INGENJÖRSHÖGSKOLAN
HÖGSKOLAN I JÖNKÖPING

Technology mapping using SIS

Laboratory 2

in course “Logic synthesis”

2002-version

Written by Tomas Bengtsson and Shashi Kumar

Tomas Bengtsson *Tomas.Bengtsson@ing.hj.se*

Shashi Kumar *Shashi.Kumar@ing.hj.se*

1. Introduction ***3***

2. Documents needed for this lab ***3***

3. Recommended preparations for this lab ***3***

4. Short introduction to FPGAs ***4***

5. Information about CLBs used in this lab ***4***

6. Tasks ***5***

6.1. Making scripts for technology mapping **5**

6.2. Technology-mapping of multiplier **5**

6.3. Technology-mapping of Gray-code converter **5**

6.4. Technology-mapping of a benchmark **5**

7. An example of Technology-mapping ***5***

7.1. Description of example **5**

7.2. Some tips **7**

7.3. The example through SIS **7**

7.3.1. Gate Decomposition 9

7.3.2. LUT Mapping 11

7.3.3. Post-processing commands 12

7.3.4. Programmable Logic Block Generation 14

Tomas Bengtsson Tomas.Bengtsson@ing.hj.se
Shashi Kumar Shashi.Kumar@ing.hj.se

1. Introduction

After a circuit has been optimized using Logic Optimization tools, the next step is to bring the circuit closer to implementation by using the available information about implementation technology. This step is called Technology Mapping. This step involves converting the abstract description (FSM or Boolean functions) of the circuit to a network of limited type of components, normally from a library of components. Due to this reason, Technology Mapping is also sometimes referred as Library Binding. This step involves, selecting components from the library and forming a network of these components. Normally the objectives in Technology Mapping are to have the final implementation using a minimum number of components or to minimize the area of the implementation.

Technology mapping to an FPGA results in the final implementation suitable for a specific FPGA type from a specific company. This is because the internal architecture of FPGAs from different companies is quite different. The internal architectures of various FPGAs from the same company also differ depending on the component series. For example, XILINX 4000 series FPGA has different type of logic blocks as compared to 3000 series. There are two further steps after a circuit has been converted to a network of blocks of a FPGA. These steps are called **Placement** and **Routing**. In the placement step, the logic blocks in the network are assigned specific physical blocks within the FPGA. In the routing step, the used logic blocks are connected using programmable interconnection resources.

In this laboratory, we are only concerned with the first step, which is converting the abstract design to a network of logic blocks for Xilinx FPGA family.

2. Documents needed for this lab

Among the documents from the first lab you will need the document from UCLA (University of California Los Angeles), which describes the extension of SIS for technology mapping. In this document we recommend you to skip the first part and start reading the part starting with a header “Commands provided by UCLA FPGA Mapping Package”. This documents can be found in Appendix A of this lab manual.

A “hand-in” form that you have to fill in to pass the lab is also given. That hand in form and this lab manual can be found in Pingpong.

3. Recommended preparations for this lab

To be able to use the lab time more efficient we recommend you to study the document from UCLA the part mentioned in section 2 “Documents needed for this lab”. It is also recommended that you complete the task described in section 6.1 “Task 1 Making scripts for technology mapping” before the lab.

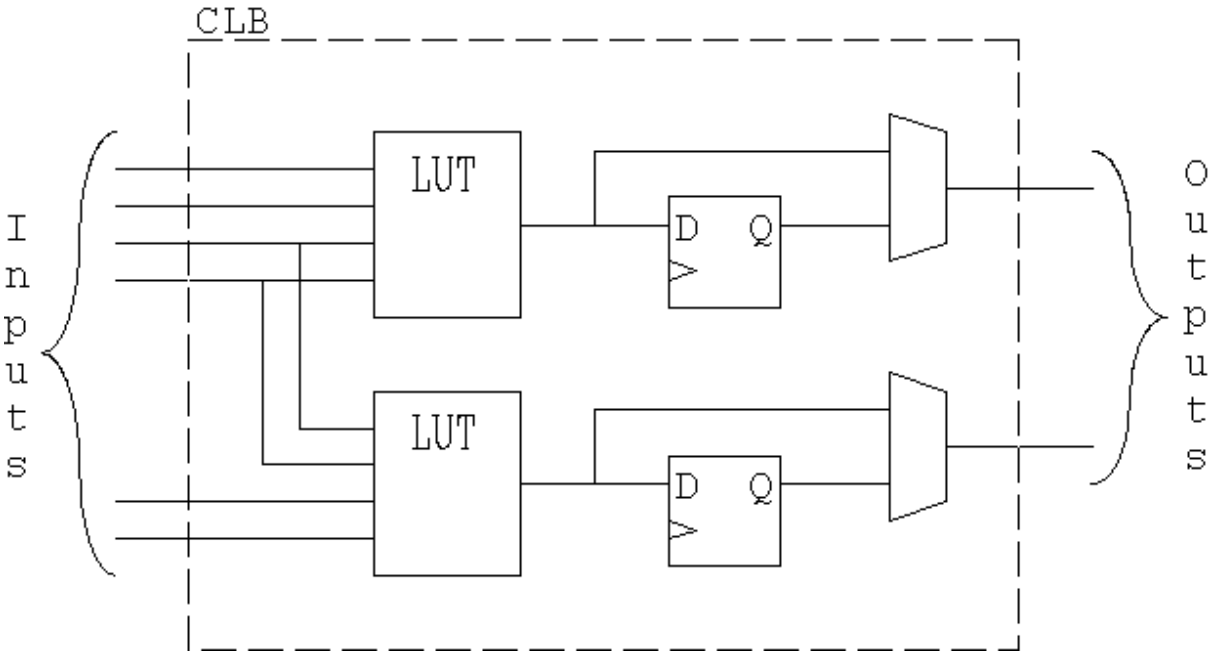
Tomas Bengtsson Tomas.Bengtsson@ing.hj.se
Shashi Kumar Shashi.Kumar@ing.hj.se

