DYNAMIC - A Java Based Toolset For Integrating Dynamic Logic Circuits Into A Standard VLSI Design Flow

A. Wassatsch, D. Timmermann

University of Rostock
Department of Electrical Engineering and Information Technology
Institute of Applied Microelectronics and Computer Science
Richard-Wagner-Str. 31, D-18119 Rostock, Germany
Tel./Fax.: ++49+381 498 3534/3601
wassatsch@e-technik.uni-rostock.de

INTERNATIONAL CADENCE USER GROUP CONFERENCE
September 10-13, 2000
San Jose, CA, USA
Outline

- Motivation
- Basics of dynamic circuit technologies
- Design requirements
- The Toolset
- Performance analysis
- Applications
- Conclusion
Motivation

Why Java for EDA applications?

– portability:
  * write once run everywhere
  * same version for all systems
  * reduction of the overall maintenance costs

– and the runtime ?:
  * just in time (jit) compiler technologies delivers the power
  * careful implementation and optimization can further accelerate execution time
  * for the minority of high-end applications like logic synthesis of large designs the performance of conventional systems is by no way sufficient

Java is not the solution for everything, but already well for a majority of the given tasks in EDA!
Motivation (2.)

Why replace CMOS with dynamic circuits?

- speed:
  - some of the fastest microprocessor designs like Compaq Alpha and the 1GHz Prototype from IBM utilize dynamic circuit technology
  - halved fan-in

- area:
  - count of transistors for a n-input gate with register:
    \[ 11 + n < 16 + 2n \] (dynamic circuit:CMOS)
  - but increased expense in the clock-tree network

- power consumption:
  - reduced dependency on clock frequency compared to CMOS
  - current consumption is determined by the signal value and not by the rate of signal value changes

- True Single Phase Clock (TSPC) as circuit technology for standard cells:
  - solid behavior of the cell signals
  - digital specification of the cell behavior possible

TSPC isn’t an universal remedy for every design!
Why do we have to expand the design flow?

- there is at this time, no support through commercial tools for logic synthesis with dynamic circuits style standard cell libraries
- differentiation of combinatorial and sequential cells, a syn lib must have (N)OR/(N)AND, INV and a register element
- “Schematic entry” of the design → manual work, isn’t appropriate for a modern design flow
- Workaround 1: structural HDL-description with respect to the demands of dynamic logic → like schematic-entry
- Workaround 2: separate instantiation of logic and pipeline register file through the HDL-description, reordering of the pipeline after the logic synthesis through “balance-registers” → resulting netlist not applicable for TSPC circuit style

development of a design flow extension necessary!
Basics of dynamic circuits

- Conflation of combinatorial and sequential elements into one cell
- No pure combinatorial blocks possible
- Processing depends on the clock
- Operational principle based on capacitive carrier storage
- Logic function is implemented in only one tree of transistors
- Examples: C\textsuperscript{2}MOS, TSPC, Domino-Logic, CVSL, DCVSL, DCSL
**True Single Phase Clock - Logic**

- **Advantages:**
  - only one clock signal necessary
  - minor fan-in stress with only one transistor tree
  - utilization of both clock signal phases through alternating activation
- **Disadvantages:**
  - increased load on the clocking signal net
- Temporally separated operating conditions
  - pre-charge: storage of a small charge quantity on internal node
  - evaluate: logic-dependent discharge
Differential TSPC

- complementary signal generation
- derived from the TSPC logic circuit
- easily cascade-able due to durable behavior

Advantages:
- complementary logic function representable
- abridgement of pipeline depth through parallel calculation

Disadvantages:
- increased load of clock and signal lines
- doubled implementation area and current consumption
Design requirements

- concerning latency uncritical and tolerant circuit environment
- particular suitable circuit architectures
  - arithmetic operators with bit-width-independent run time (CS-, SD-adder)
  - pipeline-able signal processing with non-recursive data flow
    - digital filter
    - spacial developed iterative algorithms (CORDIC, DES)
- unfavorable circuit architectures
  - runtime based elements (mono-flop, RS-flipflop)
  - architectures with strongly pronounced internal serial dependencies (Ripple Carry Adder (RCA))
  - feedbacks in data flow over more than one cell level (counter)
Minimization of the power consumption

