Fast Controllers for Data Dominated Applications

Andre Hertwig, Hans-Joachim Wunderlich
University of Stuttgart, Germany
Computer Architecture Lab

Abstract

A target structure for implementing fast edge-triggered control units is presented. In many cases, the proposed controller is faster than a one-hot encoded structure as its correct timing does not require master-slave flip-flops even in the presence of unpredictable clocking skews.

A synthesis procedure is proposed which leads to a performance improvement of 40% on average for the standard benchmark set whereas the additional area is less than 25% compared with conventional finite state machine (FSM) synthesis. The proposed approach is compatible with the state-of-the-art methods for FSM decomposition, state encoding and logic synthesis.

Keywords: *FSM synthesis, performance driven synthesis, synthesis of testable controllers*

1. Introduction

In recent years, the performance of datapath/controller-systems has been increased distinctively mainly by optimizations of the datapath. Powerful techniques like pipelining and massive parallelism as well as innovative computing algorithms improve the maximum speed of the datapath but require highly complex control units at the same time. Hence, the performance of the entire system is more and more determined by the speed of the control unit.

The state diagram for such a datapath/controller-system is not as densely meshed as for typical control dominated designs. In particular, the controller does not have a large number of wait states which corresponds to self-loops of the state transition diagram. Instead of that a number of larger cycles is implemented which corresponds to the instruction cycles of the entire system.

It is well known that the fastest implementation style of finite state machines is based on one-hot-encoding [16], as the state transition function is reduced to a switching matrix. But one-hot-encoding requires one flipflop for each state, and this already leads to a tremendous amount of hardware for medium sized control units. For this reason it has to be combined with FSM decomposition techniques, so that the entire controller is implemented by a distributed network of communicating FSMs [1, 5, 19, 26]. But it cannot be guaranteed that the submachines are always smaller than the original one, and in this case the FSM state transition diagram has to be changed in order to get an implementation which works with sufficiently high frequency.

This work was supported by the DFG under grant Wu 245/1-1

But even if FSM decomposition leads to sufficiently small submachines, the performance is affected by the need of using master-slave flipflops in order to ensure a correct timing of an edge-triggered design. This is illustrated by the small example of figure 1. If the clock delay at flipflop FF_2 is significantly smaller than the delay at FF_1 then FF_1 computes its next state value based on the next state value of FF_2 erroneously.

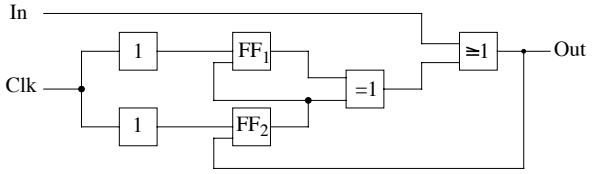

Figure 1: Incorrect function due to clocking skew

This problem is independent of the operating frequency, and can only be solved by using two-edge-triggered or master-slave flipflops. Figure 2 shows the timing diagram for this design style.

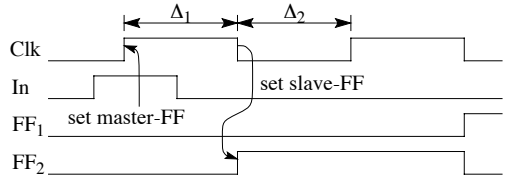

Figure 2: Timing diagram using master slave flipflops.

Each flipflop stores its new state at the rising edge but changes its output only during the falling edge. Only part Δ_2 of the entire clock phase $\Delta_1+\Delta_2$ is available for the propagation delay of both the combinational network and the set time of the flipflops, and the frequency for this design style is lower than for single-edge triggered flipflops. But in many cases master-slave flipflops are required as we may expect variations of the speed of the same gates up to the factor of 4 for a typical 0.7 µm library. Logic synthesis tools which take timing variations into account provide solutions for the worst case which are not optimal in general.