4. Short introduction to FPGAs

FPGAs are one family of programmable logic circuits. An FPGA contains programmable logic blocks and programmable interconnection between the blocks. The programmable blocks are called CLBs (Complex Logic Block). The CLBs contain one or more LUTs (Look Up Table). A LUT is a combinatory device with some inputs and one output. It can be programmed to realize any Boolean function. The CLB can be programmed so the output of the LUTs goes to the output of the CLB direct or via a flip-flop. This can be done individually for every LUT. The inputs to the CLB are connected to the inputs in the LUTs. If the CLB contains more than one LUT, some inputs to the CLB may be connected to inputs in more than one LUT.

To connect outputs and inputs of CLBs to other CLBs and to the ports of a chip the programmable interconnection part is used. In this lab we are not going to deal with this. We are only going to map logic into fit CLBs. We will use some old FPGAs, Xilinx3000 – series and Xilinx4000 – series. For our purpose we don't gain anything by using newer ones. The CLBs in both series has two LUTs. The LUTs in Xilinx3000 – series has four inputs and in Xilinx4000 – series they have five inputs.

The picture below shows an example of a simple CLB. The CLBs we will use in this lab looks a little different.



5. Information about CLBs used in this lab

As written in the previous section the LUTs in Xilinx 3000-series have four inputs each and in Xilinx 4000-series the LUTs have five inputs each. The parameter "-k" used in many technology-mapping commands should specify number of inputs to one LUT.

Tomas Bengtsson Tomas.Bengtsson@ing.hj.se
Shashi Kumar Shashi.Kumar@ing.hj.se

6. Tasks

6.1. Task 1 Making scripts for technology mapping

In this task you should prepare scripts for technology mapping. Make one script containing technology-mapping commands, which makes optimization with respect to area minimization for mapping to Xilinx 3000-series. Make another script doing the same but for minimizing the depth of the circuit. Copy those scripts and modify the copies to work for Xilinx 4000-series. You don't need to put the final commands "match_3k" and "match_4k" into the scripts. You can write those commands in the SIS-prompt when you need them instead.

Fill in the scripts in the "hand-in" form.

6.2. Task 2 Technology-mapping of multiplier

In this task you should use your multiplier from the previous lab and make technology mapping in some different ways. In this lab you should alter the following parameters:

- You can either use technology-independent optimization before you make technology mapping or you can skip technology-independent optimization. When you are making technology-independent optimization in this task you should use "rugged-script"
- You can optimize for area or for depth. To do this you should use your scripts from the previous task.
- You can technology-map for either Xilinx 3000-series or Xilinx 4000-series.

The alternatives enumerated above makes eight different combinations of optimizations. Make those and fill in the required results in the "hand-in" form. There are also some questions in the "hand-in" form you should answer.

6.3. Task 3 Technology-mapping of Gray-code converter

In this task you should use the "Gray-code to binary converter" you have made in the previous lab. The task is to technology-map it so it fits into two CLBs in Xilinx 3000-series. Do this and answer the questions in the "hand-in" form!

6.4. Task 4 Technology-mapping of a benchmark

In this task you should technology-map the benchmark "t481.pla". You should map it so that it only requires five LUTs in Xilinx 3000-series. This is the goal of this task and you decide what should be done to get there. Answer the questions in the "hand-in"-form!

