Power Aware Combinational Synthesis
HVC 2015

Jan Láník*, Oded Maler*

*CNRS and The University of Grenoble

19th November 2015
Motivation

Power consumption of integrated chips is an issue.

Our work: yet another attempt to reduce power consumption at the gate level.
Switching power dissipation at a gate

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

\[ V_{dd} \text{ . . . supply voltage} \]
\[ C_i \text{ . . . capacitance connected to the output of gate } i \]
\[ E_i \text{ . . . switching activity (number of switches per cycle) of gate } i \]
\[ f \text{ . . . clock frequency} \]

Our method: Optimizing for small average \( E_i \) during the hardware synthesis.
Hardware synthesis

Hardware analog of a compilation in software

- High level description $\rightarrow$ Silicon realization

- Optimizations for speed, space and power
- Many intermediate steps
- Many degrees of freedom
Our place in the synthesis flow

- Synthesis of combinatorial logic from arbitrary boolean functions to technology independent network of AND gates and inverters
- Optimizing for minimal (expected) switching in the gates
- Without compromising space/speed optimization
AIG (AND-Inverter graph)

- An acyclic directed graph
- Nodes = AND and NOT gates
- Efficient representation for manipulating Boolean functions
- Not canonical (unlike BDDs)
- Used for optimization and verification
AIG within synthesis flow

1) multilevel logic specification

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

2) AIG

3) Technology dependent representation

\[ \text{INVBC5} \]
\[ \text{2NANDXU6} \]
\[ \text{2ANDXU37} \]
\[ \text{2ORZA15} \]
We want to optimize AIGs by re-arranging AND cones.
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
BUT - a circuit see more than one transition during it’s lifetime

input stream = sequence of values as they are applied to the circuit inputs

we need a ‘typical sequence’

\[
\text{Input stream} + \text{Internal structure} = \text{Actual switching}
\]
A (long) input stream can be derived from simulation of the design.

Such streams are commonly used for functional verification and quantitative evaluation of the circuit.

If we have a probabilistic model for the input, we can use it to generate an input stream.
Optimization and evaluation flow

Input stream source → Learning Stream → Circuit AIG

Evaluation Stream → Evaluation → Optimized AIG
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 learning sequence.

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

1. Enumerative
   - Growing too fast
   - Realistic only for small cones (up to approximately 8 inputs).

2. Layer based approximation
   - Optimal on "layers"
   - Globally suboptimal
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 $[O(n^3), Edmonds65]$. 
Evaluation scenarios

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. A model of a simple instruction decoder
Variables depend just on the previous value
Cascades - variables ordered, depending on the previous one or two
Partitioned variables - variables forms internally dependent clusters
Arbitrary sparse network of dependencies
Synthetic models results

Independent Cascades

Partitioned Sparse
Buttons are not pressed randomly

Some sequences don’t make sense

Some operations are used more often (that’s why plus is bigger)

We build a Markov chain describing which button is going to get pressed based on reasonable assumptions
Mini instruction decoder

\[
\begin{align*}
&\text{start} \\
&\quad \quad \quad \quad p_{lm}: \text{LOADM} \quad 1-p_{sm}: \text{EVAL} \\
&\quad \quad \quad \quad p_{lm}: \text{LOAD} \quad 1-p_{lm}: \text{LOAD} \\
&\quad \quad \quad \quad p_{add}: \text{SET}_\text{ADD} \\
&\quad \quad \quad \quad p_{sub}: \text{SET}_\text{SUB} \\
&\quad \quad \quad \quad p_{mul}: \text{SET}_\text{MUL} \\
&\quad \quad \quad \quad p_{div}: \text{SET}_\text{DIV} \\
&\quad \quad \quad \quad p_{lm:} \text{LOADM} \quad 1-p_{lm:} \text{LOAD} \\
&\quad \quad \quad \quad p_{sm}: \text{EVAL} \\
&\quad \quad \quad \quad \text{STORE} \\
&\quad \quad \quad \quad \text{store} \\
&\quad \quad \quad \quad \text{set op} \\
&\quad \quad \quad \quad \text{op set} \\
&\quad \quad \quad \quad \text{loaded} \\
\end{align*}
\]

\[
\begin{align*}
p_{lm} &= 0.1 & p_{add} &= 0.4 & p_{sub} &= 0.3 \\
p_{mul} &= 0.2 & p_{div} &= 0.1 & p_{sm} &= 0.1
\end{align*}
\]
A comparison of the number of switchings in the optimized instruction compared to 20 other arbitrary realizations. The height of bars shows how much switching can be saved using the optimized circuit compared to that realization.
Conclusion

Main contributions
- Level-optimal AIG switching optimization method
- Evaluation on synthetic and toy hardware model
- Efficiency related to input randomness

Issues
- Non-optimality of level-based - seems to be only theoretical
- Small AND cones in many examples
- Other steps further down may kill the savings - we are working on a tighter integration in the ABC synthesis tool [A. Mischenko]

Thank you!