In this paper, we present a controller structure which

does not require master-slave flipflops for safe functioning, which is significantly faster than solutions obtained by standard FSM synthesis but which is only moderately larger.

The following part of the paper is organized as follows: In the next section some related work is discussed, and in section 3, the target structure is presented in detail. In section 4 it is shown that synthesis of these structures can be reduced to a version the graph coloring problem, and experimental results are discussed in section 5.

2. State of the art

Figure 3 shows the standard implementation of a control unit as a result of state encoding and logic synthesis by such classical tools as NOVA, MUSTANG, ESPRESSO, MIS e.g. [25,2]. A combinational block implements the state transition function and the output function, and the edge-triggered register represents the state.

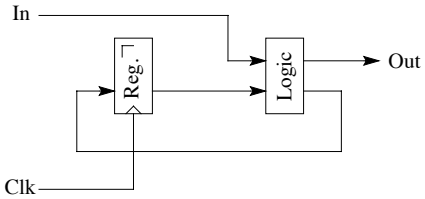

Figure 3: Standard structure of a FSM

An important advantage of this structure is the simple clocking scheme with a single clock which is not gated so that correct function and timing can easily be verified. The maximum speed of this structure is determined by Δ_1 which is the time required for changing the outputs of the master slave-flipflop, and by Δ_2 which is the maximum propagation delay through the combinational block. As this delay directly corresponds to the size of the combinational block, automata decomposition is mainly used for performance improvements. A first decomposition approach was proposed by Hartmanis and Stearns [12] who applied an algebra on partitions of states. More efficient methods are based on general factorization [1,5,19]. The result is a network of communicating FSMs which is subject of further optimizations [21,7,24]. The structure of each of the FSMs may be synthesized by sophisticated encoding and logic synthesis algorithms [9]. For fast controllers performance driven logic synthesis should be applied [17,5,23,4], and the resulting structure may be optimized by retiming algorithms [20,8,10]. The performance should also be respected during technology mapping and layout generation.

All these performance driven synthesis algorithms contain essentially two phases: first, the FSM is mapped to the target structure of figure 3, and next, performance driven encoding and logic synthesis are carried out. In [14,15] the target structure was already modified for performance, and a pipeline controller structure was proposed as shown in figure 4. Here, the set of states is the product of two sets each of them corresponds to one register. As both registers may be controlled by different edges of the same clock signal, master-slave flipflops are not required.

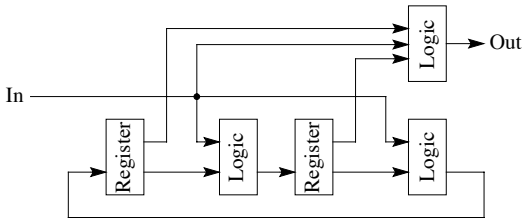

Figure 4: Pipeline controller by [14].

An important advantage of this structure is the existence of a very efficient built-in self-test strategy. In order to make the standard structure of figure 3 self-testable additional test registers are required either for pattern generation or response evaluation [11]. For the pipeline controller no additional test registers are required in order to implement BIST, as both of the system registers may be implemented as BILBO-like test registers [12] and perform pattern generation and response evaluation alternatingly.

As a drawback of this structure the combinational logic may become in the worst case twice as large as a standard realization. The target structure presented in the next section avoids this drawback but preserves all the advantages concerning speed and testability.

3. Target structure

The general form of the pipeline controller of figure 4 uses both registers concurrently for encoding a single state, and it is based on a sophisticated lattice algorithm. The algorithm can be simplified to a graph coloring algorithm if each state is represented by just one of the registers. In this case, the state transition diagram has to be marked by two colors, each of them representing one register. It is not always possible to color a graph by just two colors, for instance if there is a cycle with an odd number of nodes this coloring is impossible, and equivalent states have to be added to the state transition diagram. If the states are mapped to three registers instead of two a 3-color problem has to be solved. Any cycle with more than one node is colorable by 3 colors, and in general only few additional nodes have to be added. Additional nodes are always required for self-loops, for this reason the approach works best for data dominated applications with a small number of wait-states. The structure using three registers is shown in figure 5 as one of the registers is bypassed by a direct line, we call this structure bypass-pipeline.

