IL2225 – Lab 1

Logic synthesis I

Name: .....

Personnummer: .....

Date: .....

Assistant: .....

1 Objective

The purpose of this lab is to introduce the basics of logic synthesis using the Synopsys Design Compiler. Upon completion of this exercise, you should be able to discuss the role of logic synthesis within the overall design process and describe each of the basic steps in logic synthesis. As shown in the Figure 1, RTL code, design constraints, and the technology library files are the inputs to the design synthesis tool. Design synthesis tool maps the design on the library cells and generates the gate level netlist as the output.

Synthesis includes the following steps:

- Translation of the VHDL code into technology independent logic components, like AND/NAND etc.
- Optimization of the technology independent Netlist.
- Mapping of this technology independent Netlist on a target technology.

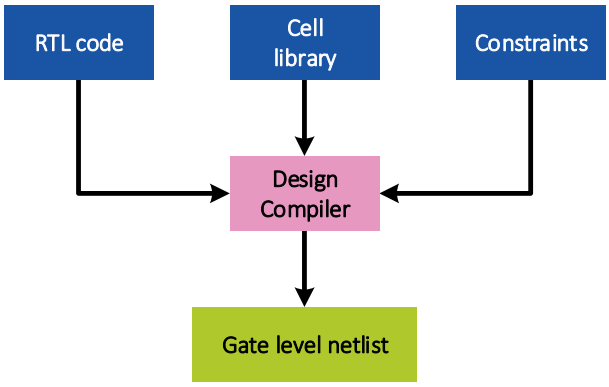


Figure 1. Logic synthesis

In this lab, we implement a FIR filter of order N=13 in VHDL. The equation to implement a FIR filter is given as:

$$y_n = \sum_k x_{n-k} \cdot c_k$$

Where *x* is the sample vector, also known as the *delay line* since it preserves the previous *k-1* delayed samples. *c* is the impulse response of the filter, also known as coefficients.

A new *x*, *x(0)*, is sampled at every sample period, marked by sample clock. When a new sample arrives, the previous samples are shifted, so that the oldest sample *x(k-1)* is shifted out.

2 Preparation tasks

Complete these tasks to get familiar with the designs that will be used during the lab.

2.1 Lab directory structure

For labs 1 and 2 you should use the following directory structure. These directories will have the following purposes.

- `IL2225_lab`: This will be the main directory which will have three scripts. The global scripts directory will be used to store the scripts to instantiate different technology libraries.
- `lab1, lab2`: These directories will store the files for the respective labs.
- `serial_fir, parallel_fir`: These directories should be inside lab1 and will be used for each of the designs
- `synopsys_dc.setup`: This file will contain the setup information for the design vision and will be discussed in the next section.
- `work`: This directory will contain the compiled code to perform the simulations.
- `rtl`: Contains the design files for synthesis
- `tb`: This directory contains the testbench for your designs
- `syn/scr`: You can save your synthesis scripts in this directory
- `syn/db`: The synthesized gate level netlist generated by the synthesis will be stored in this directory.
- `syn/rpt`: All the generated reports will be saved in this directory.

2.2 Fully parallel FIR

For this laboratory, the VHDL code for a fully parallel FIR filter is given in a folder `IL2225_lab/lab1/parallel_fir`. The architecture of the fully parallel FIR is shown in Figure 3. To implement the delay line a shift register is used.

