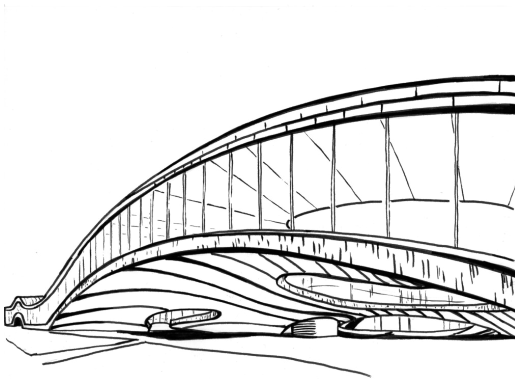
SAT in Logic Synthesis

Mathias Soeken

Integrated Systems Laboratory, EPFL, Switzerland
mathias.soeken@epfl.ch msoeken.github.io msoeken/cirkit

EPFL



Background

SAT-based area recovery in structural technology mapping

Applying logic synthesis to speedup SAT

SAT-based exact synthesis: encodings, topology families, and parallelism

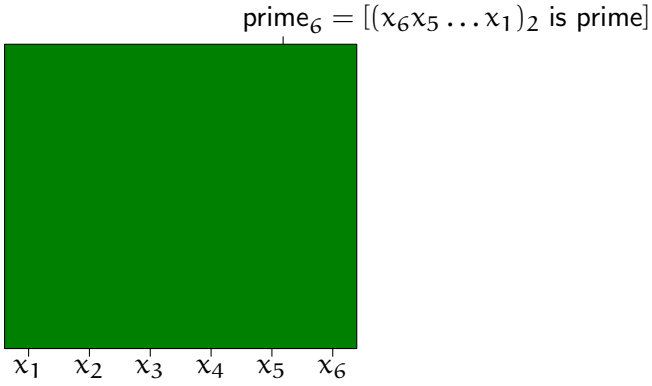
Background

SAT-based area recovery in structural technology mapping

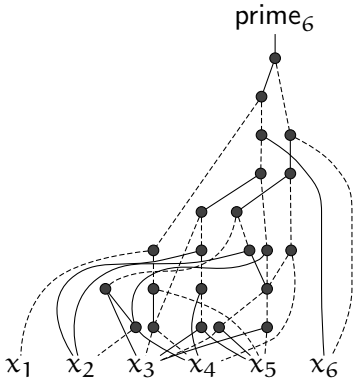
Applying logic synthesis to speedup SAT

SAT-based exact synthesis: encodings, topology families, and parallelism

Decompose large functions using k-LUT mapping

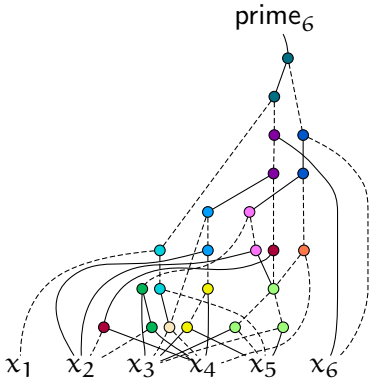


Decompose large functions using k-LUT mapping



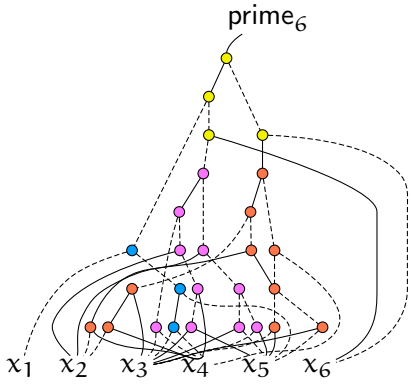
1. Represent function as simple logic network with small gate primitives (here: And-inverter graph)

Decompose large functions using k-LUT mapping



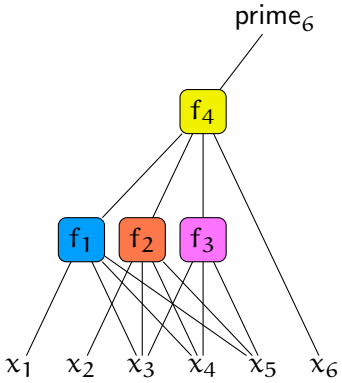
1. Represent function as simple logic network with small gate primitives (here: And-inverter graph)
2. Cover network with subnetworks with limited fanin size k (here $k = 3$ and $k = 4$)

