A Design Flow for Asynchronous Dynamic Logic and Standard Synthesis Tools

Frank Sill, Frank Grassert, Andreas Wassatsch, Dirk Timmermann

University of Rostock, Dep. ETIT
Institute of Applied Microelectronics and CS
Richard-Wagner-Str. 31; 18119 Rostock, Germany

{frank.sill,frank.grassert,dirk.timmermann}@ uni-rostock.de

ABSTRACT

For high performance designs, dynamic logic styles are in the focus due to the promising high reachable frequencies. True Single Phase Clock (TSPC) logic yields easy to design circuits with standard cells and high speed potential. The disadvantages are a difficult clock tree design and high power consumption. Asynchronous logic has the potential to solve these problems. Asynchronous Chain (AC)-TSPC logic assembles small asynchronous chains of dynamic logic gates into one period of the global clock. The results are a shorter latency for calculations, power reduction due to smaller clock load and due to no need for latches, and a simpler clock distribution network due to increased clock skew tolerance and due to the reduced clock load.
Current high level synthesis tools do not support automatic synthesis and verification of asynchronous dynamic logic. This paper presents a complete design flow of Asynchronous Chain (AC)-TSPC logic. We use the toolset DYNAMIC, which realizes a transformation of a combinational circuit into a pipelined structure, and the tool AC-DYNAMIC, which implements a splitting of a pipelined structure into an asynchronous clocked structure. Further, AC-DYNAMIC verifies the timing behavior, and realizes optimizations. The design flow is exemplarily utilized for a 32-bit-single-error-correcting circuit.

Table of Contents

1.0 Introduction

This paper presents the application and simulation of the logic style Asynchronous Chain - TSPC (AC-TSPC), presented in [4]. This logic style promises high speed due to the latch-free structure and reduced effort in the clock tree due to the self-timed parts and the reduced sensibility to clock skew. The main goal of this work is the description of a design flow. Following points are presented:

- Tool for an automatic synthesis
- Automatic extensive timing calculation / simulation
- Detailed results of timing behavior of AC-TSPC designs
- Comparisons with other logic styles

Section 2 describes the basics of the AC-TSPC self-timed structure and the important completion detection with its problems in larger designs. The function of the developed tool, which includes high level timing calculation based on the timing values of basic gates, are discussed in Section 3. Section 4 shows results from transistor level simulations and Section 5 concludes this work. Appendix 8 describes how to download and use the tool.

2.0 Self-timed structures

2.1 Dynamic logic for differential usage

The functionality of dynamic logic compared to static CMOS (SCMOS) is depicted in figure 1. The logic function of a dynamic gate is realized only with a network of N-MOS or P-MOS transistors while the logic function in SCMOS is realized with complementary N-MOS and P-MOS transistors. Consequently, dynamic logic is faster, because of the lower input load. Dynamic logic works in two phases, which are dictated by a clock signal. The following explanation is valid for an N-logic block. During precharge (clock low), the P-transistor connects

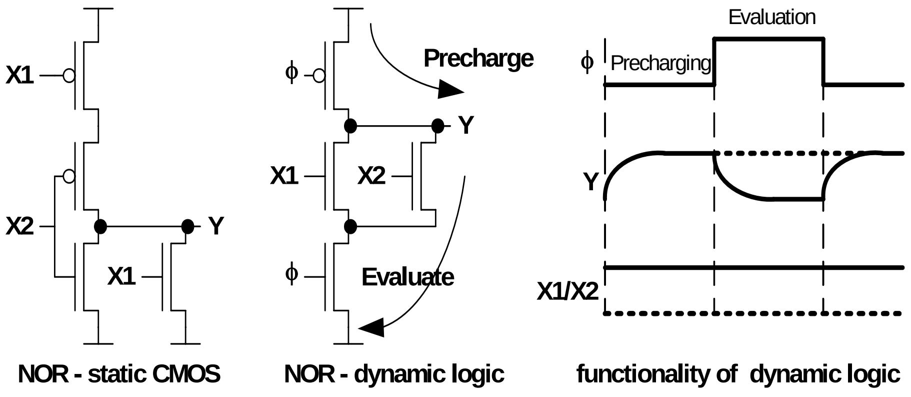

Figure 1: NOR-gate realized in static CMOS and dynamic logic; functionality of dynamic logic

