**SIS – Logic Synthesis System**

*Dott. Luigi Di Guglielmo*

*Prof. Tiziano Villa*

*University of Verona*

*Dep. Computer Science*

*Italy*
Agenda

• Introduction
• SIS – Logic Synthesis System
• The SIS Synthesis and Optimization System
• Scripting in SIS
• Examples
Introduction

• *Logic Synthesis* performs the translation from a high level description (e.g., VHDL) to a RTL description 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
SIS – Logic Synthesis System (II)

• Different algorithms for various stages of sequential synthesis:
  – State minimization
  – State assignment
  – Node simplification
  – Kernel and cube extraction
  – Technology mapping
  – Retiming
  – Retiming and Resynthesis
How to use SIS

Design specification

read

Technology independent optimization

Rewrite in different formats / Read statistics

Technology mapping

Gate-level optimization

Formats:
1. blif
2. kiss

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

Target library

1. Retiming
2. (Resynthesis)
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 two most common entry points are
  – Net-list of gates
  – Finite-state machine in state-transition-table form
Logic Implementation (Net-list)

• The net-list description is given in extended \textit{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
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)

```
.i 2
.o 2
.p 8
.s 5
.r st0
 01 st0 st1 00
 10 st0 st2 00
 01 st1 st2 00
 10 st1 st3 10
 01 st2 st3 10
 10 st2 st4 11
 11 st3 st0 00
 11 st4 st0 00
.e
```

```
11/00
01/00
10/00
10/11
11/00
10/10
01/10
01/10
```

Design Automation of Embedded Systems
The SIS Synthesis and Optimization System – Read the specifications (I)

- `$> sis$
  
  UC Berkeley, SIS 1.3.6 (compiled 2010-11-14 12:35:42)

- `sis>`

- `sis> help`
  
  – returns a list of all the commands provided by SIS

- `sis> read_blif <file_name>`
  
  – Loads a net-list description

- `sis> read_kiss <file_name>`
  
  – Loads a kiss-style STG description
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
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`
  - the input is a STG (kiss format)
  - returns a state assignment of the STG and a corresponding logic implementation (net-list) (blif format)

or

- `sis> state_assign jedi`
The SIS Synthesis and Optimization System – Node simplification (I)

- Node simplification performs two-level minimization of the Boolean function implemented at a given node.
- It relies on the DC-set due to limited controllability and observability of the node (satisfiability DCs and observability DCs).
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
The SIS Synthesis and Optimization System – Node Restructuring

• 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)

- \( \text{adf} + \text{aef} + \text{bdf} + \text{bef} + \text{cdf} + \text{cef} + g = (a+b+c)(d+e)f+g \)

<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>
The SIS Synthesis and Optimization System – Kernel (IV)

• In SIS, the user may invoke the command *fast_extract*, i.e., *fx*, to perform kerneling.
• *sis> fx*
  – the input is a net-list (blif format)
  – extracts common expressions among the nodes and rewrites the nodes of the network in terms of common expressions (blif format)
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)
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 reduces the clock period

  • Generally used to reduce the cycle time of the circuit by adding latches
The SIS Synthesis and Optimization System – Retime and Resynthesis (I)

- Retiming finds optimal register positions without altering the computational logic functions at each node.
- Combinational techniques work inside latches boundaries.
- Given a sequential circuit, these concepts can be combined:
  - Identify the largest subcircuits that can be retimed moving registers to boundaries of the subcircuits.
  - After retiming, standard combinational techniques are used to minimize the internal combinational logic.
  - Finally, the registers are retimed back into the circuit to minimize cycle time or the number of registers used.
The SIS Synthesis and Optimization System – Retime and Resynthesis (II)

- In SIS, retiming and resynthesis is not implemented yet
  - It is available in recent synthesis systems
USING THE SIS ENVIRONMENT
mark1: the running example

• Synchronous synthesis example
  – the example chosen is **mark1**
    • provided in the MCNC benchmark set
  – the specification is given in the *KISS* format
    • copy it from:
      ~ldg/lectures/daes20112012/lesson02/lesson-example
    • File: **mark1.kiss2**
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
The standard scripts

**script**

```plaintext
1 sweep; eliminate -1
2 simplify
3 eliminate -1
4
5 sweep; eliminate 5
6 simplify
7
8 resub -a
9
10 gkx -abt 30
11 resub -a; sweep
12 gcx -bt 30
13 resub -a; sweep
14
15 gkx -abt 10
16 resub -a; sweep
17 gcx -bt 10
18 resub -a; sweep
19
20 gkx -ab
21 resub -a; sweep
22 gcx -b
23 resub -a; sweep
24
25 eliminate 0
26 decomp -g *
27
28 eliminate -1; sweep
```