Decompose large functions using k-LUT mapping



1. Represent function as simple logic network with small gate primitives (here: And-inverter graph)
2. Cover network with subnetworks with limited fanin size k (here $k = 3$ and $k = 4$)

Decompose large functions using k-LUT mapping



1. Represent function as simple logic network with small gate primitives (here: And-inverter graph)
2. Cover network with subnetworks with limited fanin size k (here $k = 3$ and $k = 4$)
3. Collapse covered subnetworks into lookup-table nodes

Background

SAT-based area recovery in structural technology mapping

Applying logic synthesis to speedup SAT

SAT-based exact synthesis: encodings, topology families, and parallelism

SAT-based LUT mapping

- Input: Logic network, set of k-cuts for each gate

SAT-based LUT mapping

- Input: Logic network, set of k-cuts for each gate
- Output: k-LUT mapping

SAT-based LUT mapping

- Input: Logic network, set of k-cuts for each gate
- Output: k-LUT mapping

- Solve problem *"Does there exist a* k*-LUT mapping with* L *cuts?"* as a SAT problem.

SAT-based LUT mapping

- Input: Logic network, set of k-cuts for each gate
- Output: k-LUT mapping

- Solve problem *"Does there exist a* k*-LUT mapping with* L *cuts?"* as a SAT problem.
- Solve problem starting from some satisfiable upper bound, and improve bound until no more solution can be found

SAT-based LUT mapping

- Input: Logic network, set of k-cuts for each gate
- Output: k-LUT mapping

- Solve problem *"Does there exist a* k*-LUT mapping with* L *cuts?"* as a SAT problem.
- Solve problem starting from some satisfiable upper bound, and improve bound until no more solution can be found
- Result guaranteed to be optimum for selected cuts

[B. Schmitt, A. Mishchenko, and R.K. Brayton, *ASP-DAC* 23 (2018), 586–591]

SAT-based LUT mapping

- Input: Logic network, set of k-cuts for each gate
- Output: k-LUT mapping

- Solve problem *"Does there exist a* k*-LUT mapping with* L *cuts?"* as a SAT problem.
- Solve problem starting from some satisfiable upper bound, and improve bound until no more solution can be found
- Result guaranteed to be optimum for selected cuts

- How to encode the problem?

[B. Schmitt, A. Mishchenko, and R.K. Brayton, *ASP-DAC* 23 (2018), 586–591]

SAT-based LUT mapping

- Input: Logic network, set of k-cuts for each gate
- Output: k-LUT mapping

- Solve problem *"Does there exist a* k*-LUT mapping with* L *cuts?"* as a SAT problem.
- Solve problem starting from some satisfiable upper bound, and improve bound until no more solution can be found
- Result guaranteed to be optimum for selected cuts

- How to encode the problem?
- How to make it reliably efficient for large networks?

Encoding

Variables

- $m_i = [\text{gate } i \text{ is mapped}]$
  for each gate i

Encoding

Variables

- $\mathrm{m_i} = [\text{gate i is mapped}]$
  for each gate i
- $\mathrm{s_{i,C}} = [\text{cut C is selected for gate i}]$
  for each cut C *of gate* i

Encoding

Variables

- $\mathrm{m_i} = [\text{gate i is mapped}]$
  for each gate i
- $\mathrm{s_{i,C}} = [\text{cut C is selected for gate i}]$
  for each cut C *of gate* i

Clauses

- $(\mathrm{m_o})$ for each gate o that drives an output

Encoding

Variables

- $m_i = [\text{gate } i \text{ is mapped}]$
  for each gate i
- $s_{i,C} = [\text{cut } C \text{ is selected for gate } i]$
  for each cut C *of gate* i

Clauses

- (m_o) for each gate o that drives an output
- $m_i \rightarrow \bigvee_{C \in \text{CUTS}(i)} s_{i,C}$ for each gate i

Encoding

Variables

- $m_i = [\text{gate } i \text{ is mapped}]$
  for each gate i
- $s_{i,C} = [\text{cut } C \text{ is selected for gate } i]$
  for each cut C *of gate* i

