PROJECT: ALLIANCE DESIGN FLOW

Ain Shams University

PART 2
Logic Synthesis

Frederic AK Kai-shing LAM

Modified by LJ
Pierre & Marie Curie University

Adapted by M. Dessouky

Ain Shams University
2017

Contents

1 Introduction **3**

2 Steps to follow **3**

2.1 FSM synthesis 3

2.2 Boolean network optimization 3

2.3 Library *Mapping* 4

2.4 Netlist optimization 4

2.5 Netlist visualization 4

2.6 Netlist checking 4

2.7 Delay Simulation 5

2.8 Scan-path insertion 5

2.9 Scan-path Simulation 6

1 Introduction

In this project, you'll build a simple frame decoder chip. There are four parts of the project:

1. High-Level Design.
2. Low-Level Synthesis (this document).
3. Design for test (DFT) (this document).
4. Placement and Routing.

This document describes parts (2) and (3) of the project; Low-Level (Logic) Synthesis.

The **ALLIANCE** tools used are:

- **syf**: Finite State Machine synthesizer
- **boom:** BOOlean Minimization.
- **boog:** Library Binding.
- **loon:** Local optimizations of Nets.
- **xsch:** Graphical netlist viewer.
- **flatbeh:** Behavioral from Structural.
- **proof:** Formal Verification.
- **scapin**: Scan-path insertion (DFT)

In addition to **ModelSim** for simulation.

2 Steps to follow

In the previous part of the project, we obtained a golden FSM system description and developed a testbench for verification.

2.1 FSM synthesis

- Synthesize your description with SYF using different state encoding algorithms -a, -j, -m, -o, -r and by using the options -CEV.

```
> syf -CEV -a <fsm_source>
```

Several dataflow descriptions (.vbe) corresponding to different state encodings are obtained.

2.2 Boolean network optimization

To analyze Boolean optimization effect:

- launch Boolean optimization with the tool **BOOM** by asking an appropriate optimization in **delay** and **area** for each of the (.vbe) obtained above:

```
>boom -V -d <optimization_%>  <vbe_source> <vbe_destination>
```

- **construct a table in project documentation** to compare the literal number after factorization for each case.

2.3 Library Mapping

For each Boolean network obtained previously:

- set properly environment variables
- using the parameter file (paramfile.lax), refer to the lecture slides:
    - choose an appropriate **Optimization Mode**
    - choose an appropriate **Optimization Level**
    - set the output load capacitance on the outputs to 100 fF
- synthesize the structural view: Make sure that the output file format is set to VHDL structural (vst) through the environment variable MBK_OUT_LO=vst

```
> boog -l paramfile <vbe_source>
```

- launch **BOOG** on different *netlist*s to observe **SYF** options influence (different state encoding techniques).

2.4 Netlist optimization

For all the structural view obtained previously:

- launch **LOON** with the command:

```
>loon <vst_source> <vst_destination>  paramfile
```

This command also uses the parameter file (paramfile.lax) created in the previous step.

- **<u>construct another table in project documentation</u>** to compare all realizations with respect to delay and area. The table is required in the project report.
- **choose the best implementation**. Continue the rest of the project with only the best implementation.

2.5 Netlist visualization

- The longest path (critical path) is described in the **xsc** file produced by **LOON**. The **XSCH** tool will use it to highlight this path on the schematic. To launch the graphical schematic viewer:

```
>xsch -I vst -l <vst_source>
```

- The red color indicates the critical path.

2.6 Netlist checking

Verify the chosen *netlist* using formal verification:

- Make a formal comparison of your netlist with the original behavioral file resulting from **SYF**:

```
>flatbeh <vst_source> <vbe_dest>
```

```
>proof -d <vbe_origin> <vbe_dest>
```

Checks if the files are formally identical.

2.7 Delay Simulation

The obtained gate-level netlist has been functionally verified. Standard-cell delay has been ignored during all the above steps. Note that both **BOOG** and **LOON** synthesis and optimization tools have estimated the critical path delay. It should be easy to verify that this delay is less than the required clock period. It is time now for a delay simulation to double-check the speed performance.

Validate the synthesis results using **ModelSim** using the same testbench and assertions which was used to validate the initial behavioral FSM as follows:

- Copy **<file_name>.vst** to **<file_name>.vhd**
- Transfer **<file_name>.vhd** to Windows
- Add to the netlist

  LIBRARY sxlib_ModelSim;
- The standard cell library Sxlib must be imported into **ModelSim** as shown before.
- Rerun the testbench and validate the timing.

2.8 Scan-path insertion (DFT)

With the **SCAPIN** tool, we can insert a scan-path into the netlist. The scan-path allows the designer to observe in test mode the stored values of all registers of the circuit. The path is created by changing each register into a mux_register (i.e. by inserting a multiplexer in front of all registers) and connecting them in series.

Add a scan path to the chosen *netlist* as follows:

- Prepare a ".path" file. Open the gate-level netlist file obtained from synthesis ".vst", search for registers. Register component name in Alliance standard cell library is "sff1". Then put them in the ".path" file as follows
  # Example of a ".path" file

```
BEGIN_PATH_REG
ac_cs_0_ins
ac_cs_1_ins
ac_cs_2_ins
END_PATH_REG

BEGIN_CONNECTOR
SCAN_IN      scanin
SCAN_OUT     scanout
SCAN_TEST    test
END_CONNECTOR
```

  where ac_cs_0_ins, ac_cs_1_ins and ac_cs_2_ins are the registers in the input netlist (.vst) file that will be placed in the scan-path.

- Insert a scan-path connecting all the design registers with the following command:

```
>scapin -VRB <vst_source> <path_file> <vst_dest>
```

2.9 Scan-path Simulation

Extend the original testbench to test the scan-path and simulate with **ModelSim**. Refer to section 2.7, in addition to the DFT lecture.