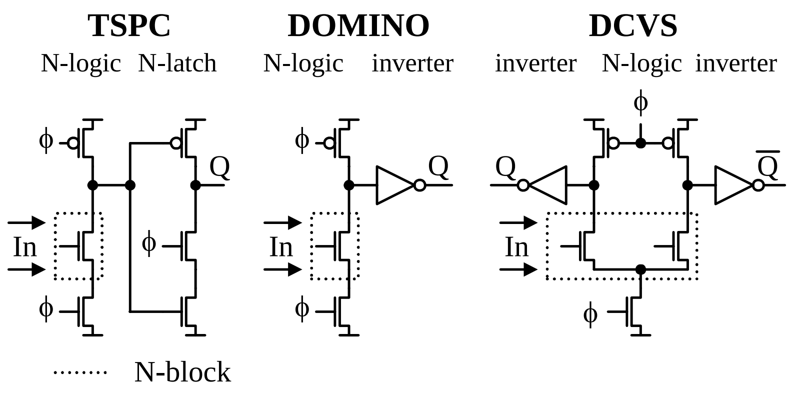

Figure 2: TSPC, DOMINO and DCVS logic

the output-node of the dynamic gate to VDD thus charging the output capacitance to high. In the evaluation phase (clock high) the clocked N-transistor is turned on. Depending upon the input values of the logic tree the output node can be discharged to ground. Therefore, the evaluation phase realizes the logic function. A P-logic block works in a complementary manner with clock signal being high during precharge and vice versa.
TSPC logic (figure 2) uses alternating dynamic N-logic and P-logic blocks connected to N- or P-latches, respectively. In one clock cycle both N- and P-blocks evaluate and precharge. The speed is determined by the slowest action, i.e. evaluation of logic-block and latch (P or N) or precharge of the internal nodes (to high or low).
DOMINO logic uses N-logic blocks only with subsequent static inverting logic, e.g., an inverter. If two sequential gates (dynamic and static part) use the same clock the evaluation proceeds to the first gate and then immediately to the second one. During precharge, each output of the inverting static logic goes low. Therefore, subsequent logic trees can not connect to ground. In two cascaded gates with different clocks the second gate accepts the output signal of the first gate if both evaluation phases overlap. The evaluation phase of the first gate has to hold until the internal node of the second gate is fully settled. Otherwise, information is lost. The precharge of the previous gate does not effects the settled outputs of the next one because of high to low transition only at the inputs.
DCVSL, as an example of a differential logic structure, always evaluates two complementary output values and operates like DOMINO. At the end of evaluation the difference at the outputs can be used for completion detection. However, if two independent domino stages are used instead the same functionality can be realized.

2.2 Basic Self-timed Scheme

Dynamic logic with a complementary structure can be arranged in an asynchronous way. There are two main ways to build up a self-timed scheme for dynamic logic: the gate outputs control the clocking of the previous or the following gate [1]. The main advantage of the first structure is a simple implementation with minimum evaluation time. If the evaluated outputs of a gate have settled, a completion signal is generated and this sets the previous gate in the precharge phase (inputs are processed – start precharge). Precharged outputs set the previous gate in the evaluation phase (outputs are precharged – start next evaluation, inputs can be processed). Because the evaluation of the outputs starts only with valid inputs, every gate is waiting for valid

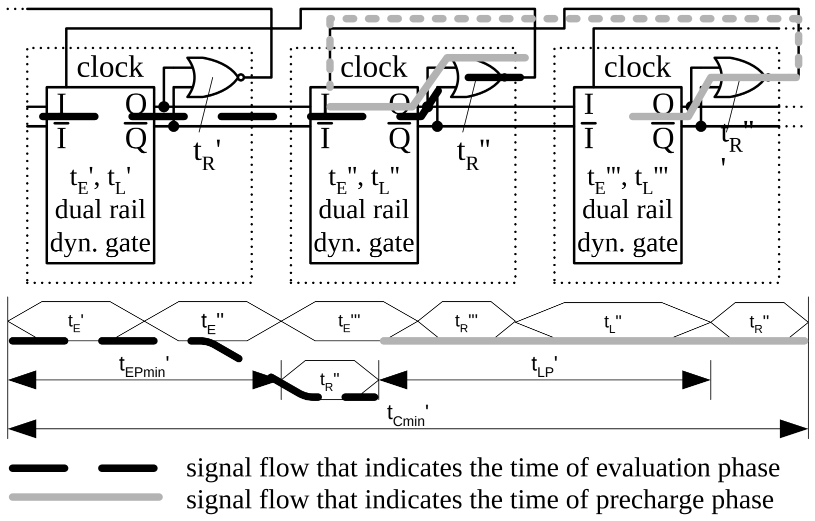
clock
clock
clock
t_E', t_L' dual rail dyn. gate
t_E'', t_L'' dual rail dyn. gate
t_E''', t_L''' dual rail dyn. gate
t_R'
t_R''
t_R'''
t_{EPmin}'
t_{LP}'
t_{Cmin}'
signal flow that indicates the time of evaluation phase
signal flow that indicates the time of precharge phase