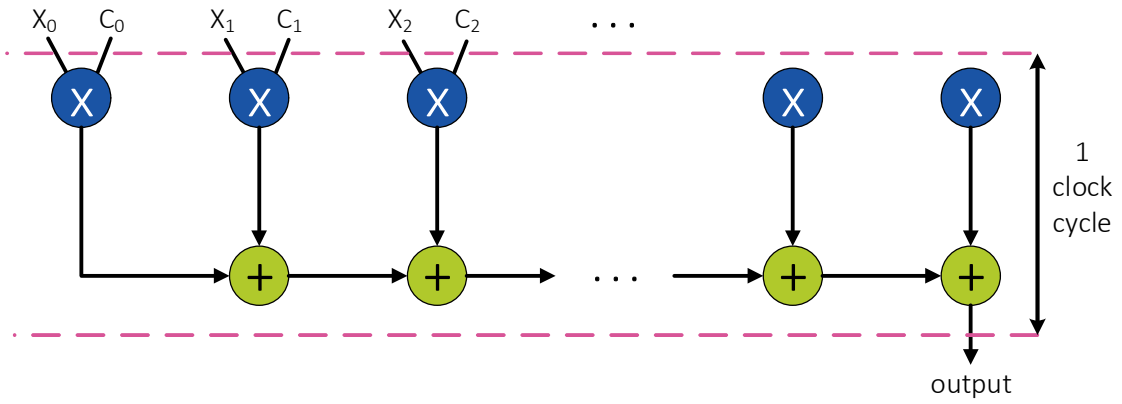


Figure 3. Fully parallel 13-tap FIR filter

Draw: Read the code and make a block diagram of the code on the paper so that we could know your understanding of the overall functionality of the code.

Simulate: Do a functional simulation of the code. A fully functional test-bench is given. This test-

bench gives only one sample to the shift register. Be ready to show your simulation during the lab check. You can use ModelSim to simulate the design. A simple simulation script is provided under `tb/` directory. You can invoke that script using the `-do` option of `vsim` command, or by running `source SCRIPT_PATH` in the ModelSim shell

Question 1: How many adders and multipliers are there in the critical path of the FIR filter with the architecture shown in Figure 3?

The filter is symmetric which means $c_i = c_{k-i}$. Therefore, the FIR equation can be written:

对称的

$$y = (x_0 + x_{12}) \cdot c_0 + (x_1 + x_{11}) \cdot c_1 + \cdots (x_6 + 0) \cdot c_6$$

If the FIR filter is symmetric, it is more efficient to implement it as shown in the figure 4.