Each time, only one of the three registers determines the state of the entire controller. This has to be ensured by an appropriate encoding. The best way is to reserve one bit t_i of each register for this purpose, so that the remaining bits can be used for an arbitrary strategy of state encoding. Any

of the known performance driven encoding and synthesis techniques may be applied to this structure, and as each of the logic blocks is expected to be smaller than the standard realization we will have performance improvements due to the shorter paths. As a result we get a token bit architecture as shown in figure 6.

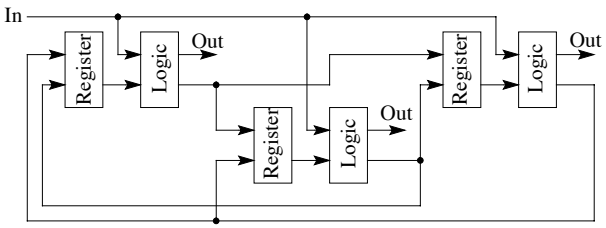

Figure 5: Bypass-pipeline

The register R_i determines the state of the controller if t_i=1. The two output function τ_i passes the token to one of the registers R_j, $i \neq j$, in the next clock. Hence always exactly one of the three bits t_0, t_1, and t_2 is set.

The timing of this controller is safe if a 3-phase clock is used. In the sequel we describe how a safe single clock scheme can be implemented without general master-slave flipflops. The basic idea is to implement only the three token flipflops in a master-slave style so that a robust token passing is guaranteed. This will not slow down performance as the token signals are never part of a critical path. They

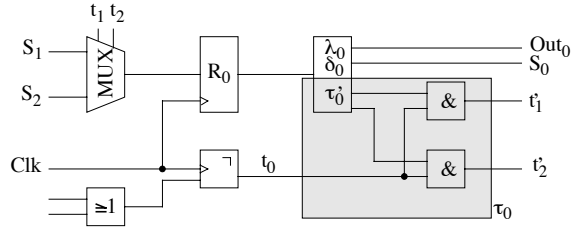

Figure 7: Implementation of robust token passing

control the multiplexers which are at the end of the paths, hence they are not needed before all the state transitions are computed. Also token passing itself is not critical as τ has to evaluate first the state of R_i in order to decide whether the token must be passed. This leads to the actual implementation as shown in figure 7 for stage 0. The token signal is further used for making the state transitions robust, too. State transitions are made robust easily by exploiting the fact that the content of a register R_i is completely irrelevant if t_i=0. Only in cases when t_i=1 the register must stay stable. But if t_i=1 then we have t_i=0 in the next cycle, and the register is not needed. Hence, we are allowed to reload the state of R_i if t_i=1, and all the relevant information of the controller is conserved sufficiently long (cf. figure 8).

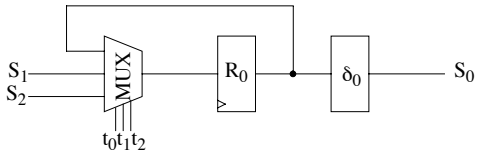

Figure 8: Robust state transitions

All this additional circuitry is significantly less than the overhead for master-slave flipflops. Moreover, these gates only ensure that critical signals are stable for a sufficiently long time but they do not prolong critical paths and they do not slow down performance.4. Synthesis by graph coloring

It is not possible to color the state transition graph using standard algorithms as the graph has also to be modified concurrently so that it is 3-colorable with a minimum number of additional nodes. In the sequel we use a heuristic approach which is also used for register allocation during compilation [3]. In a preprocessing step all self-loops are cut by introducing equivalent states (fig.9).