Figure 3: dual-rail self-timed structure and timing chart of the precharge and evaluation phase of the first logic level

inputs during the evaluation phase. Therefore, the evaluation time is only the sum of the gate evaluation times with no additional delays. The calculations start with the first gate and propagate the chain without stopping. Figure 3 shows such a dynamic dual-rail self-timed structure. For a dual rail structure with DOMINO [2], [3] or DCVSL [5] a completion signal from a NOR can be directly used as the clock signal for the previous gate.

2.3 AC-TSPC used in this work

To integrate small asynchronous chains of logic in a synchronous design like AC-TSPC, a single phase global clock with same duration of high and low phase clocks the last gate in the chains. The previous gates are connected via the self-timed scheme. If the runtime of the chain is nearly half the clock cycle time, the evaluation will be delayed before the last gate. Therefore, the runtime will be increased. But this does not corrupt the function. The resulting structure, Asynchronous Chain True Single Phase Clock (AC-TSPC), is a pipeline with asynchronous

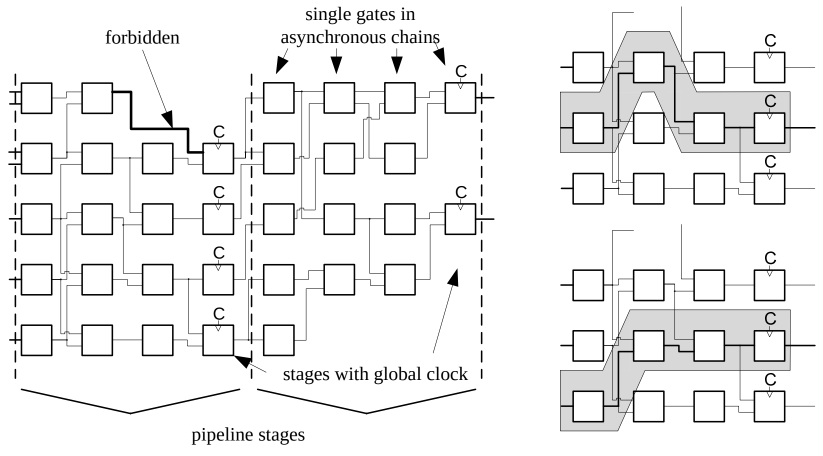

Figure 4: structure of AC-TSPC circuits; chains in the structure

chains as pipeline stages (depicted in figure 4). The pipeline stages are controlled by the global clock. The gates within the asynchronous chains, the pipeline stages, are classified in chain stages, where the outputs of the gates must connect only to gates in the following stage. As a gate can have more than one gate on its outputs, this gate can be a part of different chains. To cope with this, virtual chains are established (figure 4). These virtual chains present all possible combinations of gates within the circuit to chains. We need these virtual chains for the determination of the timing behavior of the circuit.

3.0 Developed Design Flow

This paper presents a synthesis strategy for realizing larger designs using the self-timed logic structures which were developed in "Dynamic Single Phase Logic with Self-timed Stages for Power Reduction in Pipeline Circuit Designs"[4]. Until now, the ability to apply AC-TSPC structures to larger designs has been restricted by the missing automatic timing analysis. We present a method to combine a Synopsys synthesis library with three post-Synopsys-synthesis

