SIS
Logic Synthesis System

Michele Lora
Tiziano Villa

University of Verona
Dep. Computer Science
Italy

Slides courtesy of Giuseppe and Luigi Di Guglielmo, Davide Bresolin, Michele Lora and Tiziano Villa
Logic Synthesis performs the translation from a RTL description (e.g., VHDL) into a design implementation in terms of logic gates, and optimizes the latter.

- It may be driven by different cost functions:
  - Area
  - Delay, clock speed
  - etc.

- It leads to implementations meeting the desired objectives.
SIS – Logic Synthesis System (I)

• **SIS** is an interactive tool for synthesis and optimization of sequential circuits
  – developed by the CAD group of U.C. Berkeley in the 1990s

• It produces an optimized net-list preserving the sequential input/output behavior

• It incorporates a set of logic optimization algorithms
  – users can choose among a variety of techniques at each stage of the synthesis process
• Different algorithms for various stages of sequential synthesis:
  – State minimization
  – State assignment
  – Node simplification
  – Kernel and cube extraction
  – Technology mapping
  – Retiming
How to use SIS

1. Design specification
2. Rewrite in different formats / Read statistics
3. Technology independent optimization
4. Technology mapping
5. Gate-level optimization

Formats:
1. pla
2. blif
3. kiss
4. eqn

1. FSM optimization
2. FSM to logic and reverse
3. Combinational optimization
4. Sequential (gate-level) optimization

Target library
1. Retiming
Design Specification

- A sequential circuit can be input to *SIS* in several ways allowing *SIS* to be used at various stages of the design process
- The most common entry points are
  - Net-list of gates
  - Truth table in PLA (espresso) format
  - Set of equations
  - Finite-state machine in state-transition-table form
Logic Implementation (Net-list)

• The net-list description is given in extended *BLIF* (Berkeley Logic Interchange Format)

• It consists of interconnected single-output combinational gates and latches
  – Gates are simple elements that perform logical operations
  – Latches store the state
The SIS Synthesis and Optimization System – Read the specifications

• $> \text{sis}$
  UC Berkeley, SIS 1.3.6 (compiled 2010-11-14 12:35:42)
  sis>
• sis> \texttt{help}
  – returns a list of all the commands provided by SIS
• sis> \texttt{read_pla <file_name>}
  – Loads a pla description
• sis> \texttt{read_blif <file_name>}
  – Loads a net-list description
• sis> \texttt{read_eqn <file_name>}
  – Loads an equation-based description
• sis> \texttt{read_kiss <file_name>}
  – Loads a kiss-style STG description
The SIS Synthesis and Optimization System – Write the specifications

- **sis> write_pla [file_name]**
  - Writes a PLA description
- **sis> write_blif [file_name]**
  - Writes a net-list description
- **sis> write_eqn [file_name]**
  - Writes an equation-based description
- **sis> write_kiss [file_name]**
  - Writes a kiss-style STG description
Example 1: the full-adder

# full-adder circuit
.i 3
.o 2
.ilb x y z
.ob C S
000 00
001 01
010 01
011 10
100 01
101 10
110 10
111 11
.end

• Open the file in SIS
• Try the following commands:
  – print
  – print_stats
  – print_io

• Create a .pla file with the description
The SIS Synthesis and Optimization System – Node simplification (II)

• In SIS, the user may invoke the ESPRESSO program to perform node simplification
• sis> full_simplify
  – the input is a net-list (blif or PLA format)
  – returns a minimized net-list (blif format)
  • SIS decomposes multiple output functions into single output functions and represents them separately
A logical network can be modified by
- Creating new nodes
- Deleting nodes
- Creating new connections
- Deleting connections

A particular case of node restructuring is node creation by extracting a factor from one or more nodes
The SIS Synthesis and Optimization System – Kernel (I)

• The extraction of new nodes that are factors of existing nodes is a form of division that may be performed in the Boolean or algebraic domain

• Algebraic techniques: sum-of-products are treated as standard polynomials
  – look for expressions that are observed many times in the nodes of the network and extract such common expressions
  – The extracted expression is implemented only once and the output of that node replaces the expression in any other node

• Current algebraic techniques used in SIS are based on cube-free divisors called kernels
The SIS Synthesis and Optimization System – Kernel (II)

• An expression $f$ is cube-free if no cubes divides the expression evenly
  – $ab + c$ is cube free
  – $ab+ac$ or $abc$ are not cube free

• The primary divisors of an expression are obtained by dividing the expression by cubes

• The kernels of an expression are the cube-free primary divisors of the expression
The SIS Synthesis and Optimization System – Kernel (III)

\[ adf + aef + bdf + bef + cdf + cef + g = (a+b+c)(d+e)f + g \]

<table>
<thead>
<tr>
<th>Kernel</th>
<th>Cokernel</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a+b+c )</td>
<td>( df, \ ef )</td>
</tr>
<tr>
<td>( d+e )</td>
<td>( af, \ bf, \ cf )</td>
</tr>
<tr>
<td>( (a+b+c)(d+e) )</td>
<td>( f )</td>
</tr>
<tr>
<td>( (a+b+c)(d+e)f + g )</td>
<td>( 1 )</td>
</tr>
</tbody>
</table>
In SIS, the user may invoke the command `fast_extract`, i.e., `fx`, to perform kerneling.

