Fast Synthesis of Threshold Logic Networks with Optimization

Yung-Chih Chen*, Runyi Wang, and Yan-Ping Chang
Yuan Ze University, Taiwan
Outline

- Introduction
- Background
- Threshold logic synthesis and optimization
- Experimental results
- Conclusion
Threshold logic

- **Threshold logic** is an alternative representation to conventional Boolean logic.

- Logical function $f$ of a threshold logic gate is defined as follows:

$$f(x_1, x_2, \ldots, x_n) = \begin{cases} 1, & \text{if } \sum_{i=1}^{n} x_i w_i \geq T \\ 0, & \text{otherwise} \end{cases}$$

- **Threshold logic network (TLN)**
  - A logic network composed of threshold gates.
Threshold logic

- Development of threshold logic
  - Started in 1960s, but had only a little impact on today’s IC designs
    - lack of effective hardware implementation
  - Re-attracted attention in recent years
    - advances in nanoscale device technology
      - Resonant tunneling diodes, quantum cellular automata, and single-electron transistor
        - They are possible devices for threshold logic implementation
  - Design automation techniques
    - synthesis, optimization, verification, static timing analysis, and automatic test pattern generation
Advantages of threshold logic (1/2)

- Compatible with nanoscale devices
- Compact

\[ f = x_1(x_2 + x_3 + (x_4(x_5 + x_6))) \]

- Could be a good intermediate representation in today’s design flow
  - Used to enhance logic optimization and design verification
Advantages of threshold logic logic (2/2)

Logic optimization

- Boolean logic network
- TLN
- TLN optimization (different opt. opportunities)
- Opt. TLN
- Opt. Boolean logic network

Equivalence checking

- Boolean logic networks
- TLNs
- TLN equivalence checking (lower complexity)

Fast network transformation
We aim to propose a FAST TLN synthesis approach

Problem formulation
- Input: a conventional Boolean logic
- Output: a TLN with minimized gate count
Previous works on TLN synthesis

- Work based on a **threshold function**\(^1\) identification procedure
  - Integer linear programming (ILP)-based
  - Binary decision diagram or truth table-based

- For a Boolean function
  - A threshold function $\rightarrow$ **weights** and **threshold value**
  - Not a threshold function $\rightarrow$ function **decomposition**

- For a Boolean logic network to be synthesized
  - They repeatedly identify and map all the sub-functions into threshold logic gates

- **Main disadvantage**
  - Inefficiency

---

\(^1\) **Threshold function**: a Boolean function which can be implemented with only one threshold logic gate
Our approach

- A fast synthesis approach without threshold function identification
  - Faster
  - Better or competitive synthesis quality
Outline

- Introduction
- Background
- Threshold logic synthesis and optimization
- Experimental results
- Conclusion
A threshold function

- A Boolean function which can be implemented with only one threshold logic gate

Conventional primitive functions, such as AND, OR, and NOT, are threshold functions

\[ f(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} w_i x_i \geq \theta \\ 0 & \text{otherwise} \end{cases} \]

- n-input AND
- n-input OR
- Inverter
Thus, a Boolean logic network composed of only primitive logic gates can be **FAST** transformed into a TLN by one-to-one mapping.

- Each logic gate is mapped to a threshold gate.

Actually, this is the first step of our approach.
Positive weight transformation

- In a threshold gate, a weight could be a positive or negative number.
- For easy to manipulate a threshold gate, the negative weights can be transformed into positive weights.
- We also perform this transformation to avoid negative weights during synthesis process.