**script.rugged**

```plaintext
1 sweep; eliminate -1
2 simplify -m nocomp
3 eliminate -1
4
5 sweep; eliminate 5
6 simplify -m nocomp
7 resub -a
8
9 fx
10 resub -a; sweep
11
12 eliminate -1; sweep
13 full_simplify -m nocomp
14
15 decomp -q
16 tech_decomp -o 2
17 resub -a -d
18 sweep
19 reduce_depth -b -r
20 red_removal
21 eliminate -l 100 -l
22 simplify -l
23 full_simplify -l
24 sweep
25 decomp -q
26 fx -l
27 tech_decomp -o 2
28 rlib lib2.genlib
29 rlib -a lib2_latch.genlib
30 map -s -n 1 -AFG -p
```

**script.delay**
The standard script

• sis> source script
  – Extracts iteratively common cubes and factors, resubstitutes them in the node functions, and collapses nodes whose literal-count savings is above a threshold
  – The result is a circuit with a significant improvement in the number of literals
The standard *script* (I)

- **# start script**
- **sweep**
  - Successively eliminates all the single-input nodes and constant nodes from the current network
- **eliminate -1**
  - Eliminates all the nodes in the network whose value is less than or equal to a chosen threshold
  - Value of a node
    - Number of literal saved in the literal count for the network by leaving the node in the network
    - -1: node used only once
The standard *script* (II)

- **simplify**
  - Minimizes the SOP representation of logic function at each node
- **resub -a**
  - Substitutes expressions using the algebraic division
- **gkx -abt 30**
  - Extracts multi-cube divisors from the network
    - **-a**: generates all kernels of all functions in the network when building the kernel-intersection table
    - **-b**: chooses the best kernel intersection as the new factor at each step of the algorithm
    - **-t** threshold: sets a threshold such that divisors are extracted only when their value exceeds the threshold
The standard *script* (III)

- **gcx -bt 30**
  - Extracts common cubes from a network and re-expresses the network in terms of these cubes
  - `-b`: choose the best cube at each step when examining possible cubes to be extracted
  - `-t` threshold: sets a threshold such that only a cube with a value greater than the threshold will be extracted

- **decomp -g ***
  - Decomposes all the nodes in the node list
  - `-g`: use the best algebraic decomposition for the nodes

- **# end script**
The standard *script.rugged*

- `sis> source script.rugged`
  - Extracts factors using *fx*, and quickly reduces the size of the circuit; *full_simplify* is used for powerful node minimization
The standard *script.rugged* (I)

- `# start script.rugged`
- `simplify -m nocomp`
  - simplifies each node in the network
    - `-m nocomp`: invoke the full minimization procedure but does not compute the complete offset
- `full_simplify -m nocomp`
  - simplifies each node in the network (larger DC set)
    - `-m nocomp`: invoke the full minimization procedure but does not compute the complete offset
- `# end script.rugged`
The standard `script.delay`

- `sis> source script.delay`
- Targets performance optimization
The standard \textit{script.delay} (I)

- \texttt{# start script.delay}
- \texttt{sis> decomp -q}
  - Decomposes all the nodes in the network
    - -q: quick decomp algorithm is used which extracts out an arbitrary kernel
- \texttt{sis> tech_decomp -o 2}
  - Decomposes all the nodes in the current network into AND or OR gates
    - -o: OR gates with 2 fanins will be used
The standard *script.delay* (II)

- **sis> resub -a -d**
  - -d: directs resub not to use the complement of a given node in algebraic resubstitution

- **sis> reduce_depth -b -r**
  - command used to improve the speed of a network before technology mapping
    - -b: performs clustering under the duplication ratio 2
    - -r: limits the duplication of logic

- **sis> red_removal**
  - performs sequential redundancy removal
The standard *script.delay* (III)

- **sis> map -n 1 –AFG -p**
  - This command will produce a minimum delay circuit
    - `-n 1`: better tree-covering algorithm
    - `-A`: recovers area after fanout optimization at little delay cost
    - `-F`: performs fanout optimization
    - `-G`: recovers area after fanout optimization at no cost in delay by resizing all gates in the network
    - `-p`: ignores the delay information and forces the arrival times and required times to be all 0

- **# end script.delay**
References – U.C. Berkeley

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