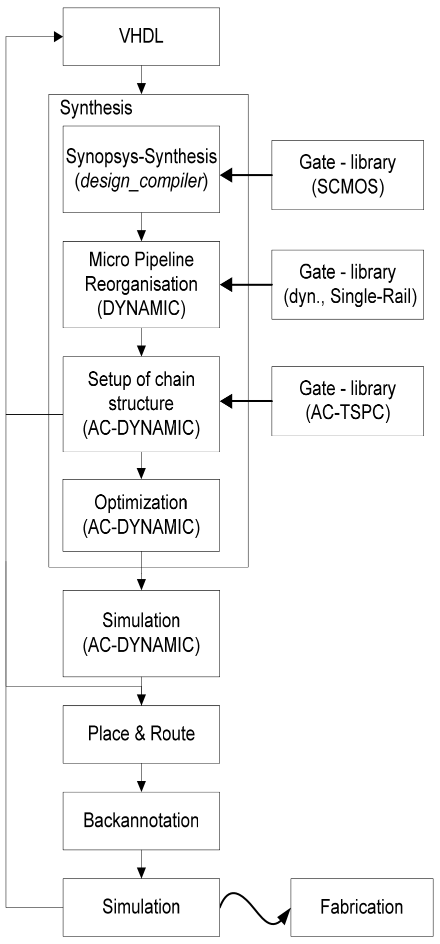
VHDL
Synthesis
Synopsys-Synthesis (design_compiler)
Gate - library (SCMOS)
Micro Pipeline Reorganisation (DYNAMIC)
Gate - library (dyn., Single-Rail)
Setup of chain structure (AC-DYNAMIC)
Gate - library (AC-TSPC)
Optimization (AC-DYNAMIC)
Simulation (AC-DYNAMIC)
Place & Route
Backannotation
Simulation
Fabrication

Figure 5: Generic Design flow for AC-TSPC

tools. The Synthesis design flow was spilt into 5 steps:

- Make a new Synopsys synthesis library
- Synthesis done with Synopsys' design_compiler
- Micro Pipeline Reorganization (MPR) done with DYNAMIC
- Chain structure setup step done with AC-DYNAMIC
- Optimization step done with AC-DYNAMIC

This design flow is depicted in figure 5.

3.1 Development of a design libraries

All necessary gates and logic functions have to be included in a new Synopsys [6] synthesis library. The library compiler was used to create a library containing the simulated results as timing values, structure, and logical function. *HSPICE* was used for simulations. For every static gate, a functional equivalent dynamic single-rail gate was designed. The results were included in a design library. The dynamic dual-rail gates with completion detection were designed. Additionally we collected the values for the minimum and maximum time for evaluation, precharge, and generating the self-timed signals.

3.2 Automatic design of dynamic pipeline

The automatic synthesis of the dynamic pipeline parts requires some additional steps. In figure 6, the strategy for an example is shown. The basic VHDL description includes the combinational functionality only and the verification of this description is easy. Next, the compilation of these parts with the design compiler using the developed library of static gates with corresponding information can be made. At this point, a netlist exists with combinational logic only and without any clocking signal (left side of figure 6). This netlist has to be processed with the Micro Pipeline Reorganization (MPR) tool DYNAMIC [8]. Because every dynamic gate includes a register function, the tool can handle each logic gate as a pipeline stage. It replaces the combinational gates with equivalent single-rail dynamic gates. Then it includes simple registers in single wires to ensure the timing behavior, because each signal has to be registered in every clock cycle (right side of figure 6). The tool DYNAMIC finally generates a netlist for the fully pipelined logic.

The condition for the use of the pipeline tool is the existence of a combinational netlist without any recursive connections. This means that no combinational feedback may exist, because the tool needs a well defined starting point and ending point for each signal path. Where a combinational loop can be split into a forward part and a feedback part, the forward part can also