Figure 9: Cutting self-loops

Now as many nodes as possible are colored by using just three colors. Since in general the state transition graph will

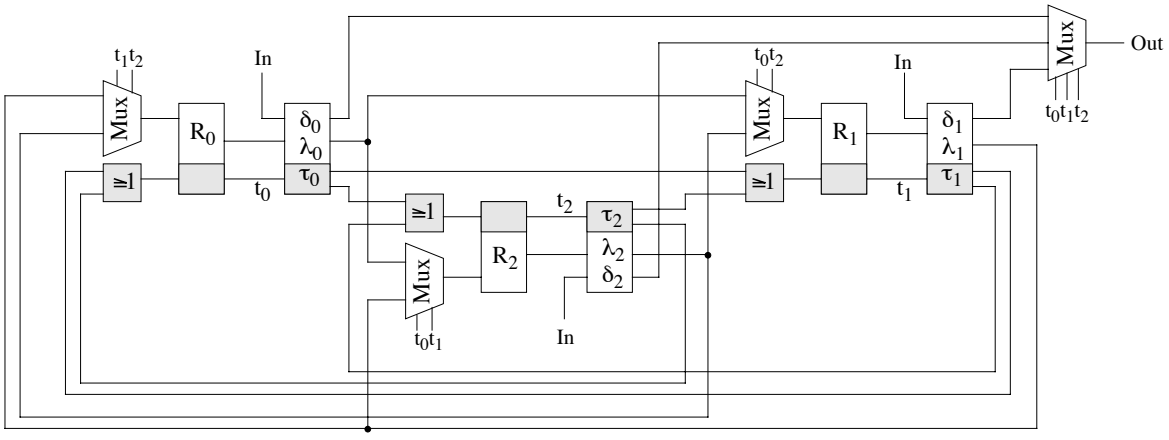

Figure 6: Token bit architecture

not be 3-colorable, a few nodes which are not colored are left. The number of these nodes should be minimal, and the experimental results of section 6 show that a standard algorithm as proposed in [3] works sufficiently well.

In the last step, the uncolored nodes have to be substituted by introducing equivalent states and graph transformations. Here we have to distinguish three cases:

Case 1: Node $v \in V$ is uncolored, and all its colored successors $w \in sd(v)$ have the same color $c_a \in C = \{c_a, c_b, c_c\}$ (cf. figure 10). In this case, an additional node v' is introduced and colored by $c(v') = c_b$, and v is colored by $c(v) = c_c$. Edges are introduced from v' to all successors $w \in sd(v)$, and all edges (x, v) are redirected to (x, v') if $c(x) = c_c$.

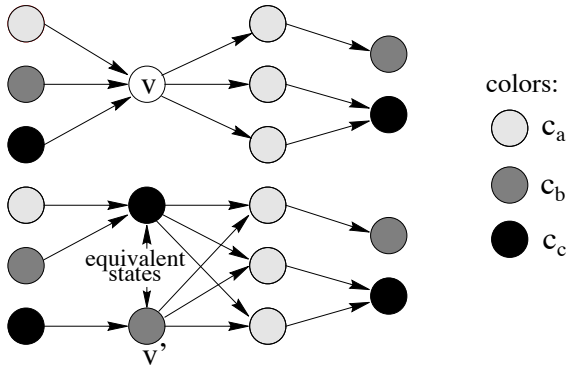

Figure 10: Equivalent graph transformation in case 1

Case 2: Node $v \in V$ is uncolored, and there is one color $c_a \in C$ not assigned to any direct predecessor $u \in pd(v)$. Then there are successors $w \in sd(v)$ with color $c(w) = c_a$, for each of these nodes an equivalent state w' is going to be introduced. For each edge (w, x) a new edge (w', x) is added, and (v, w) is redirected to (v, w'). Node w' get the color $c_b \in C$ or $c_c \in C$ if possible, otherwise w' is not colored