7. An example of Technology-mapping

7.1. Description of example

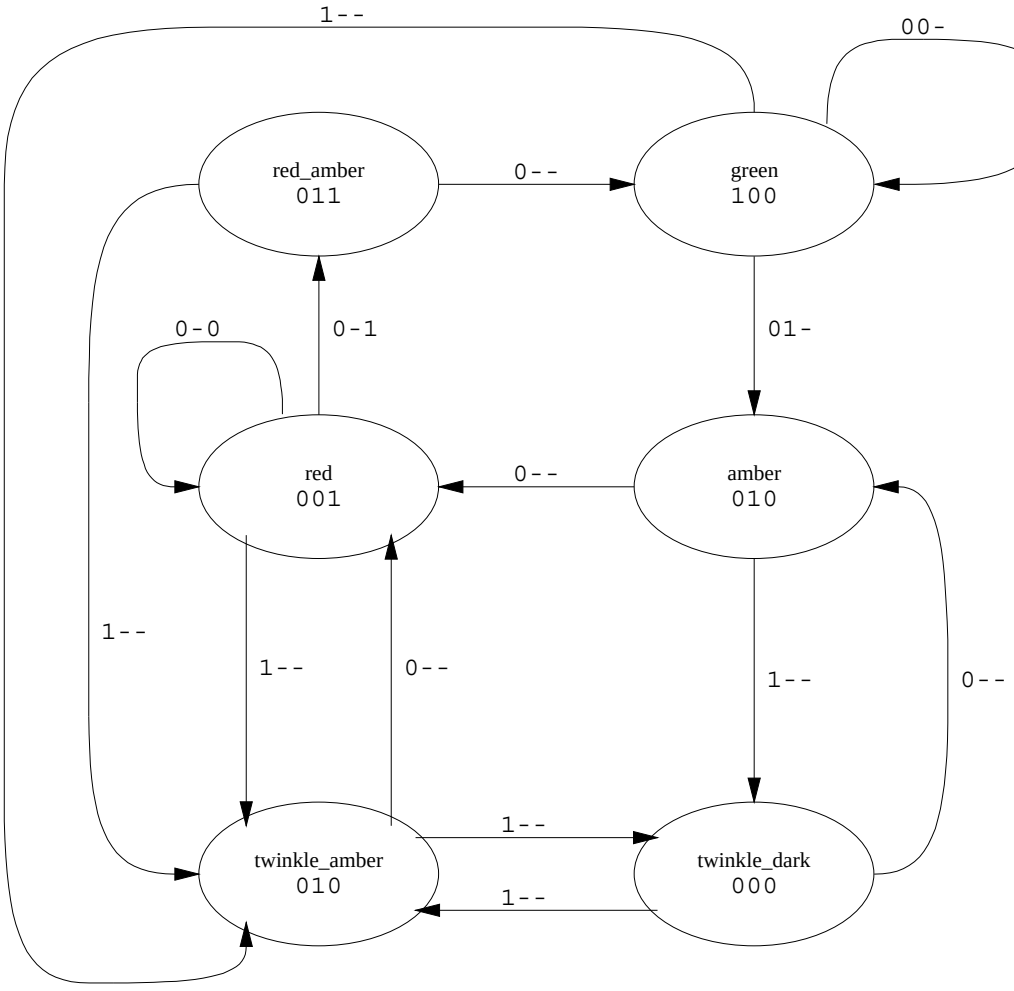
To describe an example of technology mapping, an FSM to control one traffic light is used. This traffic-light controller is nothing that can be used in traffic rather it can be used to show a traffic light fitting in a fair. The controller is made as a Moore-machine.

Tomas Bengtsson Tomas.Bengtsson@ing.hj.se
Shashi Kumar Shashi.Kumar@ing.hj.se

The FSM has three inputs. The first input let the traffic-light run in normal mode if it's "0", and in a mode with twinkle amber (amber ≈ yellow) if it is "1". In the normal mode the traffic light is red, green or it is on its way between. If the second input is "1", when the traffic light is green, it is forced to red via amber. If the third input is "1", when the traffic light is red, it is forced to green via red_amber.

The outputs from the FSM are signals to the three lamps. It is in the order green, amber and red, and "1" means on.

The state-diagram below shows the system.



Tomas Bengtsson Tomas.Bengtsson@ing.hj.se
Shashi Kumar Shashi.Kumar@ing.hj.se

A description of this in kiss-format is shown below:

```
.start_kiss
.i 3
.o 3
00-     green           green           100
01-     green           amber           100
1--     green           twinkle_amber   100
0--     amber           red             010
1--     amber           twinkle_dark    010
0-0     red             red             001
0-1     red             red_amber       001
1--     red             twinkle_amber   001
0--     red_amber       green           011
1--     red_amber       twinkle_amber   011
0--     twinkle_amber   red             010
1--     twinkle_amber   twinkle_dark    010
0--     twinkle_dark    amber           000
1--     twinkle_dark    twinkle_amber   000
.end_kiss
.end
```

This file is available as "/home/beto/public/logic_synthesis/traf.kiss" in the UNIX-system.

7.2. Some tips

It's good to use commands like "print_stats" and "print_level" to see what is happening between the different steps in the optimization and mapping process. Also remember that "write_blif" can give some useful information in some cases.

7.3. The example through SIS

First we make the technology independent optimization. (That is what the first laboratory was about.) We use "state_minimize", "state_assign" and then run "rugged-script". We then get:

```
UC Berkeley SIS with UCLA FPGA Extension (compiled 2-Apr-98 at 11:09 PM)
sis> read_kiss traf.kiss
.start_kiss
sis> state_minimize
Running stamina, written by June Rho, University of Colorado at Boulder
Number of states in original machine : 6
Number of states in minimized machine : 5
sis> state_assign
Running nova, written by Tiziano Villa,  UC Berkeley
Warning: network 'SISEAAa29918', node "v0" does not fanout
Warning: network 'SISEAAa29918', node "v1" does not fanout
Warning: network 'SISEAAa29918', node "v2" does not fanout
sis> source rugged
sis> write_blif
.model traf.kiss
.inputs IN_0 IN_1 IN_2
.outputs OUT_0 OUT_1 OUT_2
.latch    v6.0 LatchOut_v3   1
.latch    v6.1 LatchOut_v4   1
```