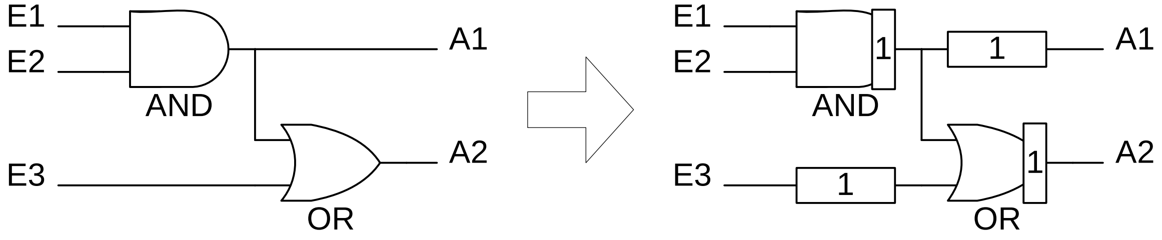

Figure 6: Example of the conversion of combinational logic into a pipeline

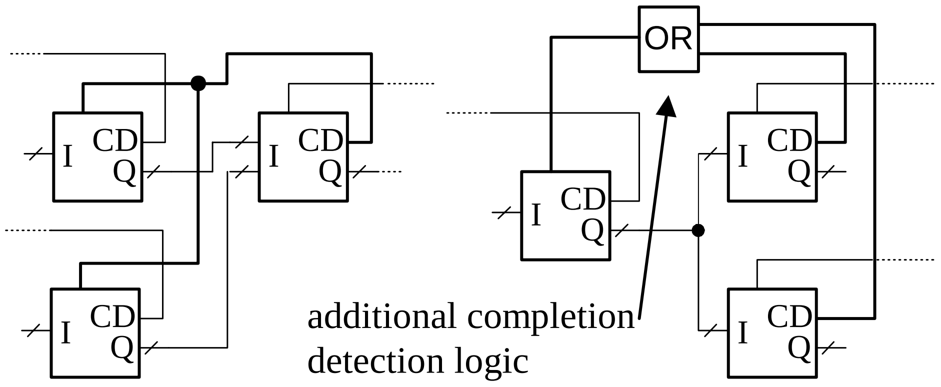

Figure 7: Problems of completion detection due to interconnecttions between consecutive gates; each block represents a dynamic dual rail gate with completion detection as shown in figure 3.

be processed with the DYNAMIC tool.

3.3 Setup of asynchronous chains

The tool AC-DYNAMIC reads the netlist of the pipelined design and creates the AC-TSPC structure (figure 4). The dynamic single-rail gates are displaced by functional equivalent dynamic dual-rail gates with completion detection.
In the next step all input gates of the circuit are connected with the global clock signal. Then, the last gates in the chains are connected with the global clock signal also. In the next step the completion detection signals are connected with the control inputs. Due to interconnections of gates, the generation of self-timed signals used as clock signals for previous gates is not trivial. To ensure the correct order of events in the self-timed scheme, a clock signal must arrive in a defined period of time. For interconnected gates, the time periods of all participated gates must be respected. There are two main cases (figure 7). First, a single completion detection signal is used as clock signal for two or more previous gates (figure 7; left). This restricts the timing behavior of the gates in the chain stage before the gate, which generates the completion detection signal. The timing behavior of all previous gates, which are connected to the self-timed signal, must nearly be the same. Second, there are two or more gates which are interconnected to the output signal of a single gate (figure 7; right). In this case, there are more than one completion signals that can be connected to the clock input. These signals must be mostly combined with additional logic, e.g., an OR gate. This logic has to ensure that the gate, which generates the output signal, does not change to precharge phase until all following gates have completed their evaluation and generated their completion signals. The tool evaluates the necessary logic and inserts it.

3.4 Optimization of AC-TSPC

The optimization goals are low latency, high maximum frequency and low area. The tool AC-DYNAMIC can vary four parameters. These are the length of the chains, the completion detection logic, and the length of the low and high phase of the clock signal. The tool calculates reference values for every configuration, which is tested. At first, the maximum clock frequency for every possible chain length and for a symmetric or asymmetric clock signal is evaluated. The timing values of all gates and the necessary demands are used for this step. In the next step, the

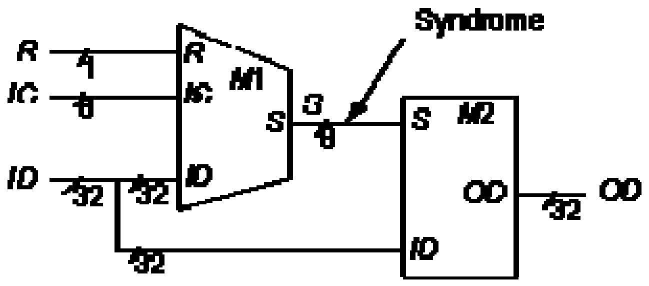

Figure 8: ISCAS-85 C499/C1355 32-Bit Single-Error-Correcting Circuit

logic for completion detection is optimized. In the last step, the tool searches for chains, which consist only of buffers. These chains can be replaced by a single TSPC buffer gate, because the outputs of this gate have the same behavior as the outputs of a buffer chain.