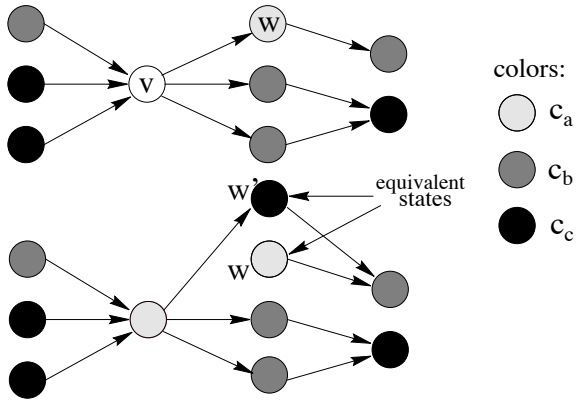

Figure 11: Equivalent graph transformation in case 2

Case 3: All the other uncolored nodes are treated in two steps: First, state v is splitted into two equivalent states v and v' so that no predecessor $w \in pd(v)$ has color $c_a \in C$ and no predecessor $w' \in pd(v')$ has color $c_b \in C$. As this new situation corresponds to case 2 it is dealt with as already described.

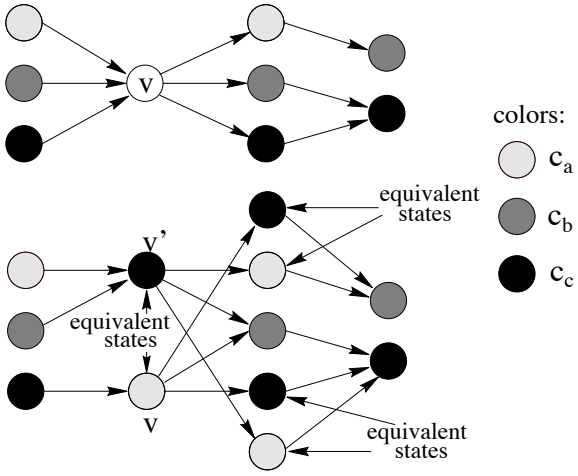

Figure 12: Equivalent graph transformation in case 3

It should be noted that in all three cases a new state v' is only created if there not already exists an equivalent state. If there already exists an equivalent node with the required color the edges are just redirected. This ensures the termination of the algorithm after a few iterations, at least when each state has two equivalent nodes with different colors as can be proved easily.

The target structure facilitates state encoding and logic synthesis significantly as each register and combinational block is dealt with independently. The well known encoding techniques are applied directly with small modifications which ensure that states of a certain color are assigned only to one register. Results reported in the next section are based on the encoding strategies of NOVA [25]. After encoding, the three truth tables of the functions $(\lambda_i, \delta_i, \tau)$ are established, and a logic synthesis method is applied. The results presented here are obtained by the standard script of SIS.

5. Experimental results

All the algorithms presented so far are implemented in C++, and in this section we report results for the benchmark set distributed for the Workshop on Logic Synthesis 1993 [22], for the control unit of the DLX processor [13], and results for an experimental controller accu16. In contrast to all the other performance-driven synthesis methods, the algorithm has its best results for the largest circuits which cannot be dealt with by the known methods. For small controllers the constant hardware overhead for the token control logic

is dominating, and methods presented in [14, 15] may perform better. As 3 flipflops are already needed for the token control we only investigated FSMs which have at least 33 states and require more than 5 flipflops.

Both bypass-pipelines and standard controllers are synthesized by the same encoding and logic synthesis algorithms based on SIS and NOVA, and the results are compared.

First, the bypass-pipeline deals with three blocks independently, each of them is smaller than the standard structure, and the overall computing time for synthesizing a bypass-pipeline is significantly smaller than the time for the standard approach. Table 1 shows the computing time on a SUN Sparc5 workstation required for state encoding.

