Power Reduction in Digital Circuits
Ph.D. Defense

Jan Láník

16th June 2016
CIFRE Ph.D. Thesis
Verimag (University of Grenoble)
Industrial partner: Atrenta/Synopsys
Supervisors: Oded Maler (Verimag), Fahim Rahim (Synopsys)
2012-2016
Introduction
Motivation

Power consumption of integrated chips is an issue

Our work: yet another attempt to reduce power consumption
Hardware synthesis

Hardware analog of a compilation in software

- High level description → Silicon realization

- Crucial step in hardware production
- Optimizations for speed, space and power
- Many intermediate steps
- Many degrees of freedom
Some Steps in the Hardware Synthesis

Behavioral level

```
always@(posedge clk or posedge rst)
begin
  if (rst or cnt==MAX) cnt <= 0;
  else
    begin
      cnt <= cnt + 1;
    end
end
```

RTL

```
(x<MAX) || rst ? x+1 : 0
```

Physical Layout

Netlist
Switching power dissipation at a gate

\[ P = \frac{1}{2} V_{dd}^2 C_i E_i f \]

- \( V_{dd} \) \ldots supply voltage
- \( C_i \) \ldots capacitance connected to the output of gate \( i \)
- \( E_i \) \ldots switching activity (number of switches per cycle) of gate \( i \)
- \( f \) \ldots clock frequency
Two methods for switching activity reduction:

1. **Power Aware Synthesis** Optimization of combinatorial logic synthesis

2. **Activity Triggers** Optimization of sequential logic blocks by clock gating (industrial project)

The two methods are independent. Common points: RTL/Netlist level, modeling input to achieve statistical switching reduction.
Power Aware Synthesis
Our place in the synthesis flow

1) multilevel logic specification

\[ X = a \cdot b \]
\[ Y = \overline{b} + c \]
\[ Z = X + Y \]

2) AIG

3) Technology dependent representation

\[ a \]
\[ b \]
\[ c \]
AIG (AND-Inverter graph)

- Acyclic directed graph
- Nodes: AND and NOT gates
- Efficient representation
- Not canonical
- Many optimizations

Our method: yet another optimization on the AIG level
We want to optimize AIGs by re-arranging AND cones

AND cones in AIG

Referred by an inverter

Referred twice
2 ways to realize 8AND by 2ANDs

- we assume synchronized design, 0 time delay
- 1 switch = change of value at a gate output
- gate values determined by input values

we assume synchronized design, 0 time delay
1 switch = change of value at a gate output
gate values determined by input values
Input stream and switching

- A circuit sees more than one transition during its lifetime
- Input stream: sequence of values as they are applied to the circuit inputs

\[
\text{Input stream} + \text{Internal structure} = \text{Actual switching}
\]

What is a ‘typical’ input stream?

\[
\text{Input model} + \text{Internal structure} = \text{Expected switching}
\]
Input model

- Ideally: Markov chain
- Realistically: Long input stream provided by designers
Optimization and evaluation flow
An AND cone is semantically equivalent to an $n$-input AND gate

Goal: find 2AND realization for the given cone with a minimal switching w.r.t. the typical (training) stream

Constrained to minimal-depth 2AND (timing)
Solution:

1. **Enumerative**
   - Problem space growing fast
   - Efficient up to approximately 8 inputs (symmetry reduction)
   - Sufficient for most of the cones in real designs

2. **Layer based approximation**
   - Optimal on ‘layers’
   - Locally optimal
   - Efficient for larger cones
Balanced and unbalanced trees

![Balanced Tree](image1)

![Unbalanced Tree](image2)
<table>
<thead>
<tr>
<th>in</th>
<th>all trees</th>
<th>canonical</th>
<th>balanced</th>
<th>canonical &amp; balanced</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
<td>3</td>
<td>12</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>120</td>
<td>15</td>
<td>24</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>1680</td>
<td>105</td>
<td>480</td>
<td>30</td>
</tr>
<tr>
<td>6</td>
<td>3.0240e+04</td>
<td>945</td>
<td>4320</td>
<td>135</td>
</tr>
<tr>
<td>7</td>
<td>6.6528e+05</td>
<td>1.0395e+04</td>
<td>2.0160e+04</td>
<td>315</td>
</tr>
<tr>
<td>8</td>
<td>1.7297e+07</td>
<td>1.3514e+05</td>
<td>4.0320e+04</td>
<td>315</td>
</tr>
<tr>
<td>9</td>
<td>5.1892e+08</td>
<td>2.0270e+06</td>
<td>2.9030e+06</td>
<td>1.1340e+04</td>
</tr>
<tr>
<td>10</td>
<td>1.7643e+10</td>
<td>3.4459e+07</td>
<td>1.0161e+08</td>
<td>1.9845e+05</td>
</tr>
<tr>
<td>11</td>
<td>6.7044e+11</td>
<td>6.5473e+08</td>
<td>2.2353e+09</td>
<td>2.1830e+06</td>
</tr>
<tr>
<td>12</td>
<td>2.8159e+13</td>
<td>1.3749e+10</td>
<td>3.3530e+10</td>
<td>1.6372e+07</td>
</tr>
<tr>
<td>13</td>
<td>1.2953e+15</td>
<td>3.1623e+11</td>
<td>3.4871e+11</td>
<td>8.5135e+07</td>
</tr>
</tbody>
</table>
Each pairing of input signals into an AND gate produces certain switching number. Minimizing the switchings in the first level corresponds to minimal perfect matching in a weighted graph $O(n^3)$, Edmonds65, Lawrer76.
Examples of suboptimality