3.5 Simulation

After the tool AC-DYNAMIC evaluates the best configuration for the chain length, clock frequency, and completion detection logic, the timing behavior of the circuit is simulated. Because the exact evaluation and precharge timing values of a single gate depend on input vectors, temperature and process parameters for each gate a minimum and a maximum evaluation or precharge time can be specified. The tool determines for each gate in a chain these minimum and maximum timing values and verifies that the self-timed signals do not violate these limits. To simulate the functional behavior, we use *HSPICE*.

4.0 Example

As an example we use the ISCAS-85 c1355 32-bit single-error-correcting circuit [9]. The c1355 circuit has 41 inputs, 32 outputs and 546 gates and is a single-error-correcting circuit. The 41 inputs are combined to form an 8-bit internal bus S, which is then combined with 32 primary inputs to form the 32 primary outputs (figure 8) [10]. The results in table 1 show that the static implementation has a lower area (15% of the AC-TSPC implementation). But the maximum

	SCMOS	AC-TSPC
Gates	546	2098 - (AC-TSPC) 184 - (logic for completion detection)
Max. Clock frequency	193 MHz	528 MHz
Latency	5.15 ns	7.57 ns

Table 1: comparison of results for different implementations of the ISCAS c1355 benchmark

clock frequency of the AC-TSPC version of the circuit depends 270% of the SCMOS version. It must be considered, that the values for the static version are measured without flip-flops.

5.0 Conclusions and Recommendations

This paper presents a solution for a completely supported design flow of Asynchronous Chain (AC)-TSPC logic. This logic style combines latch-free evaluation with fast dynamic logic to reach maximum throughput in high performance applications. The presented tool generates netlists consisting of dynamic gates, calculates the exact timing behavior using a developed strategy and compares these values to the functional limits given from the self-timed scheme. Therefore, this tool overcomes the lack of a missing synthesis and exact timing analysis. The results of the extensive work on timing calculation and on implementation are the very first expressive statements of the applicability of AC-TSPC. The simulation results of the tool were validated with transistor level simulations.

6.0 Acknowledgements

This paper has been improved by the SNUG technical committee member Kurt Baty.

7.0 References

[1] R. H. Krambeck, C. M. Lee and H.-F. S. Law, “High-Speed Compact Circuits with CMOS”, IEEE Journal of Solid-State Circuits, IEEE, Vol. SC-17, No. 3, Jun. 1982.

[2] D. Harris and M. A. Horowitz, “Skew-Tolerant Domino Circuits”, IEEE Journal of Solid-State Circuits, Vol. 32, No. 11, Nov. 1997.

[3] G. Yee and C. Sechen, “Clock-Delayed Domino for Dynamic Circuit Design”, IEEE Transactions on VLSI Systems, Vol. 8, No. 4 , Aug. 2000.

[4] F. Grassert and D. Timmermann, “Dynamic Single Phase Logic with Self-timed Stages for Power Reduction in Pipeline Circuit Designs”, IEEE International Symposium on Circuits and Systems (ISCAS), May 2001, pp. IV 144-147.

[5] L. G. Heller, W. R. Griffin, J. W. Davis and N. G. Thoma, “Cascode Voltage Switch Logic: A Differential CMOS Logic Family”, Proceedings of International Solid-State Circuits Conference, IEEE, 1984, pp. 16-17.

[6] Synopsys, Inc., 700 East Middlefield Road, CA 94043-4022 United States of America. Synopsys Synthesis and Simulation Tools, 2000 edition.

[7] F. Grassert and D. Timmermann, “Asynchronous Chain True Single Phase Clock Logik (AC-TSPC)”, 3. Schwerpunktkolloquium des DFG Schwerpunktprogramms Grundlagen und Verfahren verlustarmer Informationsverarbeitung (VIVA), ISBN: 3-00-008995-0, p.136 - 141, Chemnitz, March 2002

[8] S. Flügel, M. Grothmann, M. Haase, P. Nimsch, A. Wassatsch, H. Ploog, F. Grassert, and D. Timmermann, “A Design Flow for 12.8 GBit/s Triple DES using Dynamic Logic and Standard Synthesis Tools”, SNUG Europe, S. E3.2. 1-8, Munich, March 2001

[9] M. Hansen, H. Yalcin, and J. P. Hayes, "Unveiling the ISCAS-85 benchmarks: a case study in reverse engineering," IEEE Design and Test, vol. 16, no. 3, pp. 72-80, July-Sept. 1999.