Tomas Bengtsson Tomas.Bengtsson@ing.hj.se
Shashi Kumar Shashi.Kumar@ing.hj.se

```
.latch    v6.2 LatchOut_v5   0
.start_kiss
.i 3
.o 3
.p 12
.s 5
.r S1
0-- S0 S2 010
1-- S0 S3 010
0-0 S2 S2 001
0-1 S2 S4 001
1-- S2 S0 001
0-- S3 S0 000
1-- S3 S0 000
0-- S4 S1 011
1-- S4 S0 011
00- S1 S1 100
01- S1 S0 100
1-- S1 S0 100
.end_kiss
.latch_order LatchOut_v3 LatchOut_v4 LatchOut_v5
.code S0 000
.code S2 111
.code S3 001
.code S4 010
.code S1 110
.names LatchOut_v3 LatchOut_v5 OUT_0
10 1
.names LatchOut_v3 LatchOut_v5 OUT_1
00 1
.names OUT_0 LatchOut_v4 OUT_2
01 1
.names v6.1 v6.2 LatchOut_v5 v6.0
11- 1
1-0 1
.names IN_0 IN_1 OUT_1 OUT_2 LatchOut_v5 v6.1
0-1-- 1
0--1- 1
00--0 1
.names IN_0 IN_2 LatchOut_v4 LatchOut_v5 v6.2
--00 1
0011 1
.exdc
.inputs IN_0 IN_1 IN_2 LatchOut_v3 LatchOut_v4 LatchOut_v5
.outputs v6.0 v6.1 v6.2 OUT_0 OUT_1 OUT_2
.names LatchOut_v3 LatchOut_v4 LatchOut_v5 v6.0
10- 1
011 1
.names LatchOut_v3 LatchOut_v4 LatchOut_v5 v6.1
10- 1
011 1
.names LatchOut_v3 LatchOut_v4 LatchOut_v5 v6.2
10- 1
011 1
.names LatchOut_v3 LatchOut_v4 LatchOut_v5 OUT_0
10- 1
011 1
.names LatchOut_v3 LatchOut_v4 LatchOut_v5 OUT_1
```

```
10- 1
011 1
.names LatchOut_v3 LatchOut_v4 LatchOut_v5 OUT_2
10- 1
011 1
.end
```

The key-word ".exdc" means that the following blif-description is the don't-care-set. The description above is the optimized description of the function where don't-cares are forced to one and zero to make the function as small as possible.

7.3.1. Gate Decomposition

In the description of technology mapping from UCLA, it's written that command "tech_decomp" should be run before "dmig"-command is run. The parameter "-k 4" in the "dmig"-command is chosen to 4 because the plan is to map this to an FPGA with 4-input LUTs.

```
sis> tech_decomp -a 1000 -o 1000
sis> dmig -k 4
sis> write_blif
.model traf.kiss
.inputs IN_0 IN_1 IN_2
.outputs OUT_0 OUT_1 OUT_2
.latch    v6.0 LatchOut_v3   1
.latch    v6.1 LatchOut_v4   1
.latch    v6.2 LatchOut_v5   0
.start_kiss
.i 3
.o 3
.p 12
.s 5
.r S1
0-- S0 S2 010
1-- S0 S3 010
0-0 S2 S2 001
0-1 S2 S4 001
1-- S2 S0 001
0-- S3 S0 000
1-- S3 S0 000
0-- S4 S1 011
1-- S4 S0 011
00- S1 S1 100
01- S1 S0 100
1-- S1 S0 100
.end_kiss
.latch_order LatchOut_v3 LatchOut_v4 LatchOut_v5
.code S0 000
.code S2 111
.code S3 001
.code S4 010
.code S1 110
.names LatchOut_v3 LatchOut_v5 OUT_0
10 1
.names LatchOut_v3 LatchOut_v5 OUT_1
```

Tomas Bengtsson *Tomas.Bengtsson@ing.hj.se*

Shashi Kumar *Shashi.Kumar@ing.hj.se*

```
00 1
.names OUT_0 LatchOut_v4 OUT_2
01 1
.names [21] [22] v6.0
1- 1
-1 1
.names [25] [26] [27] v6.1
1-- 1
-1- 1
--1 1
.names [23] [24] v6.2
1- 1
-1 1
.names v6.1 LatchOut_v5 [21]
10 1
.names v6.1 v6.2 [22]
11 1
.names IN_0 IN_2 LatchOut_v4 LatchOut_v5 [23]
0011 1
.names LatchOut_v4 LatchOut_v5 [24]
00 1
.names IN_0 IN_1 LatchOut_v5 [25]
000 1
.names IN_0 OUT_2 [26]
01 1
.names IN_0 OUT_1 [27]
01 1
.exdc
.inputs IN_0 IN_1 IN_2 LatchOut_v3 LatchOut_v4 LatchOut_v5
.outputs v6.0 v6.1 v6.2 OUT_0 OUT_1 OUT_2
.names LatchOut_v3 LatchOut_v4 LatchOut_v5 v6.0
10- 1
011 1
.names LatchOut_v3 LatchOut_v4 LatchOut_v5 v6.1
10- 1
011 1
.names LatchOut_v3 LatchOut_v4 LatchOut_v5 v6.2
10- 1
011 1
.names LatchOut_v3 LatchOut_v4 LatchOut_v5 OUT_0
10- 1
011 1
.names LatchOut_v3 LatchOut_v4 LatchOut_v5 OUT_1
10- 1
011 1
.names LatchOut_v3 LatchOut_v4 LatchOut_v5 OUT_2
10- 1
011 1
.end
```

