

Lab Manual

Laboratory
Synthesis of Digital Systems

Prof. Dr.-Ing. Ulf Schlichtmann

April 30, 2014

Institute for Electronic Design Automation
Technische Universität München
Arcisstr. 21, 80333 München
Tel.: (089) 289 23666

Synthesis of Digital Systems - Laboratory

Institute for Electronic Design Automation
Technische Universität München

Contents

1. Checklist for Completion of the Laboratory	1
2. Getting Started with Vivado-HLS Tools	3
3. Lab B: High Level Synthesis using Vivado-HLS	4
3.1. Design Preparation	4
3.2. C-code Validation	5
3.3. Solution1: HDL Generation from C-code	5
3.4. Solution2: Loop Optimization Directives	8
3.5. Solution3: Loop Pipelining for Outer Loops	12
3.6. Solution4: Dataflow Optimization for the Complete Design	14
3.7. Solution5: Flattening the Sub-blocks to Top-level	16
3.8. RTL Verification	16
3.9. IP Creation	17
Bibliography	19

List of Figures

3.1. Solution Configuration window.	5
3.2. Vivado_HLS - Description of various panels.	6
3.3. Analysis report for solution1.	7
3.4. Analysis perspective for dct (like figure144).	7
3.5. Performance for dct 2d.	8
3.6. Nested loops of dct_1d.	9
3.7. Directive panel for solution2.	10
3.8. Synthesis report for solution2.	11
3.9. Data dependencies preventing the flattening of nested loops.	12
3.10. Directive tab for solution3.	13
3.11. Report comparison for solution3 vs solution2.	14
3.12. Performance bottleneck because of IO operations.	14
3.13. Directives for solution4.	15
3.14. Analysis perspective (MH and PP) for solution4.	16
3.15. Directives for solution5.	17
3.16. Cosimulation report for dct.	18

1. Checklist for Completion of the Laboratory

1. Evaluation

- Report: This laboratory consists of four parts. Each lab has a set of questions in the last *Questionnaire* section. At the end of each lab, you must submit a report with the answers to those questions, according to respective deadlines.
- Project code: Please insert your name and account as comments at the beginning of the source files. For submission, you must zip the lab assignment and upload it on the server. Only the code inside these directories will be evaluated. For evaluation, you must guarantee that the submitted project code can be directly imported into the Xilinx tools and tested.
- Written exam: 90 min. 1/3 of the points for the final exams will be related to Lab content.

2. Grade

This laboratory makes 50% of the Synthesis of Digital Systems (SDS) course. 25% of the final grade will be on code submission and reports, 1/2 part for code and other 1/2 part for the reports. Final written exam will be 75% of the grade, 1/3 of the points will be related to Lab content.

3. The working environment and tools

For login at the institute the user name is `your_lrz_account` (e.g. `ne42zup`) while the password is the same as that of the mytum email box. For remote login, the server domain name is `sdsX.regent.e-technik.tu-muenchen.de` where `X` is a number between 1 and 9. Eventually you need to install the vpn client as described here: <http://www.lrz.de/services/netz/mobil/vpn/>. If you use `ssh`, the full command is:

```
ssh -X your_lrz_account@sdsX.regent.e-technik.tu-muenchen.de
```

The lab working directory is:

```
/usr/local/labs/SDS/current/your_lrz_account
```

Code Submission must be present in respective directories (LabA, LabB, LabC, and LabD) under,

```
/usr/local/labs/SDS/current/<your_lrz_account>/Lab<X>
```

4. Materials

- Link to information on the ZedBoard: <http://www.zedboard.com>.
- Schedule, lab manual, project material and lecture notes are or will be available on Moodle.

1. Checklist for Completion of the Laboratory

5. Contact

- In case you encounter problems while doing the laboratory, please write an email (always with your lrz login on title) to the following address:
eda-sds-support@lists.lrz-muenchen.de

Please attach the relevant files that your question refers to. Always use this email as the first option in case that you want to get an answer to the problem that you can not solve after long-time thinking by yourself.

- Or you can come to the tutor hour on Wednesday, 16:45-18:15, in room 2909.

Tutor: Benjamin Bordes
benjamin.bordes@tum.de

- Or contact the lab supervisor:

M.Sc. Munish Jassi
Room 2909
Tel.: (089) - 289 23651
E-Mail: munish.jassi@tum.de

2. Getting Started with Vivado-HLS Tools

1. Vivado-HLS, 2013.3 version
2. Copy getting started project from <project docs>/GettingStartedHLS to <work dir>
 - <work dir>: /usr/local/labs/SDS/current/<your_lrz_account>/
 - <project docs>: /usr/local/labs/SDS/current/<your_lrz_account>/project_documents
3. To load environmental settings, *\$module load xilinx/vivado/2013.3*
4. To start tool, Vivado-HLS: `$vivado_hls -p fir_prj`.

If you are not able to invoke `vivado_hls_v2013.3`, try directly invoking the binary as `/nfs/-tools/xilinx/Vivado_HLS/2013.3/bin/vivado_hls -p fir_prj` . Get familiar with the Vivado-HLS options.
5. Click C-Simulation. It will end with Pass! message on console
6. Click C-Synthesis. Analysis the final synthesis report.
7. Click Analysis perspective. Get familiar with options and reports.