- `sis > fx`
  - the input is a net-list
  - extracts common expressions among the nodes and rewrites the nodes of the network in terms of common expressions
The SIS Synthesis and Optimization System – Technology mapping (I)

• A tree-covering algorithm is used to map arbitrary complex logic gates into cells available in a technology library

• Technology mapping consists of two phases:
  – Decomposing the logic to be mapped into a network of 2-input NAND gates and inverters
  – Covering the network by patterns that represent the possible cells in the library

• During the covering stage the area or the delay of the circuit is used as an optimization criterion
The SIS Synthesis and Optimization System – Technology mapping (II)

• In SIS, the user may invoke the command *rlib* and *map* to select a library for the technology mapping and perform the mapping, respectively

• **sis> rlib <library_name>**
  
  **sis> map**

  – the input is a net-list (blif format)
  
  – map complex logic gates into cells of the chosen technology library (genlib format)
Mapping the full adder

• Load the *minimal.genlib* library
• Map the full-adder to the library
• Try the following commands:
  – print
  – print_gate
  – print_map_stats
Example 2: the ticket machine

- Print a ticket after 1.50 euros are deposited
- Single slot for 1 euro and 50 cents coins
- No change
State Transition Graph (STG) (I)

• A state transition table for a finite-state machine is specified using the KISS format
• It is used in state assignment and state minimization programs
• STG
  – States are symbolic
  – The transition table indicates the next symbolic state and output bit-vector given a current state and an input bit-vector
  – Don’t care conditions are indicated by a missing transition or by a ‘-’ in an output bit
    • A present-state/input combination has no explicit next-state/output, or
    • For a present-state/input combination a ‘-’ output stands for either 0 or 1
State Transition Graph (STG) (II)

```
.start_kiss
.i 2
.o 1
.r q0
00 q0 q0 0
10 q0 q1 0
01 q0 q2 0
00 q1 q1 0
10 q1 q3 0
01 q1 q0 1
00 q2 q2 0
10 q2 q0 1
01 q2 q0 1
00 q3 q3 0
10 q3 q0 1
01 q3 q0 1
.end_kiss
.end
```
The SIS Synthesis and Optimization System – State minimization (I)

• State minimization works on STGs
  – Degrees of freedom (i.e., unspecified transitions or explicit output don’t cares) can be exploited to produce a machine with fewer states

• State minimization looks for equivalent states in order to minimize the total number of states
  – Two states are equivalent if they produce the same output sequences given the same input sequences
The SIS Synthesis and Optimization System – State minimization (II)

• In SIS, the user may invoke the STAMINA program to perform state minimization
  – STAMINA is a state minimizer for incompletely specified machine

• sis> state_minimize stamina
  – the input is a STG (kiss format)
  – the original STG is replaced by the one computed by STAMINA with (possibly) fewer states
Minimize the Ticket counter FSM

• Create a .kiss2 file with the description of the FSM
• Open the file in SIS
• Minimize the FSM
• Compare the minimized machine with the original one
  – use write_kiss to print the minimized FSM
The SIS Synthesis and Optimization System – State assignment (I)

• State assignment provides the mapping from a STG to a net-list

• State assignment requires a state transition table and computes binary codes for each symbolic state

• Binary codes are used to create a logic level implementation
  – substituting the binary codes for the symbolic states, it creates a latch for each bit of the binary code
The SIS Synthesis and Optimization System – State assignment (II)

• In SIS, the user may invoke either JEDI or NOVA programs to perform state assignments

• sis> state_assign nova

or

sis> state_assign jedi

– the input is a STG (kiss format)

– returns a state assignment of the STG and a corresponding logic implementation (net-list) (blif format)
State-assign the Ticket counter FSM

- Do the state assignment of the minimized FSM
  - Use `print`, `print_stats`, and `print_latch` to obtain information

- Simulate the FSM
  - Use `simulate input1 input2` to give inputs
  - `print_state` gives the current state of the FSM

- Compare the results of `jedi` and `nova`
Mapping the ticket counter

• Load the *lib2.genlib* and the *lib2_latch.genlib* libraries
  – Use `rlib -a` to load the second library!
• Map the ticket counter FSM
• Try the following commands:
  – `print_stats`
  – `print_gate`
  – `print_map_stats`
The SIS Synthesis and Optimization System – Retiming (I)

• Retiming is an algorithm that moves registers across logic gates to minimize
  – Cycle time, or
  – Number of registers, or
  – Number of registers subject to a cycle-time constraint

• It operates on synchronous edge-triggered designs

• The sequential I/O behavior of the circuit is maintained
The SIS Synthesis and Optimization System – Retiming (II)

• In SIS, the user may invoke the command `retime` to perform the retiming of the circuit

• `sis> retime`
  – the input is a net-list (blif format)
  – Add more latches, or re-position the latches, to reduce the clock period

  • Generally used to reduce the cycle time of the circuit by adding latches
Retiming the ticket counter

- Use the command `retime` to optimize the circuit
- Try the following commands:
  - `print_stats`
  - `print_gate`
  - `print_map_stats`
- Simulate the new circuit
Synthesis and optimizations

• SIS provides scripts for performing logic network optimizations
  – The standard *script*
  – The standard *script.rugged*
  – The standard *script.delay*

• Such scripts derive from the experience of SIS developers
References – U.C. Berkeley

• More SIS infos are available at http://embedded.eecs.berkeley.edu/pubs/downloads/sis/index.htm
  – Documentation
  – Examples