7.3.2. LUT Mapping

When gate decomposition is done there are some commands to choose between, which map the function to LUTs (Look Up Tables).

```
sis> dagmap -k 4
sis> write_blif
.model traf.kiss
.inputs IN_0 IN_1 IN_2
.outputs OUT_0 OUT_1 OUT_2
.latch    v6.0 LatchOut_v3   1
.latch    v6.1 LatchOut_v4   1
.latch    v6.2 LatchOut_v5   0
.start_kiss
.i 3
.o 3
.p 12
.s 5
.r S1
0-- S0 S2 010
1-- S0 S3 010
0-0 S2 S2 001
0-1 S2 S4 001
1-- S2 S0 001
0-- S3 S0 000
1-- S3 S0 000
0-- S4 S1 011
1-- S4 S0 011
00- S1 S1 100
01- S1 S0 100
1-- S1 S0 100
.end_kiss
.latch_order LatchOut_v3 LatchOut_v4 LatchOut_v5
.code S0 000
.code S2 111
.code S3 001
.code S4 010
.code S1 110
.names LatchOut_v3 LatchOut_v5 OUT_0
10 1
.names LatchOut_v3 LatchOut_v5 OUT_1
00 1
.names LatchOut_v3 LatchOut_v4 LatchOut_v5 OUT_2
01- 1
-11 1
.names [21] [22] v6.0
1- 1
-1 1
.names [25] [26] [27] v6.1
1-- 1
-1- 1
--1 1
.names IN_0 IN_2 LatchOut_v4 LatchOut_v5 v6.2
--00 1
0011 1
.names LatchOut_v5 [25] [26] [27] [21]
01-- 1
```

```
0-1- 1
0--1 1
.names v6.2 [25] [26] [27] [22]
11-- 1
1-1- 1
1--1 1
.names IN_0 IN_1 LatchOut_v5 [25]
000 1
.names IN_0 LatchOut_v3 LatchOut_v4 LatchOut_v5 [26]
001- 1
0-11 1
.names IN_0 LatchOut_v3 LatchOut_v5 [27]
000 1
.exdc
.inputs IN_0 IN_1 IN_2 LatchOut_v3 LatchOut_v4 LatchOut_v5
.outputs v6.0 v6.1 v6.2 OUT_0 OUT_1 OUT_2
.names LatchOut_v3 LatchOut_v4 LatchOut_v5 v6.0
10- 1
011 1
.names LatchOut_v3 LatchOut_v4 LatchOut_v5 v6.1
10- 1
011 1
.names LatchOut_v3 LatchOut_v4 LatchOut_v5 v6.2
10- 1
011 1
.names LatchOut_v3 LatchOut_v4 LatchOut_v5 OUT_0
10- 1
011 1
.names LatchOut_v3 LatchOut_v4 LatchOut_v5 OUT_1
10- 1
011 1
.names LatchOut_v3 LatchOut_v4 LatchOut_v5 OUT_2
10- 1
011 1
.end
```

7.3.3. Post-processing commands

The post-processing command “mpack” can for some cases merge two LUTs into one LUT.

```
sis> mpack -k 4
sis> write_blif
.model traf.kiss
.inputs IN_0 IN_1 IN_2
.outputs OUT_0 OUT_1 OUT_2
.latch    v6.0 LatchOut_v3   1
.latch    v6.1 LatchOut_v4   1
.latch    v6.2 LatchOut_v5   0
.start_kiss
.i 3
.o 3
.p 12
.s 5
.r S1
0-- S0 S2 010
1-- S0 S3 010
```

