)	3	6	218	1096	23976	22524	**19616**

TABLE 2. Results obtained for MCNC FSM Benchmarks using heuristic A or B



FSM	#PI	#PO	#ST	#PT
IMEC1	14	67	101	221
IMEC5	6	76	213	353
IMEC6	6	150	75	152
IMEC7	9	138	178	409
IMEC8	8	35	227	366
IMEC9	19	94	190	134
IMEC10	5	120	96	130

Figure 6. Experimental results using large Moore sequencers

Industrially used Moore sequencers are characterized by the property that in many STGs a large part of edges are unconditional transitions called chains and DC-trees [KoWa93]. CNT exploits this additional optimization potential for the partitioning and overall optimization of separated logic blocks. Figure 6 shows that methods implemented in the program CNT are especially suitable for large Moore sequenzers like application specific sequencers from IMEC (also used in [GeSa92]) whose graph representation contains a relatively small number of forks.

6 Conclusions and Future Works

We have presented an approach for partitioning and state encoding of FSM based on extracting an autonomous FSM and creating a simultaneous state encoding model. The implemented procedure CNT leads especially for large Moore sequencers to significant improvements of the synthesized designs.
The procedure CNT should be closer adapted to qualitative and quantitative characteristics of the given STG (like number and topology of state chains and joins). We currently extent the experimental environment including synthesis tools SIS and SYNOPSYS and round off our approach by taking into account random logic blocks and adapted heuristics. The proposed partitioning approach is supposed to be a helpful starting step for multi-level synthesis using the decreased complexity of relatively independent subproblems. Another extension aims at programmable logic. In this case the partitioned blocks have to fulfill strong limitations concerning the number of inputs, outputs and products.

Bibliography

[AmBa87] Amann, R.; Baitinger, U.G.:New State Assignment Algorithms for Finite State Machines Using Counters and Multiple-PLA/ROM Structures. ICCAD'87, pp. 20-23.

[AmBa89] Amann, R.; Baitinger, U.G.: Optimal State Chains and State Codes in Finite State Machines. IEEE Transactions on Computer-Aided Design, Vol. 8, No. 2, Febr. 1989, pp.153-170.

[AsDN92] Ashar, P.; Devadas, S.; Newton, A.R.: Sequential Logic Synthesis. Kluwer Academic Publishers, Boston/ Dortrecht/London, 1992.

[BaBr93] Baranov, S.; Bregman, L.: Automata Decomposition and Synthesis with PLAM. Microprocessing and Microprogramming, North Holland, vol.38 (1993), pp.759-766.

[BHMS84] Brayton,R.; Hachtel, G.; McMullen, C.; Sangiovanni-Vincentelli, A.: Logic Minimization Algorithms for VLSI Synthesis, Kluwer Academic Publishers, Boston, 1984.

[DeMi85] De Micheli, G.; Brayton, A.R.; Sangiovanni-Vincentelli, A.: Optimal State Assignment for Finite State Machines. IEEE Trans. on CAD, Vol. 4, No. 4, 1985, pp. 269-284.

[DeMi94] De Micheli, G.: Synthesis and Optimization of Digital Circuits. Mc Graw-Hill, Inc., New York, 1994.

[Deva89] Devadas, S.: General Decomposition of sequential machines: Relationships to state assignment. Design Automation Conference (DAC´89), June 1989, pp. 13-27.

[FeMu93] Feske, K.; Mulka, S.: FSM-Partitionierung und -Synthese auf der Grundlage von PLAs und Autonomen Automaten. Tech.Report SFB - 358 - B1 - 5/93, FhG, IIS/EAS Dresden, November 1993.

[GeSa92] Gerbaux, L.; Saucier, G.: Automatic synthesis of large Moore sequencers. The European Conference on Design Automation (EDAC‘92), 1992 March 1992, pp. 237-244.

[HaSt66] Hartmanis, J.; Stearns, R.E.: Algebraic Structure Theory of Sequential Machines. Prentice-Hall, Inc., Englewood Cliffs, N.J., 1966

[JoKo91] Jozwiak, L.; Kolsteren, J.C.: An Efficient Method for the Sequential General Decomposition of Sequential Machines. Microprocessing and Microprogramming, North Holland, vol. 32 (1991), pp. 657-664).

[KNRR88] Kraemer, H.; Neher, M.; Rietsche,G.; Rosenstiel, W.: Data Path and Control Synthesis in the CADDY Synthesis System. Intern. Workshop on Logic and Architectural Synthesis for Silicon Compilation, Grenoble, 1988.

[KGFF94] Koegst, M.; Grass, W.; Franke, G.; Feske, K.: Simultaneous State Encoding and Communication Cost Optimization for FSM Net Design. Proc. of the Twentieth EUROMICRO conference, IEEE, IEE, Liverpool, UK, September 1994.

[KoWa93] Kottsieper, J.; Waldschmidt, K.: Application of the novel associative programmable arry-structure Multi-Match-PLA in synthesis of decomposed finite state machines. Microprocessing and Microprogramming, vol.38, Sept.1993, pp. 455-465, pub. for EUROMICRO'93

Paul89] Paulin, G.P.: Horizontal Partitioning of PLA-based Finite State Machines. 26th ACM/IEEE Design Automation Conference, 1989, pp. 333-338.

[Putk91] Puttkammer, A.: Entwicklung, Implementierung und Bewertung von Methoden zur Modularisierung von Steuerwerken. Diplomarbeit, Universität Passau, Fachbereich Mathematik und Informatik, 1991.

[Sauc93] Saucier, G.: Synthesis of a Finite State Machine on any target. EDAC'93, Paris.

[SSLM92] Sentovich, E. M.; Singh, K. J.; Lavagno, L. et al.,: SIS: A System for Sequential Circuit Synthesis", Memorandum No. UCB/ERL M92/41, Electronics Research Laboratory, Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA, May 1992.

[ViSa89] Villa, T.; Sangiovanni-Vincentelli, A.: NOVA: State Assignment of Finite State Machines for Optimal Two-Level Logic Implementations. 26th ACM/IEEE Design Automation Conference, 1989.

[Yang91] Yang, S.: Logic Synthesis and Optimization Benchmarks, User Guide Version 3.0.

[YaOI93] Yang, W.-L.; Owens R. M.; Irwin, M. J.: Multi Way Decomposition based on Interconnect Complexity. European Design Automation Conference (EURO-DAC '93), 1993, pp. 390-395.

[ZSRM90] Zegers, J.; Six, P.; Rabaey, J.; De Man, H.: CGE: Automatic Generation of Controllers in the CATHEDRAL-II Silicon Compiler. EDAC‘90, Glasgow, 1990, pp.617-621.