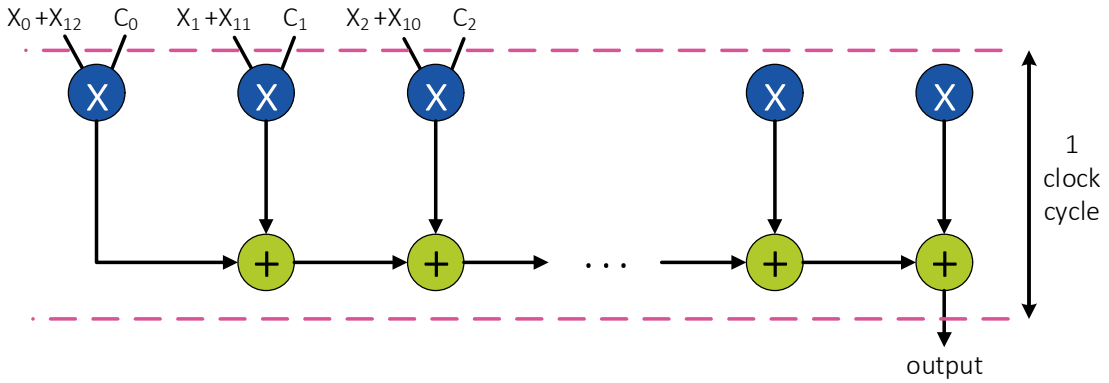


Figure 4. Symmetric fully parallel 13-tap FIR filter

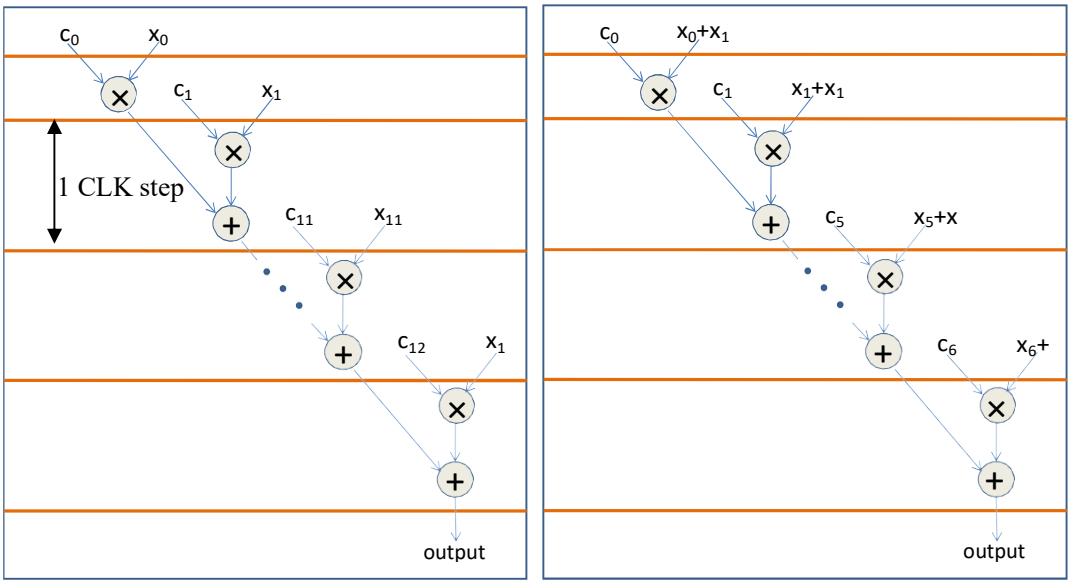
Question 2: How many adders and multipliers are there in the critical path of the FIR filter with the architecture shown in Figure 4? Comparing the architectures shown in figure 3 and 4 which one has a shorter critical path?

2.3 Fully serial FIR

In the `IL2225_lab/lab1/serial_fir` folder, the given code implements a fully serial FIR filter (Figure 5.a.) Each line in Figure 5 represents one clock step.

Draw: Read the code and make a block diagram of the code on the paper so that we could know your understanding of the overall functionality of the code.

Simulate: Do a functional simulation of the code. A fully functional test-bench is given. This test-bench gives only one sample to the circular buffer.



(a)Fully serial FIR filter (b) Symmetric fully serial FIR filter

Figure 5. Two different architectures for fully Serial FIR filter

The efficient way of implementing a symmetric fully serial FIR filter is shown in figure 5.b.

Question 3: How many adders and multipliers are there in the critical path of the FIR filter with the architecture shown in Figure 5.a and 5.b?

Question 4: Compare the number of the clock cycles that each of the architectures (5.a and 5.b) takes to produce the output. Which design has more throughput?

2.4 Partially parallel FIR filter

In this lab, you are required to implement an asymmetric partially parallel FIR filter. Save your code in `IL2225_lab/lab1/partially_parallel_fir/rtl`. The architecture of the partially parallel FIR filter is shown in Figure 6.

A few hints on the partially parallel implementation:

- You can use the code of the serial FIR filter as the base to implement the partially parallel architecture.
- In every clock cycle, **two** MACs are working in parallel. Therefore, in every clock cycle, the FSM should generate two addresses to read samples from the circular buffer (Figure 7).
- Note that if the number of filter taps is odd, in the last cycle only one MAC works.

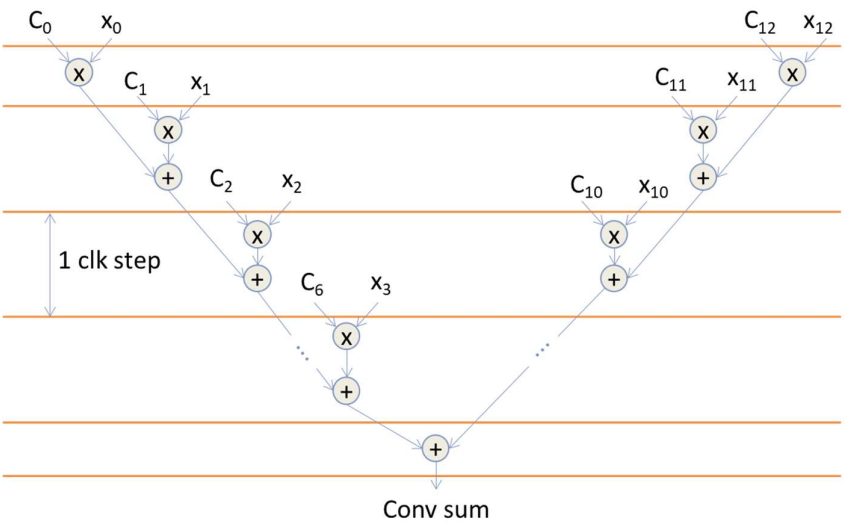


Figure 6. Partially parallel FIR filter

Test the functionality of your code.