Tomas Bengtsson Tomas.Bengtsson@ing.hj.se
Shashi Kumar Shashi.Kumar@ing.hj.se

```
0-0 S2 S2 001
0-1 S2 S4 001
1-- S2 S0 001
0-- S3 S0 000
1-- S3 S0 000
0-- S4 S1 011
1-- S4 S0 011
00- S1 S1 100
01- S1 S0 100
1-- S1 S0 100
.end_kiss
.latch_order LatchOut_v3 LatchOut_v4 LatchOut_v5
.code S0 000
.code S2 111
.code S3 001
.code S4 010
.code S1 110
.names LatchOut_v3 LatchOut_v5 OUT_0
10 1
.names LatchOut_v3 LatchOut_v5 OUT_1
00 1
.names LatchOut_v3 LatchOut_v4 LatchOut_v5 OUT_2
01- 1
-11 1
.names [21] [22] v6.0
1- 1
-1 1
.names [25] [26] [27] v6.1
1-- 1
-1- 1
--1 1
.names IN_0 IN_2 LatchOut_v4 LatchOut_v5 v6.2
--00 1
0011 1
.names LatchOut_v5 [25] [26] [27] [21]
01-- 1
0-1- 1
0--1 1
.names v6.2 [25] [26] [27] [22]
11-- 1
1-1- 1
1--1 1
.names IN_0 IN_1 LatchOut_v5 [25]
000 1
.names IN_0 LatchOut_v3 LatchOut_v4 LatchOut_v5 [26]
001- 1
0-11 1
.names IN_0 LatchOut_v3 LatchOut_v5 [27]
000 1
.exdc
.inputs IN_0 IN_1 IN_2 LatchOut_v3 LatchOut_v4 LatchOut_v5
.outputs v6.0 v6.1 v6.2 OUT_0 OUT_1 OUT_2
.names LatchOut_v3 LatchOut_v4 LatchOut_v5 v6.0
10- 1
011 1
.names LatchOut_v3 LatchOut_v4 LatchOut_v5 v6.1
10- 1
011 1
```

```
.names LatchOut_v3 LatchOut_v4 LatchOut_v5 v6.2
10- 1
011 1
.names LatchOut_v3 LatchOut_v4 LatchOut_v5 OUT_0
10- 1
011 1
.names LatchOut_v3 LatchOut_v4 LatchOut_v5 OUT_1
10- 1
011 1
.names LatchOut_v3 LatchOut_v4 LatchOut_v5 OUT_2
10- 1
011 1
.end
```

7.3.4. Programmable Logic Block Generation

In our installation of SIS it is possible to map to Xilinx 3000 and 4000 –series.

```
sis> match_3k -v
##PI=3  #PO=3   #LUT=11 #CLB=6  #LEVEL=3
#0001: ( OUT_2 , v6.2 )
#0002: ( OUT_1 , [26] )
#0003: ( OUT_0 , [27] )
#0004: ( v6.1 , [21] )
#0005: ( v6.0 , [25] )
#0006: ( [22] ) sis> match_3k -v
```

The argument “-v” makes it print the list about which LUTs should be in the same CLB.

8. Appendix

Appendix A

```
 +-------------------------------------------------------------------------+
 | RASP_SYN: LUT-Based FPGA Technology Mapping Package (Release B 1.0)     |
 |           -- Synthesis Core of the UCLA RASP Systems                    |
 +-------------------------------------------------------------------------+
 | Copyright (C) 1991-1997 the Regents of University of California         |
 +-------------------------------------------------------------------------+
 | Authors: Eugene Ding, VLSI CAD Lab, UCLA CS Dept. <eugene@cs.ucla.edu>  |
 |          Yean-Yow Hwang, VLSI CAD Lab, UCLA CS Dept.<yeanyow@cs.ucla.edu>|
 |          Chang Wu, VLSI CAD Lab, UCLA CS Dept. <changwu@cs.ucla.edu>    |
 |          Songjie Xu, VLSI CAD Lab, UCLA CS Dept. <sxu@cs.ucla.edu>      |
 | Project Director:  Prof. Jason Cong, UCLA CS Dept. <cong@cs.ucla.edu>    |
 +-------------------------------------------------------------------------+
 | This release includes the following mapping algorithms:                 |
 |      DAG_Map   version 1.0                                              |
 |      FlowMap   version 2.1                                              |
 |      FlowMap-r version 2.0                                              |
 |      FlowSYN   version 2.0                                              |
 |      CutMap    version 1.2                                              |
 |      ZMap      version 1.0                                              |
 |      TurboMap  version 1.0                                              |
 +-------------------------------------------------------------------------+

-------------------
<0> ACKNOWLEDGEMENT
```

```
-------------------

The FlowMap and CutMap and TurboMap packages are integrated into the SIS
system and uses many of the routines provided by SIS. The SIS system was
developed in UC Berkeley Electronic Research Lab.

--------------------------------------
<1> RELEASE AGREEMENT AND CONTACT INFO
--------------------------------------

Please refer to "release.statement".

-----------
<2> CONTENT
-----------

sis                     -- binary of SIS compiled with FlowMap and
                           CutMap packages.
doc                     -- this file.
release.statement       -- to be read first.
rasp_syn                -- a csh script of FPGA mapping
select                  -- mapping result selector

This release contains programs primarily developed by September 1997.
More functions will be added to future release when they are stablized.
It runs on Sun SPARCstation under SunOS 4.1.3 and Solaris.

Some commands are not included in the release due to nondisclosure
agreement.

RASP_SYN package provides a complete solution to SRAM-based FPGA mapping
engine. The entire flow of RASP_SYN is:

    1. gate decomposition to get K-bounded circuit, where K is the
       fanin limit of LUTs of the target architecture
    2. generic LUT mapping
    3. post-processing mainly for area reduction
    4. architecture specific mapping.

RASP_SYN comes with a user-friendly csh script for the ease of use.
However, you can modify the script or write your own based on your
specific needs.

------------------------
<3> TECHNICAL REFERENCES
------------------------

J. Cong, Y. Ding, "An Optimal Technology Mapping Algorithm for Delay
  Optimization in Lookup-Table based FPGA Designs," IEEE Trans. on CAD,
  Vol. 13, No. 1, Jan. 1994, pp. 1-12.

J. Cong, Y. Ding, "On Area/Depth Trade-off in LUT-Based FPGA Technology
  Mapping," IEEE Trans. on VLSI Systems,  Vol 2., No. 2, June 1994,
  pp. 137-148.

J. Cong, Y. Ding, T. Gao, K. Chen, "LUT-Based FPGA Technology Mapping
  under Arbitrary Net-Delay Models," Computers & Graphics,
  Vol.18, No.4, 1994, 507-516.

J. Cong, Y. Ding, "Beyond the Combinatorial Limit in Depth Minimization
  for LUT-Based FPGA Designs," Proc. 1993 IEEE/ACM Int'l Conf. on CAD,
  Santa Clara, CA, Nov. 1993, pp. 110-114.

K.Chen, J.Cong, Y.Ding, A.Kahng, P.Trajmar, "DAG-MAP: Graph-Based
```