Circuit	Bypass-pipeline (h:min:s)	Standard structure (h:min:s)
dlx	0:0:7.9	0:0:22.8
planet	0:0:6.7	0:0:20.6
planet1	0:0:6.7	0:0:20.6
s1488	0:0:12.0	0:0:26.8
s1494	0:0:12.2	0:0:26.8
scf	0:0:56.9	0:7:22.9
s298	0:10:52.0	1:19:21.8
accu16	0:0:2.6	0:0:2.9
s510	0:0:8.3	0:0:12.2

Table 1: Run time for state encoding on a SUN Sparc5

Table 2 illustrates the effect of the graph coloring algorithm. |S| denotes the number of states in the original FSM, $|S_{l\text{-free}}|$ is the number of states after preprocessing where all states with selfloops are doubled. $|S_{total}|$ is the total number of states after 3-coloring, and the remaining 3 columns describe how these states are distributed among the registers.

circuit	|S|	$\|S_{l\text{-free}}\|$	$\|S_{total}\|$	$\|S_1\|$	$\|S_2\|$	$\|S_3\|$	comp. time
dlx	49	52	52	23	15	14	0.1s
planet	48	49	49	15	18	16	<0.1s
planet1	48	49	49	15	18	16	<0.1s
s1488	48	49	49	14	17	18	0.2s
s1494	48	49	49	18	17	14	0.2s
scf	121	122	122	36	48	38	0.2s
s298	218	219	219	64	83	72	1.2s
accu16	28	36	40	11	15	14	0.1s
s510	47	70	70	24	24	23	0.1s

Table 2: Results of the coloring algorithm

It is interesting that most of the graphs are already colorable by three colors after self-loop cutting. Only the experimental controller accu16 requires an introduction of additional equivalent states. It should be noted that the FSM s510 has a rather large number of wait states, and the total number of states has already increased distinctly after preprocessing. Obviously it is not part of a data dominated application, and the algorithm is not expected to provide best results here.

In the average the increment of the number of states is small, and we expect that the bypass-pipeline is only moderately larger than the standard structure. This is confirmed by table 3 which shows the transistor count of the different blocks. The last column gives the quotient of the transistors required for the bypass-pipeline and for the standard structure. In general, the bypass-pipeline leads to 25% larger circuits, only s510 requires 100%.

Circuit	Bypass-pipeline (number of transistors)						Std. structure (#transistors)			A
	log.1	log.2	log.3	mux	reg.	total	logic	reg.	total	
dlx	1634	1206	1502	284	336	4962	4076	204	4280	1.16
planet	1150	1764	1158	384	336	4792	3544	204	3748	1.28
planet1	1150	1764	1158	384	336	4792	3544	204	3748	1.28
s1488	1980	2016	1486	396	354	6232	4990	204	5194	1.20
s1494	2392	1642	1302	396	354	6086	5078	204	5274	1.15
scf	3238	2702	2054	888	426	9308	6512	238	6750	1.38
s298	4316	7692	6174	312	462	18956	16360	272	16632	1.14
accu16	576	624	516	248	318	2282	1214	170	1384	1.64
s510	1058	1454	1256	272	372	4412	2002	204	2206	2.00

Table 3: Transistor count

The next table indicates that the small increase in area corresponds to a significant improvement of the performance. Table 4 shows the maximum propagation delay of all the blocks. The speed of the bypass-pipeline is determined by the maximum of these 3 delays plus the delay of the additional circuitry. It is shown that the total run time of the bypass-pipeline is significantly lower than the time of the standard solution. Mainly we have performance improvements of 40%.