Clauses

- (m_o) for each gate o that drives an output
- $m_i \rightarrow \bigvee_{C \in \text{CUTS}(i)} s_{i,C}$ for each gate i
- $s_{i,C} \rightarrow \bigwedge_{j \in C} m_j$ for each cut C of gate i

Encoding

Variables

- $m_i = [\text{gate } i \text{ is mapped}]$
  for each gate i
- $s_{i,C} = [\text{cut } C \text{ is selected for gate } i]$
  for each cut C *of gate* i

Clauses

- (m_o) for each gate o that drives an output
- $m_i \rightarrow \bigvee_{C \in \text{CUTS}(i)} s_{i,C}$ for each gate i
- $s_{i,C} \rightarrow \bigwedge_{j \in C} m_j$ for each cut C of gate i
- $\sum m_i \leqslant L$

Windowing

- Runtime degrades quickly as networks become larger

Windowing

- Runtime degrades quickly as networks become larger
- Restrict network size using windowing (to, e.g., 128 gates)

[B. Schmitt, A. Mishchenko, and R.K. Brayton, *ASP-DAC* **23** (2018), 586–591]

Windowing

- Runtime degrades quickly as networks become larger
- Restrict network size using windowing (to, e.g., 128 gates)
- Windowing is applied to mapped network, not cutting through mapped cuts

Windowing

- Runtime degrades quickly as networks become larger
- Restrict network size using windowing (to, e.g., 128 gates)
- Windowing is applied to mapped network, not cutting through mapped cuts
- Find good pivots to extract windows

Windowing

- Runtime degrades quickly as networks become larger
- Restrict network size using windowing (to, e.g., 128 gates)
- Windowing is applied to mapped network, not cutting through mapped cuts
- Find good pivots to extract windows
- Cache windows to avoid duplicate optimization effort

[B. Schmitt, A. Mishchenko, and R.K. Brayton, *ASP-DAC* 23 (2018), 586–591]

Implementation

ABC

- Command `&satlut`

Implementation

ABC

- Command `&satlut`
- Input must be 6-LUT mapped AIG

Implementation

ABC

- Command `&satlut`
- Input must be 6-LUT mapped AIG

mockturtle

- C++ function `satlut_mapping<Ntk>`

Implementation

ABC

- Command `&satlut`
- Input must be 6-LUT mapped AIG

mockturtle

- C++ function `satlut_mapping<Ntk>`
- Works on arbitrary logic networks

Implementation

ABC

- Command `&satlut`
- Input must be 6-LUT mapped AIG

mockturtle

- C++ function `satlut_mapping<Ntk>`
- Works on arbitrary logic networks
- Works on arbitrary cut sizes

Implementation

ABC

- Command `&satlut`
- Input must be 6-LUT mapped AIG

mockturtle

- C++ function `satlut_mapping<Ntk>`
- Works on arbitrary logic networks
- Works on arbitrary cut sizes
- Not fully finished (PR #122)

Background

SAT-based area recovery in structural technology mapping

Applying logic synthesis to speedup SAT

SAT-based exact synthesis: encodings, topology families, and parallelism

CNF generation for logic networks

- SAT-based verification and synthesis tasks require logic network to be represented as CNF
  e.g., equivalence checking, model checking, Boolean resubstitution

CNF generation for logic networks

- SAT-based verification and synthesis tasks require logic network to be represented as CNF
  e.g., equivalence checking, model checking, Boolean resubstitution

- Traditional approach is to use Tseytin's encoding

CNF generation for logic networks

- SAT-based verification and synthesis tasks require logic network to be represented as CNF
  e.g., equivalence checking, model checking, Boolean resubstitution

- Traditional approach is to use Tseytin's encoding
- Each gate is assigned an auxiliary variable

[N. Eén, A. Mishchenko, and N. Sörensson, *SAT* 10 (2007), 272–286]

CNF generation for logic networks

- SAT-based verification and synthesis tasks require logic network to be represented as CNF
  e.g., equivalence checking, model checking, Boolean resubstitution

- Traditional approach is to use Tseytin's encoding
- Each gate is assigned an auxiliary variable
- Gate function is transformed into CNF