FPGA Technology Mapping for Delay Optimization," IEEE Design & Test of Computers, Sept. 1992

J. Cong, J. Peck, Y. Ding, "RASP: A General Logic Synthesis System for SRAM-based FPGAs," Proc. ACM 4th Int'l Symp. on FPGA, pp. 137-143, 1996

J. Cong, Y. Hwang, "Simultaneous Depth and Area Minimization in LUT-Based FPGA Mapping," Proc. ACM 3rd Int'l Symp. on FPGA, Feb. 1995, pp. 68-74.

J. Cong, Y. Hwang, "Structural Gate Decomposition for Depth-Optimal Technology Mapping in LUT-based FPGA Designs," Proc. ACM/IEEE 33rd Design Automation Conf., pp. 726-729, 1996.

J. Cong, C. Wu, "An Improved Algorithm for Performance Optimal Technology Mapping with Retiming in LUT-Based FPGA Design," Proc. IEEE Internal Conference on Computer Design, pp. 572-578, 1996

Xilinx, FPGA Data Book, 1994

<4> USAGE

4.1 Running with a super script

Super Script of UCLA FPGA Mapping

```
Usage: rasp_syn circuit -sis path -k k -device xc3k/xc4k -algo algo -relax r
                        -objective area/delay/tradeoff/all
```

Rasp_syn is a csh script for an easy usage of UCLA FPGA Mapping algorithms. In default, the input is in EQN format with extension .eqn. The output is an LUT network with/without matching information in EQN format as well. Please keep the program "select" in the current directory.

To use other data formats as BLIF or SLIF which are supported by SIS of UCB, please set FMT in rasp_syn script to blif or slif and use .blif or .slif as the name extension of the input file. The output format will be changed automatically, except the CLB matching file format, which will be kept in EQN format. For Xilinx XC3K/XC4K CLBs, the CLB clustering information will be presented as:

```
#CLB_number: (lut1, lut2)
lut1 = ..
lut2 = ..
```

There are two ways to run rasp_syn:

1. Running with single given mapping algorithm

   The algorithm must be specified with option -algo algorithm. The target is K-LUT. The output circuit is in circuit.k in EQN format.

2. Running with multiple algorithms

   Rasp_syn can run all the built-in algorithms automatically and return the best result (in terms of area or delay) or a set of results based on area-delay tradeoff or all the results for you.

   To run multiple algorithms, you simply do not specify any algorithm with -algo option.

Options

```
    -sis        Specify the path of sis. The default is sis and the path
                must be specified in the environment.
    -k          Used only in single algorithm mode. K is the input number
                of LUTs. The output is in circuit.k.
    -device     Used only in multi-algorithm mode. This is the default mode.
                The current supported devices are:
                xc3k    Xilinx XC3000 Family
                xc4k    Xilinx XC4000 Family
    -algo       Specify the mapping algorithm in single algorithm mode.
                The current supported algorithms are:
                flowmap: FlowMap
                flowmap-r: FlowMap-r
                flowsyn: FlowSYN
                cutmap:  CutMap
                zmap:    ZMap for delay
                zma:     ZMap for area
    -relax      Used only in single algorithm mode with FlowMap-r.
                R is the depth relaxation.
    -objective  Used only in multi-algorithm mode. The objective can be:
                area:   Area first.     This is the default objective.
                delay:  Depth first
                tradeoff: Area-delay tradeoff
                all:    All the results
```

4.2 Running SIS without the super script

SIS is a complete logic synthesis package. All of the following commands have been built in SIS which can be run directly from SIS.

Commands provided by UCLA FPGA Mapping Package

---

1. Gate Decomposition Commands

* dmig [-k <K_value>] [-f]

  Decompose a simple gate network into a K-bounded network
  (i.e. each gate has no more than K inputs), or
  complex gates into K-bounded gates with -f option.
  For obtaining a simple gate network, use sis command
  "tech_decomp -a 1000 -o 1000."

  -k specifies max. gate input size K, with a default value 2.

  -f decompose complex gates in the network

* dogma [-k <K_value>]

  Decompose a simple gate network into a 2-bounded network
  such that flowmap, cutmap, or zmap can obtain a best (small) depth.