Circuit	Bypass-pipeline					Standard structure		
	logic 1	logic 2	logic 3	reg+ mux	total	logic	reg.	total
dlx	23.40	19.62	20.91	5.52	28.92	46.76	6.30	53.06
planet	20.08	26.09	20.02	5.52	31.61	49.31	6.30	45.61
planet1	20.08	26.09	20.02	5.52	31.61	49.31	6.30	45.61
s1488	28.05	29.32	27.38	5.52	34.84	61.48	6.30	67.78
s1494	33.80	25.26	27.47	5.52	39.32	61.56	6.30	67.86
scf	40.40	34.97	30.63	5.52	45.92	72.45	6.30	78.75
s298	58.30	84.96	70.44	5.52	90.48	154.31	6.30	160.61
accu16	11.37	11.99	11.38	5.52	17.51	15.61	6.30	21.91
s510	21.74	31.07	25.36	5.52	36.59	33.93	6.30	40.23

Table 4: Propagation delays

Finally, table 5 compares the improvement of the performance and the increase of the area for all the benchmarks investigated. For all the controllers for data dominated applications with at least 33 states we get a significant gain in the performance at very low cost. Only the FSM s510 with

a large number of wait-states is not suitable for the method presented. This was already predicted after the preprocessing step.

Circuit	Delay [%]	Transistors [%]
dlx	- 45.50	+ 15.93
planet	- 30.70	+ 27.85
planet1	- 30.70	+ 27.85
s1488	- 48.60	+ 19.98
s1494	- 42.06	+ 15.40
scf	- 41.69	+ 37.90
s289	- 43.66	+ 13.97
accu16	- 20.08	+ 64.88
s510	- 9.05	+ 100.00

Table 5: Costs and performance for the bypass-pipeline

6. Conclusions

For controllers in data dominated applications a target structure has been presented which allows a significantly higher performance than standard realizations. The higher frequency is possible since the combinational blocks are smaller and the propagation paths are shorter, and also since single-edge triggered flipflops are sufficient for guaranteying correct timing.

A synthesis procedure has been presented, and the results show high improvements of the speed with rather small hardware overhead. The approach is compatible with the known state encoding and logic synthesis techniques, and it may also be used for implementing basic blocks after FSM decomposition.

7. References

[1] P. Ashar, S. Devadas, A. R. Newton: Optimum and Heuristic Algorithms for an Approach to Finite State Machine Decomposition, in: IEEE Trans. on CAD, Vol. 10, No. 3, March 1991, pp. 296-310

[2] R. K. Brayton, R. Rudell, A. L. Sangiovanni-Vincentelli, A. Wang: MIS: A Multi-Level Optimization System; in: IEEE Trans. on CAD , Vol. 6, No. 11, November 1987, pp. 1062-1081

[3] P. Briggs, K.D. Cooper, K. Kennedy, L. Torczon: Coloring Heuristics for Register Allocation; in: Proc. SIGPLAN '89 Conf. on Programming Language Design and Implementation; Portland, Oregon, 1989, pp. 275-284

[4] H.-C. Chen, D. Hung-Chang Du, L.-R. Liu: Critical Path Selection for Performance Optimization; in: IEEE Trans. on CAD, Vol. 12, No. 2, February 1993, pp. 185-195

[5] G. DeMicheli: Synchronous Logic Synthesis:Algorithms for Cycle-Time Minimization; in: IEEE Trans. on CAD, Vol. 10, No. 1, January 1991, pp. 63-73

[6] S. Devadas, A. R. Newton: Decomposition and Factorization of Sequential Finite State Machines; in: IEEE Trans. on CAD, Vol. 8, No. 11, November 1989, pp. 1206-1217

[7] S. Devadas: Optimizing Interacting Finite State Machines Using Sequential Don't Cares; in: IEEE Trans. on CAD, Vol. 10, No. 12, December 1991, pp. 1473-1484

[8] S. Dey, M. Potkonjak, S. G. Rotweiler: Performance Optimization of Sequential Circuits by Eleminating Retiming Bottlenecks; in: Proc. IEEE/ACM Int. Conf. on CAD 1992, Santa Clara, Cal., 1992, pp. 504-510