A level-greedy pairing with 6 switches

An optimal pairing with 5 switches
Examples of suboptimality: missing topologies

(a)

(b)
We evaluate on 2 classes of examples:

1. **Synthetic products of Markov chains**
   - Different forms of interaction/correlation between variables
   - Another parameter characterizes the amount of randomness/determinism

2. **Verilog models of 2 small designs**
   - A simple decoder for a hand held calculator
   - A Serial Peripheral Interface from Opencores
Synthetic examples

![Graph showing relative switching reduction vs. uniformity parameter α]

- Red dashed line: Partition 2
- Black solid line: Partition 4
Small realistic circuits

### Net effect of our method on AIG level

<table>
<thead>
<tr>
<th></th>
<th>maximum</th>
<th>minimum</th>
<th>level-greedy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mini Instruction Decoder</td>
<td>250338</td>
<td>128118</td>
<td>158726</td>
</tr>
<tr>
<td>Core SPI (opencores)</td>
<td>20577</td>
<td>19681</td>
<td>19681</td>
</tr>
</tbody>
</table>

### Interference due to other optimization methods

<table>
<thead>
<tr>
<th></th>
<th>maximum</th>
<th>minimum</th>
<th>level-greedy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mini Instruction Decoder</td>
<td>73976</td>
<td>72288</td>
<td>72288</td>
</tr>
<tr>
<td>Core SPI (opencores)</td>
<td>19555</td>
<td>19233</td>
<td>19233</td>
</tr>
</tbody>
</table>
Shortcomings

- Preprocessing diminishes the savings
- Unclear if reduction preserved on mapped netlist
Activity Triggers
Clock gating

Disabling registers when not needed by ‘gating the clock’ to save power

Problem: How to compute the enabling condition?
Clock gating conditions - granularity

Coarse grained
- Design decision
- On the high level - whole functional blocks
- Handcrafted enable conditions
- Efficient, easy to implement, high in the clock tree

Fine grained
- Small register groups deep in the designs
- Complex, not intuitive
- Need tools to find them
- Can be expensive
Clock gating conditions - granularity

**Intermediate**

- Missing link
- Medium sized blocks
- Understandable, but not necessarily obvious
- Human designer should be able to find them if he did a time consuming detailed analysis
- Tools in demand
Activity Triggers

Events related to a change of activity status of a design block.

\[ \alpha \quad \beta \]

MODULE ACTIVE

MODULE IDLE
Example
UART - serial line transmission

- **line idle**
- **start bit**
- **data transmission**
- **two stop bits**

1 1 0 1 0 0 0 0
Tool components

1. Detection of potential triggers
2. Formal verification (my contribution)
Statistical detection

Correlation analysis performed on vcd or fsdb traces generated from simulation.
For detection we consider only events that are bit/bus transitions.
E.g. a signal $x$ going from 0 to 1 or a 4-bit bus $Y$ going from $4'b0001$ to $4'b0010$

1. design decomposition
2. idle periods detection
3. potential events filtering (based on size and sequential distance)
4. ranking potential events (coverage and ran measures)
Potential start and stop signals location

- *Stop events* · · · in a short window before the beginning of stable periods
- *Start events* · · · in a short window before the end of stable periods
Coverage ⋅⋅ the ratio of idle periods that are correlated with the event

Noise ⋅⋅ the ratio of events that are ‘out of place’
An activity trigger is a triple \((\alpha, \beta, d)\)

- \(\alpha\) . . . start condition
- \(\beta\) . . . stop condition
- \(d\) . . . shutdown delay
Monitor automaton

\[
\neg \beta \lor \alpha
\]

\[
\beta \lor \neg \alpha
\]

\[
\alpha
\]

\[
\neg \alpha
\]

\[
\alpha \land \text{stable}
\]

\[
\neg \text{stable}
\]

\[
d - 1 \text{ delay nodes}
\]