[10] ISCAS-85 C499/C1355 32-Bit Single-Error-Correcting Circuit, http://www.eecs. umich.edu/ ~jhayes/iscas/c499.html

8.0 Appendix

Instruction for AC-Dynamic

You can find the tar file at: http://www-md.e-technik.uni-rostock.de/ma/sf01/dynamic/index.html.
It was written in Java, tested to run on linux/Suse 8.2 and Sun Solaris 5.8

You can start AC-Dynamic with different parameters. A help screen will display with

ac_dynamic –h

The chain length is fixed with the parameter *–c* followed by a number (blank). All possible chain lengths will be simulated if this parameter is not set. The parameter *–a* chooses an asymmetric clock signal, while it is not set a symmetric clock signal is used. All clean buffer chains will be transformed in a single TSPC-gate with the parameter *–bufchains*. This parameter can only set in combination with a fixed chain length. If the parameter *–debug* is set, a debug modus starts after every optimization. In this modus you can get information about every gate and the structure of the circuit. The parameter *–l* followed by a number (blank) set the quantity of information, which is displayed as the program runs. The smallest value is 0. The parameters *–no*, *-nbo* and *–uni* affect the optimization. No optimization will proceed if the parameter *–no* (no optimization) is set. The parameter *–uni* has to set if the tool should insert Uni-Logics. If the parameter *–nbo* (new buffer optimization) is set, buffers will be inserting to delay some signals. Sometimes it is needful to get all information of the gates, which the tool uses internally. You get an according netlist with the parameter *–pipe*.

You start the tool with:

ac_dynamic [parameter] XNF-file

You can choose several parameters.
You get two netlists if the tool is ready. The name of these netlists depends on the symmetry of the clock signal:

Timing_best_sym_clk.xnf and *Timing_best_latenz_sym_clk.xnf*
or
Timing_best_unsym_clk.xnf and *Timing_best_latenz_unsym_clk.xnf*

These netlists include the solutions fort he highest clock frequency and the shortest latency time.

If the debug modus is active, you get a selected menu after the evaluation for every chain length. You can choose the options by the according numbers. If you select the number 2, you get the rules. These rules specify which input signals are necessary to start an evaluation.
Over the numbers 2 to 8 and 12 to 18 you can display the times for every gate, whereas *e* means the evaluation, *p* means precharge and *i* means input signal. You get the properties of every gate

with number 10 (specify the coordinates in the array) and number 20 (specify the name of the gate). You have to choose number 21 if you want check this netlist work with a special clock frequency. Then at first you have to input the length of the low phase of the clock signal in ns, then the length of the high phase, and then the debug level. To exit choose number 0.

Format of the Timing-Information

The timing parameter has to integrate for every gate into the XNF-netlist file. If there are no parameters, the tool uses standard values (AMS 0.6µm). The syntax of the timing parameter is:

SYM, …
TIM,*pre_min*,*pre_max*,*eva_min*,*eva_max*,*ru_min*,*ru_max*,*rd_min*,*rd_max*,*clk_fall_min*,*clk_fall_max*,*clk_rise_min*,*clk_rise_max*
RULES, (port;port;…)(port;port…)(..)
…

pre means precharge phase, *eva* means evaluation phase, *ru* means ready up (evaluation time for self timed signal after gate is ready with precharge phase), *rd* means ready down (evaluation time for self timed signal after gate is ready with evaluation), *clk_fall* and *clk_rise* means the delay of the clock signal if the gate is the last gate in a chain. RULES means which input signal are necessary to start an evaluation.

The file *timing.cfg* includes the timing values for the self timed logic, the ports of the circuit, and for additional buffers. *uni* means the parameter of the universal logic. *vb* is the prefix for the parameter of the buffer, which have to input to complete the last chains.

If all gates should have the same parameter, use the tool new_timing:

new_timing <eva> <eva_master> <eva_diff> <ru> <ru_diff> <pre> <pre_master> <pre_diff> <rd> <rd_diff> <or> <or_diff> <filename>

You can set special parameters for the last 3 gates in a chain (called master gates) with the *_master_* parameter. *or* means the parameter for a 2-input-or. These values will extrapolate for or-gates with more inputs. *diff* means the positive and negative difference from the set time.