[9] X. Du, G. Hachtel, B. Lin, A. R. Newton: MUSE: A MUltilevel Symbolic Encoding Algorithm for State Assignment; in: IEEE Trans. on CAD, Vol. 10, No. 1, January 1991, pp. 28-38

[10] G. Even, I. Spillinger, L. Stock: Retiming Revisited and Reversed; in: IEEE Trans. on CAD, Vol. 15, No. 3, March 1996, pp. 348-357

[11] B. Eschermann, H.-J. Wunderlich: Optimized Synthesis Techniques for Testable Sequential Circuits; in: IEEE Trans. on CAD, Vol. 11, No. 3, March 1992, pp. 301-313

[12] J. Hartmanis, R. E. Stearns: Algebraic Structure Theory of Sequential Machines, Prentice Hall, Englewood Cliffs, 1966

[13] J. L. Hennessy, D. A. Patterson: Computer Architecture: A Quantitative Approach; 2nd Ed.; Morgan Kaufmann Publisher, 1996

[14] S. Hellebrand, H.-J. Wunderlich: Synthesis of Self-Testable Controllers; in: Proc. EDAC/ETC/EuroAsic, Paris, 1994, pp. 580-585

[15] S. Hellebrand, H.-J. Wunderlich: An Efficient Procedure for the Synthesis of Fast Self-Testable Controller Structures; in: Proc. ACM/IEEE Int. Conf. on CAD, San Jose, Ca., 1994

[16] S. C.-Y. Huang, W. Wolf: Performance-Driven Synthesis in Controller-Datapath Systems; in: IEEE Trans. on VLSI Systems, Vol. 2, No. 1, March 1994, pp. 68-80

[17] K. Keutzer, A. Malik, A. Saldanha: Is Redundancy Necessary to Reduce Delay?; IEEE Trans. on CAD, Vol. 10, No. 4, April 1991, pp. 427-435

[18] B. Könemann, J. Mucha, G. Zwiehoff: Built-in Logic Block Observation Techniques, Proc. IEEE Int. Test Conf., Cherry Hill, N.J., 1979,pp. 37-41

[19] K. Lam, S. 6das: Performance-Oriented Decomposition of Sequential Circuits; in: Proc. IEEE Int. Symposium on Circuits and Systems, New Orleans, LA, 1990, pp. 2642-2645

[20] C. Leiserson, J. Saxe: Retiming Synchronous Circuitry; in: Algorithmica, Vol. 6, No. 1, 1991, pp. 5-35

[21] S. Malik, E. M. Sentovich, R. K. Brayton, A. Sangiovanni-Vicentelli: Retiming and Resynthesis: Optimizatizing Sequential Networks with Combinational Techniques; in: IEEE Trans. on CAD , Vol. 10, No. 1, January 1991, pp. 74-84

[22] K. McElvain: IWLS'93 Benchmark Set: Version 4.0, distributed as part of the IWLS'93 benchmark distribution

[23] P. C. McGeer, R. K. Brayon, A. L. Sangiovanni-Vincentelli, S. K. Sahni: Performance Enhancement through the Generalized Bypass Transform; Proc. IEEE/ACM Int. Conf. on CAD 1991, Santa Clara, Ca., 1991, pp. 184-187

[24] J.-K. Rho, F. Somenzi: Don't Care Sequences and the Optimizaton of Interacting Finite State Machines; in: IEEE Trans. on CAD, Vol. 13, No. 7, July 1994, pp. 865-874

[25] T. Villa, A. Sangiovanni-Vincentelli: NOVA: State Assignment of Finite State Machines for Optimal Two-Level Logic Implementations; in: IEEE Trans. on CAD, Vol. 9, Sep. 1990, pp. 905-924

[26] W. Wolf: Decomposing Data Machines; in: Proc. European Conference on Design Automation, Amsterdam, Netherlands, 1991, pp. 378-38