![Diagram showing positive weight transformation](image-url)
Controlling-1 and -0 inputs

- **Controlling-1 input** of a threshold gate $g$
  - An input which can determine the output value of $g$ to 1 regardless of the other inputs

- **Controlling-0 input** of a threshold gate $g$
  - An input which can determine the output value of $g$ to 0 regardless of the other inputs

**Diagram 1**

- $x_1 = 1$ implies $f = 1$
- $x_1 = 0$ implies $f = 0$

**Diagram 2**

- $w_i \geq T$
- $\sum_{i=1}^{n} w_i - w_i < T$
Outline

- Introduction
- Background
- Threshold logic synthesis and optimization
- Experimental results
- Conclusion
Flowchart of the proposed method

1. Boolean logic network
2. Network transformation by one-to-one mapping
3. TLN
4. TLN optimization
5. Opt. TLN

- Inverter elimination with reverse positive weight transformation
- Optimization with predefined transformations
- One-to-one mapping
TLN optimization with predefined transformations

- Eight transformations for threshold logic
  - Sufficient conditions for eliminating a gate or merging two adjacent gates (i.e., one gate and one of its fanin gates)
  - Work only for threshold gates with only positive weights

- Simple example of merging two adjacent gates
Transformations 1 & 2

- **T1**: Constant gate elimination
  \[ \sum_{i=1}^{n} w_i < T \]
  \[ T \leq 0 \]

- **T2**: Adjacent AND or OR gate merging
T3: AND gate-based merging (adapted from [9])

An AND gate can be merged with one of its fanin gates.

\[ w_r = \sum_{i=1}^{m} w_{f_i} - T_f + 1 \]
\[ T_r = (n - 1)w_r + T_f \]

Transformation 4

- T4: OR gate-based merging (adapted from [18])
  - An OR gate can be merged with one of its fanin gates

Transformations 5 & 6

- **T5:** Sum-of-product form to product-of-sum form conversion

  
  
  \[ x_1 x_2 + x_2 x_3 \]

  
  \[ x_2 (x_1 + x_3) \]

  T3: AND gate-based merging

- **T6:** Product-of-sum form to sum-of-product form conversion

  
  
  \[ (x_1 + x_2)(x_1 + x_3) \]

  
  \[ x_2 + x_1 x_2 \]

  T4: gate-based merging
Transformation 7

- T7: Controlling-1 input-based merging
  - $g_f$ is a controlling-1 input of $g$ and
  - $g_f$ is an OR gate
Transformation 8

- **T8: Controlling-0 input-based merging**
  - $g_f$ is a controlling-0 input of $g$ and
  - $g_f$ is an AND gate

\[
\sum_{i=1}^{n-1} w_i < T_g
\]

\[
T_r = T_g + (m-1) * w_f
\]
Overall flow of TLN optimization

- The are three iterations and each iteration targets certain types of transformations
  - First iteration
    - T2
  - Second iteration
    - T5 and T6
  - Third iteration
    - T3, T4, T7, T8, and T1

- At each iteration, each gate is selected as a target gate one at a time in the topological order, and we check and perform the transformation under consideration to the target gate if applicable
Outline

- Introduction
- Background
- Threshold logic synthesis and optimization
- Experimental results
- Conclusion
Experimental setup

- C language within the ABC [2] environment
- Linux platform with two 1.90GHz CPUs and 32GB memory
- Benchmarks
  - IWLS 2005 benchmark suite
  - And-Inverter Graph format
- Comparison
  - ILP-based method [18] + lp_solve


Experimental results

- For fair comparison
  - Fanin count constraint
    - 6
- Save an average of 28% threshold gates
- Much more efficient