[N. Eén, A. Mishchenko, and N. Sörensson, *SAT* 10 (2007), 272–286]

CNF generation for logic networks

- SAT-based verification and synthesis tasks require logic network to be represented as CNF
  e.g., equivalence checking, model checking, Boolean resubstitution

- Traditional approach is to use Tseytin's encoding
- Each gate is assigned an auxiliary variable
- Gate function is transformed into CNF
- Example $c = a \wedge b$:

$$(a \vee \bar{c})(b \vee \bar{c})(\bar{a} \vee \bar{b} \vee c)$$

[N. Eén, A. Mishchenko, and N. Sörensson, *SAT* 10 (2007), 272–286]

CNF generation for logic networks

- SAT-based verification and synthesis tasks require logic network to be represented as CNF
  e.g., equivalence checking, model checking, Boolean resubstitution

- Traditional approach is to use Tseytin's encoding
- Each gate is assigned an auxiliary variable
- Gate function is transformed into CNF
- Example $c = a \wedge b$:

$$(a \vee \bar{c})(b \vee \bar{c})(\bar{a} \vee \bar{b} \vee c)$$

- Advantage: Resulting CNF is linear in the number of gates

[N. Eén, A. Mishchenko, and N. Sörensson, *SAT* 10 (2007), 272–286]

CNF generation for logic networks

- SAT-based verification and synthesis tasks require logic network to be represented as CNF
  e.g., equivalence checking, model checking, Boolean resubstitution

- Traditional approach is to use Tseytin's encoding
- Each gate is assigned an auxiliary variable
- Gate function is transformed into CNF
- Example $c = a \wedge b$:

$$(a \vee \bar{c})(b \vee \bar{c})(\bar{a} \vee \bar{b} \vee c)$$

- Advantage: Resulting CNF is linear in the number of gates
- Disadvantage: Requires the use of many auxiliary variables

[N. Eén, A. Mishchenko, and N. Sörensson, *SAT* 10 (2007), 272–286]

LUT-based CNF generation

- Perform LUT mapping to control tradeoff between number of auxiliary variables and number of clauses

LUT-based CNF generation

- Perform LUT mapping to control tradeoff between number of auxiliary variables and number of clauses
- Idea: perform Tseytin encoding on LUTs, not on gates

[N. Eén, A. Mishchenko, and N. Sörensson, *SAT* 10 (2007), 272–286]

LUT-based CNF generation

- Perform LUT mapping to control tradeoff between number of auxiliary variables and number of clauses
- Idea: perform Tseytin encoding on LUTs, not on gates
- Advantage: Number of auxiliary variables corresponds to number of *mapped* gates

[N. Eén, A. Mishchenko, and N. Sörensson, *SAT* 10 (2007), 272–286]

LUT-based CNF generation

- Perform LUT mapping to control tradeoff between number of auxiliary variables and number of clauses
- Idea: perform Tseytin encoding on LUTs, not on gates
- Advantage: Number of auxiliary variables corresponds to number of *mapped* gates
- Cost function based on CNF size

[N. Eén, A. Mishchenko, and N. Sörensson, *SAT* 10 (2007), 272–286]

LUT-based CNF generation

- Perform LUT mapping to control tradeoff between number of auxiliary variables and number of clauses
- Idea: perform Tseytin encoding on LUTs, not on gates
- Advantage: Number of auxiliary variables corresponds to number of *mapped* gates
- Cost function based on CNF size
- May even lead to fewer number of overall clauses

[N. Eén, A. Mishchenko, and N. Sörensson, *SAT* 10 (2007), 272–286]

Background

SAT-based area recovery in structural technology mapping

Applying logic synthesis to speedup SAT

SAT-based exact synthesis: encodings, topology families, and parallelism

[W. Haaswijk, M. Soeken, A. Mishchenko, and G. De Micheli, SAT-Based Exact Synthesis: Encodings, Topology Families, and Parallelism *TCAD*, to appear]

SAT in Logic Synthesis

Mathias Soeken

Integrated Systems Laboratory, EPFL, Switzerland
mathias.soeken@epfl.ch msoeken.github.io msoeken/cirkit

EPFL