  -k specifies the LUT input size K, with a default value 5.

---

2. LUT Mapping Commands

* dagmap [-k <K_value>]

  Map a K-bounded network into a K-LUT network of small depth

```
  (might not be optimal).

  -k  specifies the LUT input size K, with a default value 5.


* flowmap [ -k <K_value> ] [-r <R_value> ] [ -s <S_value> ]

  Map a K-bounded network into a K-LUT network of optimal depth,
  or within the optimal depth plus R.
  Area can be further reduced by post-processing packing routines.

  -k  specifies the LUT input size K, with a default value 5.

  -r  specifies the relaxed depth value R.
      If -r is not used, every node is at its optimal depth,
      -r 0 will trade depth on non-critical paths for a smaller area
      (the LUT network still has an optimal depth),
      -r R will allow depth to increase by R (then dfmap is called to
      reduce the area).

  -s  specifies the cone input size S for which resynthesis of cones
      are performed for a smaller LUT network depth.

* dfmap [ -k <K_value> ]

  Map a K-bounded network into a K-LUT network of optimal area
  WITHOUT any node duplication.
  It is used after flowmap -r and mffc_shrink, and is followed
  by a LUT packing procedure. For example, we use dfmap in
  "flowmap -k 5 -r 1; mffc_shrink -k 5; dfmap -k 5; greedy_pack -k 5"

  -k  specifies the LUT input size K, with a default value 5.


* cutmap [ -k <K_value> ] [-x ]

  Map a K-bounded network into a K-LUT network of optimal depth
  with simultaneous area minimization.
  Area can be further reduced by post-processing packing routines.

  -k  specifies the LUT input size K, with a default value 5.

  -x  specifies depth relaxation on non-critical paths.

* zmap [ -k <K_value> ] [-c ]

  Map a K-bounded network into a K-LUT network of optimal depth
  with simultaneous area minimization (cut enumeration approach).
  Area can be further reduced by post-processing packing routines.

  -k  specifies the LUT input size K, with a default value 5.

  -c  will minimize area only with no bound on depth

* turbomap [ -k <K_value> ] [-c <clock_value> ] [ -a <area_reduction> ]

  Map a K-bounded network into a K-LUT network with the minimum clock
  period. Area can be further reduced by post-processing packing routines.

  -k  specifies the LUT input size K,
      default value: 5.

  -c  specifies an upper-bound on the clock period,
      -1: no upper-bound, (default)
```

```
  -a  specifies whether to perform clock period bounded area minimization
      1: allow label relaxation, 2: no label relaxation
      default: no area reduction



-----------------------------------------------------------------------
3. Post-Processing Commands

* mpack [ -k <K_value> ] [ -m ] [ -p | -g ]

  Reduce the number of nodes in the K-bounded network through
  predecessor packing and/or gate decomposition.
  (Each node can be regarded as a LUT).

  -k  specifies the LUT input size K, with a default value 5.

  -m  enable collapsing into multiple fanouts (for better results)

  -p  predecessor packing only, no gate decomposition.
  -g  gate decomposition only, no predecessor packing.


* greedy_pack [ -k <K_value> ] [ -m ] [ -p | -g ]

  Same as mpack, but with a fast graph matching heuristics.

* mppack = mpack -p
* greedy_ppack = greedy_pack -p


* flowpack [ -k <K_value> ]

  Pack LUTs together when possible for reducing area.

  -k  specifies the LUT input size K, with a default value 5.




-----------------------------------------------------------------------
4. Programmable Logic Block Generation Commands

* match_3k [ { -v | -f <outfile> } [ -e ] ]

  Pair up 4-LUTs for XC3000 CLBs and print pairing results.
  (Two 4-LUTs with 3 common inputs can be put in one CLB.)
  The network is not changed.

  -v  print the CLB output(s) to standard output

  -f  print the CLB output(s) to <outfile>

  -e  also print LUT network functions in EQN format


* match_4k [ { -v | -f <outfile> } ] [-h <threshold> ]

  Group up LUTs for XC4000 CLBs and print grouping results.
  (If 5-LUT can be decomposed properly, it can be put into one
   CLB with an additional 4-LUT.)
  The network is not changed.
```

Tomas Bengtsson Tomas.Bengtsson@ing.hj.se
Shashi Kumar Shashi.Kumar@ing.hj.se

```
  -v  print LUT grouping results and LUT network functions to
      standard output

  -f  print LUT grouping results and LUT network functions to <outfile>

  -h  employ fast 4-LUT pairing heuristics when the number of
      4-LUTs is more than <threshold> in the LUT network



------------------------------------------------------------------------
5. Utility Commands

* mffc_shrink [ -k <K_value> ]

  Collapse MFFCs (with no more than K inputs) into a single node.
  This is a pre-processing step of DF-Map for improving dfmap results.

  -k  specifies MFFC input size K, with a default value 5.

* fl_nstatus [ -s ]

  Print network information (number of PIs, POs, levels, nodes, etc.).

  -s  also print node fanin/fanout distributions

* prn [ -m ]

  Print network information (number of PIs, POs, levels, nodes, etc.).

  -m  also print statistics on the MFFCs of the network
```