<table>
<thead>
<tr>
<th>benchmark</th>
<th>/N/</th>
<th>ILP-based method</th>
<th>Our method</th>
<th>ratio</th>
<th>T(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>N</td>
<td>T(s)</td>
<td></td>
</tr>
<tr>
<td>pci_conf.</td>
<td>84</td>
<td>91</td>
<td>2.2</td>
<td>62</td>
<td>0.68</td>
</tr>
<tr>
<td>stepper.</td>
<td>157</td>
<td>124</td>
<td>3.1</td>
<td>83</td>
<td>0.67</td>
</tr>
<tr>
<td>ss_pcm</td>
<td>172</td>
<td>173</td>
<td>4.4</td>
<td>135</td>
<td>0.78</td>
</tr>
<tr>
<td>usb_phy</td>
<td>357</td>
<td>287</td>
<td>7.2</td>
<td>221</td>
<td>0.77</td>
</tr>
<tr>
<td>sasc</td>
<td>563</td>
<td>461</td>
<td>12.5</td>
<td>333</td>
<td>0.72</td>
</tr>
<tr>
<td>simple_spi</td>
<td>775</td>
<td>597</td>
<td>16.1</td>
<td>436</td>
<td>0.73</td>
</tr>
<tr>
<td>pci_spoci.</td>
<td>878</td>
<td>559</td>
<td>15.6</td>
<td>399</td>
<td>0.71</td>
</tr>
<tr>
<td>i2c</td>
<td>941</td>
<td>659</td>
<td>18.1</td>
<td>482</td>
<td>0.73</td>
</tr>
<tr>
<td>systemcdes</td>
<td>2641</td>
<td>2018</td>
<td>57.7</td>
<td>1377</td>
<td>0.68</td>
</tr>
<tr>
<td>spi</td>
<td>3429</td>
<td>2421</td>
<td>75.6</td>
<td>1614</td>
<td>0.67</td>
</tr>
<tr>
<td>des_area</td>
<td>4410</td>
<td>2774</td>
<td>94.4</td>
<td>2011</td>
<td>0.72</td>
</tr>
<tr>
<td>tv80</td>
<td>7233</td>
<td>4996</td>
<td>191.1</td>
<td>3559</td>
<td>0.71</td>
</tr>
<tr>
<td>mem_ctrl</td>
<td>8815</td>
<td>6573</td>
<td>267.6</td>
<td>4721</td>
<td>0.72</td>
</tr>
<tr>
<td>systemcaes</td>
<td>10585</td>
<td>7677</td>
<td>334.4</td>
<td>5333</td>
<td>0.69</td>
</tr>
<tr>
<td>ac97_ctrl</td>
<td>10395</td>
<td>8326</td>
<td>330.0</td>
<td>6194</td>
<td>0.74</td>
</tr>
<tr>
<td>usb_funct</td>
<td>13320</td>
<td>9860</td>
<td>468.6</td>
<td>6842</td>
<td>0.69</td>
</tr>
<tr>
<td>pci_bridge32</td>
<td>17814</td>
<td>13595</td>
<td>769.9</td>
<td>10496</td>
<td>0.77</td>
</tr>
<tr>
<td>aes_core</td>
<td>20509</td>
<td>14163</td>
<td>761.2</td>
<td>10057</td>
<td>0.71</td>
</tr>
<tr>
<td>wb_conmax</td>
<td>41070</td>
<td>28518</td>
<td>2148.3</td>
<td>21956</td>
<td>0.77</td>
</tr>
<tr>
<td>ethernet</td>
<td>57205</td>
<td>47004</td>
<td>4978.9</td>
<td>35243</td>
<td>0.75</td>
</tr>
<tr>
<td>des_perf</td>
<td>71327</td>
<td>59886</td>
<td>7210.0</td>
<td>42719</td>
<td>0.71</td>
</tr>
<tr>
<td>vga_lcd</td>
<td>88854</td>
<td>74095</td>
<td>10918.6</td>
<td>55402</td>
<td>0.75</td>
</tr>
<tr>
<td>average</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.72</td>
</tr>
<tr>
<td>total</td>
<td></td>
<td></td>
<td></td>
<td>28685.6</td>
<td>3.4</td>
</tr>
</tbody>
</table>
Outline

- Introduction
- Background
- Threshold logic synthesis and optimization
- Experimental results
- Conclusion
Conclusion

- We proposed a simple and fast approach for TLN synthesis and optimization
  - Much more efficient and effective than an ILP-based method

- Future work
  - Apply this compact logic representation to enhance conventional logic optimization and design verification
Thank you for attention