Question 5: How many adders and multipliers are there in the critical path of the FIR filter with the architecture shown in Figure 6?

Question 6: How many clock cycles does this architecture take to produce the output?

Draw: Create a table and compare the critical path and throughput between the different architectures.

3 Design Vision initialization

When using the Synopsys synthesis tool, there are startup files that are read by Synopsys. The name of this start up file is `.synopsys_dc.setup`. A `.synopsys_dc.setup` file is a script file that is executed automatically when `design_vision` or `dc_shell` is invoked. The setup file is useful for setting up variables and commands that you want to execute every time you invoke Design Vision. In this start up file four important parameters must be set before you can perform any synthesis.

- `search_path`: The parameter search_path is used to specify to the synthesis tool all the paths that it should search in when looking for a synthesis technology library to reference during synthesis.
- `target_library`: The file pointed to by the parameter target_library is the library that contains all the logic cells for mapping during synthesis.
- `symbol_library`: The parameter symbol_library points to the library that contains visual information on the logic cells in the synthesis technology library. All logic cells have a symbolic representation and information about the symbols is stored in this library.
- `link_library`: The parameter link_library points to the library that contains information on the logic gates. Design Vision reads the setup file in the root of the home directory. Here the information that is general for all designs should be stored. A typical setup file located in the root of the home directory is:

```
set designer "Tom Cruise"
set company "KTH"
[getenv "SYN"]
set search_path "${SynopsysHome}/libraries/syn"
```

Then the setup file in the current directory is read. Here design specific information should be stored, e.g. the file that specifies the target technology.

```
set target_library "tcbn90gtc.db"
set symbol_library "tcbn90g.sdb"
set link_path "$search_path"
```

The Design Compiler is invoked by typing `dc_shell` in the UNIX shell. By using the -f option a TCL script can be provided, and the script will be executed once DC starts. Once DC has started the GUI can be launched by writing `gui_start`

3.1 Design Vision Start-Up

1. Move to directory LAB1

```
Prompt> cd IL2225_labs/lab1/serial_fir
```

2. Invoke the design vision

```
/IL2225_labs/lab1/serial_fir> dc_shell -gui
```

3. Setup the path of the libraries by running the `.synopsys_dc.setup` script.

```
dc_shell> source .synopsys_dc.setup
```

4. Whenever you need help, you can access online documentation by clicking **Help/online help.**

5. The path of the library is

```
/afs/it.kth.se/pkg/synopsys/extra_libraries/standard_cell/TSMC/tcbn90g_110a/Front
_End/timing_powe r/tcbn90g_110a/
```

At this path you can find all the required libraries. So, whenever you need to find a specific library file, look into this path.
If any of the commands given in this lab is not working, try to press tab while entering the command in the command window in design vision. You will see a quick help telling you the exact syntax of the command.
You must know the reasons for the results you will be entering in the tables in rest of the lab. These reasons will help us judge if you have enough understanding to pass this lab.

4 Logic Synthesis Steps

We have already mentioned the process of synthesis which is described as translation plus optimization plus mapping. In terms of the Synopsys tools, translation is performed by the *analyze* plus *elaborate* command sequence. Optimization and mapping are performed by the compile command. To run each of the synthesis steps, both GUI and scripts can be used. To automate the synthesis process, you can create a script containing all of the required commands.

4.1 Design Entry

Before synthesis the design must be entered into DC in a standard format (VHDL, Verilog, EDIF, db etc). DC provides the following two methods of design entry: (a) *read* command (b) *analyze*+*elaborate* commands.

Read or Analyze

- The *read* command reads in files that are in another format than HDL, such as *dcc* and *pla*. Although it supports HDL format, it does not do the checking accomplished by analyze and elaborate.
- The *analyze* command reads a VHDL or Verilog file, checks for proper syntax and synthesizable logic, and stores the design in an intermediate format. Remember that you should read/analyze the files in order: 1- Packages, 2- All VHDL modules except top module, 3- Top level module (`parallel_fir` for example).

```
analyze -format vhdl -lib work VHDL_FILE_PATH
```

Elaborate

The elaborate command creates a design from the intermediate format produced by analyze. The elaborate command replaces the HDL operators in the design with synthetic operators and determines the correct bus sizes. The design is now expressed by generic technology independent components (GTECHlibrary). For example:

```
elaborate TOP_PARALLEL_FIR -architecture STRUCTURAL -library DEFAULT
```

4.2 Setting constraints

In this section we will learn to set constraints on our design. Refer to Lecture 3 "Logic Synthesis – Constraints" for more details on the constraint meaning and setup process.