- not the modification of an internal status consumes current, but the continuous revitalization of this status ⇒ DRAM
- power consumption depends on the distribution of the signal level frequency and not of the signal level change frequency
- possibilities for the reduction of power consumption:
  - by architecture selection
  - by changed optimization target
  - by adapted cell library
Minimization of the power consumption (2.)

influence of the selected architecture

- choosing of consumption-minimal state codings for incomplete code space allocation
- one hot encoding for state machines
- ring counters

example 4 bit vector

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
<td>0001</td>
<td>0011</td>
<td>0111</td>
<td>1111</td>
</tr>
<tr>
<td>1</td>
<td>0010</td>
<td>0101</td>
<td>1011</td>
<td>1101</td>
<td>0001</td>
</tr>
<tr>
<td>2</td>
<td>0110</td>
<td>0110</td>
<td>1101</td>
<td>0011</td>
<td>0001</td>
</tr>
<tr>
<td>3</td>
<td>1001</td>
<td>1010</td>
<td>1110</td>
<td>1110</td>
<td>0001</td>
</tr>
<tr>
<td>4</td>
<td>1000</td>
<td>1010</td>
<td>1100</td>
<td>1100</td>
<td>0001</td>
</tr>
</tbody>
</table>

- ring counter \( \frac{1}{m} \) active \( \Rightarrow 1 \)
- binary counter 0.5 active \( \Rightarrow 0.5n \)
The DYNAMIC Toolset

- written in JAVA
  - executable on any system with the necessary java runtime environment
  - web based execution possible → web based design offers
    - with jini/rmi reorganization of the design environment concept
- input filters written with JavaCC/JTree
  - easy creation of input filters for new formats
  - similar to lex/yacc for C
- threaded operation mode, “parallel” execution of sequential tasks
- modularized structure, one block for each task
- command line and gui interface
### Performance analysis

#### speed up facts:
- optimization of the utilized algorithms
- quality of the runtime environment

JIT technology brings sufficient run times

speed scales well with clocking frequency over the different target systems

<table>
<thead>
<tr>
<th>Design name</th>
<th>Design-size Cells</th>
<th>/Pipes/ Cells</th>
<th>runtime in seconds Sparc 400MHz/2G Solaris C/Java/JIT</th>
<th>runtime in seconds Athlon 500MHz/256M Linux Java/JIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>add_rpl16</td>
<td>44/29/1115</td>
<td></td>
<td>673/52/19</td>
<td>29/17</td>
</tr>
<tr>
<td>add_cla16</td>
<td>216/10/479</td>
<td></td>
<td>90/17/20</td>
<td>9/8</td>
</tr>
<tr>
<td>add_clf16</td>
<td>353/11/799</td>
<td></td>
<td>231/35/21</td>
<td>21/15</td>
</tr>
<tr>
<td>add_bk16</td>
<td>140/11/444</td>
<td></td>
<td>84/15/19</td>
<td>8/6</td>
</tr>
<tr>
<td>add_rpl32</td>
<td>83/1/4029</td>
<td></td>
<td>nn/439/20</td>
<td>262/136</td>
</tr>
<tr>
<td>add_cla32</td>
<td>440/16/1591</td>
<td></td>
<td>1248/85/24</td>
<td>50/32</td>
</tr>
<tr>
<td>add_clf32</td>
<td>590/13/1429</td>
<td></td>
<td>750/75/31</td>
<td>42/30</td>
</tr>
<tr>
<td>add_bk32</td>
<td>336/13/1080</td>
<td></td>
<td>508/48/21</td>
<td>26/19</td>
</tr>
<tr>
<td>mult_cs8</td>
<td>288/23/1173</td>
<td></td>
<td>713/57/11</td>
<td>37/25</td>
</tr>
<tr>
<td>mult_cs12</td>
<td>628/32/2684</td>
<td></td>
<td>nn/191/26</td>
<td>137/91</td>
</tr>
<tr>
<td>mult_w8</td>
<td>574/23/1390</td>
<td></td>
<td>758/68/13</td>
<td>45/31</td>
</tr>
<tr>
<td>mult_w12</td>
<td>1141/30/2905</td>
<td></td>
<td>3494/196/28</td>
<td>140/96</td>
</tr>
<tr>
<td>des_slice</td>
<td>2046/27/6680</td>
<td></td>
<td>nn/896/132</td>
<td>3186/435</td>
</tr>
<tr>
<td>des_pipe</td>
<td>32920/241/119964</td>
<td></td>
<td>nn/22h13m</td>
<td>nn/nn</td>
</tr>
<tr>
<td>key_56</td>
<td>1680/9/4802</td>
<td></td>
<td>nn/551/89</td>
<td>416/364</td>
</tr>
</tbody>
</table>
TSPC design flow