MODULE ACTIVE

MODULE IDLE

PROPERTY VIOLATION
Formalization (PLTL)

Stability of a signal:

$$stable(x) = (x \land \oplus(x)) \lor (\neg x \land \ominus(\neg x))$$

Stability of a block (set of signals):

$$stable(M) = \bigwedge_{x \in M} stable(x)$$
Formalization (PLTL)

Stop Condition:

\[ \ominus^d \beta \land \bigwedge_{i=0}^{d} \ominus^i \neg \alpha \]

Start condition:

\[ \alpha \]
We need to prove:

$$\neg \alpha S \left( \ominus^d \beta \wedge \bigwedge_{i=0}^{d} \ominus^i \neg \alpha \right) \Rightarrow stable(M)$$
Stability modeling

\[ \text{stable}(\text{reg}_{n-1}) \]

\[ \land_i \text{stable}(\text{reg}_i) \]

\[ \text{stable}(\text{reg}_n) \]
Monitor automaton

\[ \neg \beta \lor \alpha \]

\[ \beta \lor \neg \alpha \]

\[ d-1 \text{ delay nodes} \]

\[ \neg \alpha \]

\[ \neg \alpha \land \text{stable} \]

\[ \alpha \land \text{stable} \]

\[ \neg \text{stable} \]

\[ \text{PROPERTY VIOLATION} \]
Formal verification flow

1. Original RTL $\Rightarrow$ circuit representation
2. Stability modeling and monitor automaton added to the circuit
3. Verification using reachability engines from ABC (BMC + PDR)
Constraint support

- Implicit assumptions often not present in the design
- Can be added in form of assertions
- Typical cases: configuration registers or input following specific pattern
- This is crucial to avoid spurious counter examples
- We support System Verilog Assertions (SVA)
Automatic flow
Semi-automatic flow

1. Add constraints
2. RTL
3. Find candidates
4. Simulation .vcd/.fsdb
5. For each candidate
   - Formal check
     - Counterexample
     - Timeout
     - Proof
6. Spurious
7. YES
SENDS – A video processing architecture
SENDS results

![Bar chart showing Mean Power (mW) for 3x3 and 5x5 pixels neighborhood before and after power optimization.]

- **Mean Power (mW)**
  - 3x3 pixels neighborhood: 4.68 mW
  - 5x5 pixels neighborhood: 10.57 mW

- **Mean Power after power optimization (mW)**
  - 3x3 pixels neighborhood: 3.11 mW
  - 5x5 pixels neighborhood: 6.36 mW
# Automatic mode results

<table>
<thead>
<tr>
<th>design</th>
<th>power</th>
<th>power covered</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4.169 mW</td>
<td>75.84%</td>
</tr>
<tr>
<td>2</td>
<td>7.163 mW</td>
<td>54.93%</td>
</tr>
<tr>
<td>3</td>
<td>0.479 mW</td>
<td>49.62%</td>
</tr>
<tr>
<td>4</td>
<td>7.145 mW</td>
<td>49.47%</td>
</tr>
<tr>
<td>5</td>
<td>5.314 µW</td>
<td>31.04%</td>
</tr>
<tr>
<td>6</td>
<td>0.606 mW</td>
<td>16.30%</td>
</tr>
<tr>
<td>7</td>
<td>8.891 mW</td>
<td>15.58%</td>
</tr>
<tr>
<td>8</td>
<td>92.491 mW</td>
<td>6.77%</td>
</tr>
<tr>
<td>9</td>
<td>55.851 mW</td>
<td>4.54%</td>
</tr>
<tr>
<td>10</td>
<td>92.444 mW</td>
<td>2.87%</td>
</tr>
<tr>
<td>11</td>
<td>1.430 µW</td>
<td>1.61%</td>
</tr>
<tr>
<td>12</td>
<td>4.079 mW</td>
<td>0.70%</td>
</tr>
<tr>
<td>13</td>
<td>149.955 mW</td>
<td>0.61%</td>
</tr>
</tbody>
</table>
Conclusions
Activity Triggers: Summary

- New class intermediate-block-size clock gating conditions
- Heuristic detection based on trace analysis
- **Formal proof of validity**
- Semi-automatic and automatic methodology
- Integrated within a commercial tool
Activity Triggers: Limitations

- Constraints
- Room for improvement in scalability
- Room for improvement in detection
Power Aware Synthesis: Summary

- Power Aware Synthesis
- Optimizing switching in combinational logic
- Average case optimization
- Implemented as prototype within ABC
- Experimented on synthetic as well as realistic models
- Works well on AIG level
Power Aware Synthesis: Limitations

- Interference from other optimization techniques
- Unclear if switching reduction can be preserved in the next synthesis steps