In general, the constraints can be of following kinds:

Selecting Wire Load Models

The wireload model contains information that DC utilizes to estimate the interconnect delays during the pre-layout phase of the design. Several models appropriate to the different size of the logic are included in the technology library. These models define the capacitance, resistance, area factors. You can check all the available models by checking the tcbn90gtc.lib.

Question 7: Open this file in an editor and find the available wire_load models. List four of them in the following table. What is the difference between the wireload **models** that you have listed?

Table: Wireload model

Wire-load-model	Resistance	Capacitance	Slope

Select wireload mode by:

```
set_wire_load_mode (top/enclosed/segmented)
```

Select one of the wireload models using this command:

```
set_wire_load_model -name ‘‘name of wireload model"
```

Question 8: What do you understand by wireload **mode**? (Not wireload **model**)

Setting Operating Conditions

Set of operating conditions defined in the library specify the process, temperature, voltage and RC tree values. These values are used during synthesis and timing analysis of the circuit.

Question 9: What is the default operating condition in tcbn90gtc.lib?

Explore the target directory for tcbn90gtc.lib and open another lib file. Read the operating conditions in that file and fill in the following table:

Table: Operating conditions

Library	Condition	Process	Temperature	Voltage	tree_type
tcbn90gtc.lib	NCCOM	1	25 C	1.0 V	balanced_tree

Set the operating condition of the design using the following command.

```
set_operating_conditions ’’OPERATINGCONDITION’’
```

Question 10: Why do we need to specify the operating condition?

Modeling Clock

The design can be constrained by a clock. A clock can be created by the following command.

```
create_clock -name "clk" -period 5 -waveform { 0 2.5 } { clk }
```

False Path

The reset pin doesn't contribute to the signal path and can produce false timing analysis results. It's important to ignore the timing path originating from reset pins. For that reason we need to use the

following command.

```
set_false_path -from [get_port rst_n]
```

4.3 Synthesis and Analysis

Mapping

The technology independent design is mapped to technology dependent library cells using the command:

```
compile -map_effort medium
```

Save Design

For saving the design you can use this command:

```
write -hierarchy -format ddc -output ''OUTPUT_FILE_PATH''
```

After saving a design, you can load it using `read` command.

For saving the design in the Verilog netlist format use this command:

```
write -hierarchy -format verilog -output ``OUTPUT_FILE_PATH''
```

Generate Reports

Now that the design has been mapped to standard cells, it is the time to generate reports. You need to generate the following reports. The ">>" operator appends all the reports in one file.

1. Constraints
   `report_constraints > REPORT_PATH`

2. Cells
   `report_cell > REPORT_PATH`

3. Area
   `report_area > REPORT_PATH`

4. Timing
   `report_timing > REPORT_PATH`

5. Power
   `report_power > REPORT_PATH`

5 Experiments:

In this part, you have to synthesize all three discussed FIR filter architectures.
You can create a synthesis script for each design by writing all commands in a file `./syn/dc_synthesis.tcl` to automate the synthesis process and execute it by running the following command:

```
dc_shell> source ./syn/dc_synthesis.tcl
```

You can find a skeleton for such script in the `syn/scr` directory in your lab files. Note that the script is not completed, and it is provided as a guidance for you to complete it.

5.1 Fully Parallel FIR Filter:

Do the following experiments for all 3 designs located in:

1. `~/IL2225_labs/lab1/parallel_fir`
2. `~/IL2225_labs/lab1/serial_fir`
3. `~/IL2225_labs/lab1/partially_parallel_Pfir`

Start-up the DC form each of the 1- 3 design folders.
Analyze all of the files located in the rtl folder and *Elaborate* the top level module.

Question 11: How many memory elements (latches and flip-flops) are there in the design?

Architecture	Parallel FIR	Partially Parallel FIR	Serial FIR
Memory elements			

Hint! You can see the number of flops and latches used in the command line during the synthesis process.