- integration of the dynamic circuit technology into a standard CMOS design flow
- encapsulation of the particular dynamic characteristics into simulation and synthesis libraries
- deployment of standard cell technique
Micro Pipeline Reorganizer (MPR)

- **target**: Improvement of the pipeline throughput of a given circuit
- **task**: en-queuing of a pipeline structure by insertion of additional register cells
- **requirements**:
  - basic cells with only on output signal
  - no feedbacks in the netlist
- **netlist formats**: XNF, struct. VHDL, EDIF
optimization begins at output ports of the network with the labeling of the net-level
acquisition of the driving cell outputs
verification of the net-level values of the associated cell inputs
if necessary, insertion of buffer cells (register)
increment of the current net-level value
repeat until all regarded networks are driven only by the input ports of the net-list
Applications

- Pipeline expansion due to strong internal serial signal dependency $\Rightarrow$ factor $2^n - 2$

- Has a pipeline effectiveness degree $E = \frac{b}{m} \sum_{i=1}^{m} \frac{N_i}{n_i} = \frac{8}{14} \times \frac{9.62}{14} = 0.392$
Design example: MPR 8x8 CSA Multiplier

- 8bit x 8bit Carry-Save Multiplier from the Synopsys Design-ware
- increase of the pipeline levels
- pipeline effectiveness degree

\[ E = \frac{b}{m} \sum_{i=1}^{m} \frac{N_i}{n_i} \]

\[
\begin{align*}
E &= \frac{16}{23} \cdot \frac{20.98}{23} = 0.635
\end{align*}
\]
DTSPC example: Signed Digit Adder Cell

0.6μ 5V AMS-CUB Technology

SD-Addition in one clock cycle in opposition to 8 clock cycles with single cell implementation.
Design example: TDES

- Streaming data (de/)encryption with Triple DES
- 0.6μ 5V AMS-CUB Technology @ 200/800MHz
- Tree different design styles utilized
  - Standard CMOS for control logic
  - Circuits with manually arranged TSPC cells
  - Automatically synthesized DES-pipeline constructed by TSPC cells

- Core pipeline implements the 16 DES stages with 241 rows, approx. 30000 gates
Alternative application types

- Acceleration of standard CMOS Designs
  - Increase of the performance by automatically generated pipeline structures, pipeline depth is determined by the netlist, no specification from the designer needed/possible as with the “balance-register”-approach
  - Deployment for FPGA-development: one pipeline stage per clb

- Implementation of wave pipelines
  - Approach: exchange of the register functionality by simple delay elements (buffers)
  - For FPGA development: specification of the necessary routing delays simplified by finer granularity of the netlist
Conclusion

- application of dynamic circuit technology requires utilization of adapted circuit architectures
- integration in standard CMOS design flow possible by partial extension
- reference implementation of a TSPC standard cell library based on AMS $0.35\mu m$ or $0.6\mu m$ Technologies
- implementation of reference designs (CORDIC, Digit-Online-Neuro, DES)
- evaluation of alternative application type for the toolset
- development of a standard cell generator for TSPC