In the logical hierarchal window of design vision GUI, right click on FIR top level, then click schematic view to see the schematic generated by the design vision.

Set the design constraints as explained in the previous section and compile the design with the clock period of 5ns. Then save the design and generates the reports.
To display critical paths in the schematic view:

1. Choose, *Schematic > New Design Schematic View.* Make sure the schematic view is the active view.
2. Select the critical paths by choosing, *Select > Paths From/Through/To*, and then clicking OK in the Select Paths dialog box. The default options are set to select the path or paths with the worst negative slack in the design.
3. Highlight the selected paths by choosing, *Highlight > Selected.*
4. If you can not see the critical path, try to ungroup the blocks of the design hierarchy until you do.

5. Use the magnification tools and the pointer to identify the start points and endpoints of the critical path.
6. Compare the start/endpoint with the one in the timing report.

You can also run the command `report_timing` to get a report of the worst slack path.

Question 12: Does the critical path cross the boundaries of the modules? Write the start point and end point of the critical path in the table.

Architecture	Parallel FIR	Partially Parallel FIR	Serial FIR
Start point			
End point			

Question 13: Does the start and the end points of the critical path match with what is reported in the timing report?

Click on the `arithmetic_unit` on the schematic to see inside of the module. Highlight the critical path of this module.

Question 14: How many MACs are there in the critical path of the `arithmetic_unit`? Justify the answer.

Architecture	Parallel FIR	Partially Parallel FIR	Serial FIR
Nr. of MACs in critical path			

Question 15: Which designs have the shortest and the longest critical paths? Does the critical path match with the one that you calculated in questions 2, 3 and 5?

Clock period = 5ns

Architecture	Parallel FIR	Partially Parallel FIR	Serial FIR
Area			
Timing Slack			

Question 16: Is the performance requirement met in all designs? Which design is the smallest and which one is the largest (area)? Why?

Change the clock period and recompile the designs.

Clock period = 2.5 ns

Architecture	Parallel FIR	Partially Parallel FIR	Serial FIR
Area			
Timing Slack			

Question 17: Is the timing requirement met or violated for each of the designs? Why?

6 Gate Level Simulation

In this section of the lab, your task is to do gate-level simulation using modelsim for your **partially parallel FIR filter**. The purpose of the gate-level simulation is to verify the functionality of the gate-level netlist. You should show your working gate-level simulations to the lab assistant. For gate level simulation you need:

1. The Verilog netlist of your design (./db/partially_parallel_FIR.v)
2. types_and_constants.vhd package
3. Gate level netlist testbench. It is advisable to have a separate testbench for the gate level netlist. The synthesis tool might change the name of tour top level entity, for example if you have generics your design will be appended with the size of the generic, e.g. serial_FIR_WIDTH_5.
4. tcbn90g.v library file, you can copy or symlink it from the following directory to your db directory. For example:
   `ln -s /afs/it.kth.se/pkg/synopsys/extra_libraries/standard_cell/TSMC/tcbn90g_110a/Front_End/verilog/tcbn90g_110a/tcbn90g.v ./db/`

For gate-level simulation you need to consider that:

- You can compile your designs using modelsim by running the following commands:

```
partially_parallel_FIR$> vlog ./db/tcbn90g.v ./db/partially_parallel_FIR.v

partially_parallel_FIR$> vcom ./rtl/types_and_constants.vhd ./tb/partially_parallel_tb.vhd
```

- Finally you can launch the simulation by running:

```
partially_parallel_FIR$> vsim work.partially_parallel_tb -t 5us
```

Remember that when you do gate level simulations, you should disable the optimizations in your simulator. You can do that in vsim by ticking the *disable optimization* box. In newer versions of vsim this option has been discontinued and will prompt an error. In this case you can disable the optimizations for inside the optimization options, by setting optimization level to -O0 and enabling *full visibility to all modules* in the visibility tab.

Similarly to the RTL simulations, you can create a script to automate this process.

Question 17: Does your gate level simulation match the RTL simulation? Be prepared to show the simulation during the Lab check.

Optional: Complete the synthesis script that you use to automate the synthesis process of your design. You can find a template in `./syn/scr/dc_synthesis.tcl`