3. Lab B: High Level Synthesis using Vivado-HLS

In this lab we will learn the principles of High Level Synthesis (HLS). We will carry on hardware synthesis of a DCT-2D function and optimize the hardware synthesis by progressively adding optimization directives. As mentioned in the introduction, the design target for this project is to achieve a throughput of less than 300 clock cycles.

Here are the steps we will follow for this lab,

1. Design preparation
2. C-code validation
3. Solution1: HDL generation from C-code
4. Solution2: Loop optimization directives
5. Solution3: Loop pipelining for outer loops
6. Solution4: Dataflow optimization for the complete design
7. Solution5: Flattening the sub-blocks to top-level
8. RTL verification
9. IP creation

Tasks:

3.1. Design Preparation

1. Set up the environment and invoke Vivado-HLS tool on the terminal. Now, open a new terminal, go to <work dir >directory, make a new directory LabB: `$mkdir LabB`, and go to that directory.
2. Once inside LabB, type `$source /nfs/tools/xilinx/Vivado/2013.3/settings64.sh` or `$module load xilinx/vivado/2013.3` and then `$vivado_hls` to invoke Vivado-HLS.
3. Click *Create New Project* and set Project name: *"DCT_HLS_Project"* and Click *Next*. In the *Design Specification* window set Top Function as *"dct"*. In *Design Files* click *Add Files...* and browse to *dct.cpp* in the <project docs >/LabB and click *OK* to add it into the project. Then click *Next*.
4. In the *TestBench* window, click *Add Files...* and browse to add *dct_test.cpp*, *in.dat*, and *out.golder.dat* files to the project, click *OK* to add these to project. Click *Next*.
5. In the *Solution Configuration* window, set Solution Name: *solution1*; Clock Period: 8 (in ns); Uncertainty: <keep it blank> (it will take the default value of 12.5% of clock period). Click ... (browse) on *Part Selection*. In the pop-up select device *xc7z020clg484-1*. To filter the device, mark these filters, Product Category: General Purpose; Family:zynq; Sub-Family:zynq; Package: clg484; Speed Grade:-1. Click *OK*.

3. Lab B: High Level Synthesis using Vivado-HLS

6. *Solution Configuration* window should now look like in Figure 3.1. Click *Finish* to create the project.
7. A new project *DCT_HLS_Project* will be created in the Vivado_HLS with all the files for sources and test benches included (Figure 3.2).

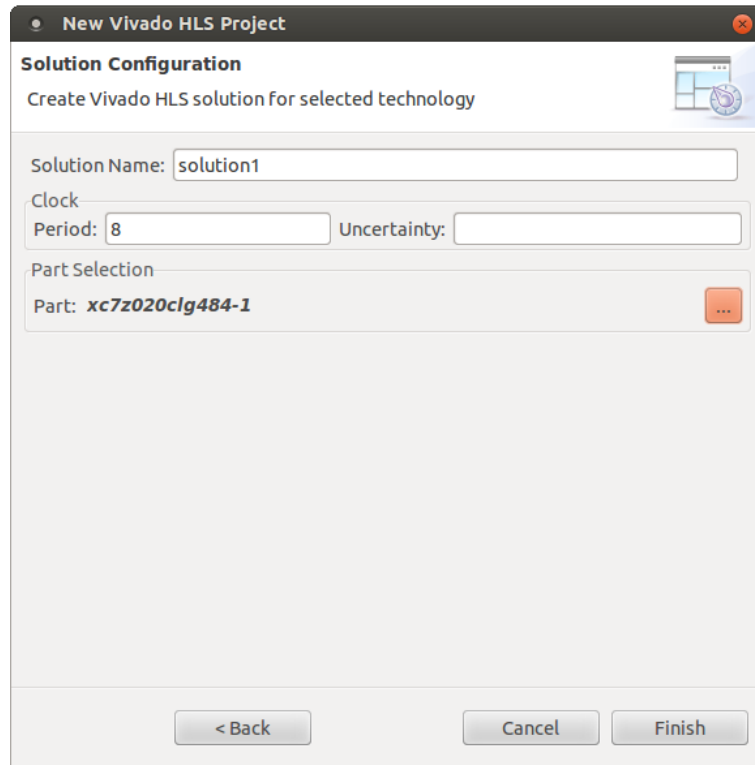


Figure 3.1.: Solution Configuration window.

3.2. C-code Validation

First step is to validate the C-code of *dct.cpp*. The file *dct_test.cpp* contains the test bench for *dct.cpp*. This test bench calls the *dct(input[],output[])* function and passes the valves read from *in.dat*. The output values generated by the *dct* function are written to *out.dat*, which is then compared to the golden result values in *out.golden.dat* for validation of final results.

1. For validation, click *Run C Simulation* from the toolbar or select *Project>Run C Simulation*. Click *OK* (we don't need any option). On successful validation, "Test passed!" Message will appear on the Console.

3.3. Solution1: HDL Generation from C-code

1. To synthesize the C-code, click *Run C Synthesis*, or select *Solution>Run C Synthesis>Active Solution*. Synthesis should go through without errors and synthesis report will open automatically at the end of the synthesis.

3. Lab B: High Level Synthesis using Vivado-HLS

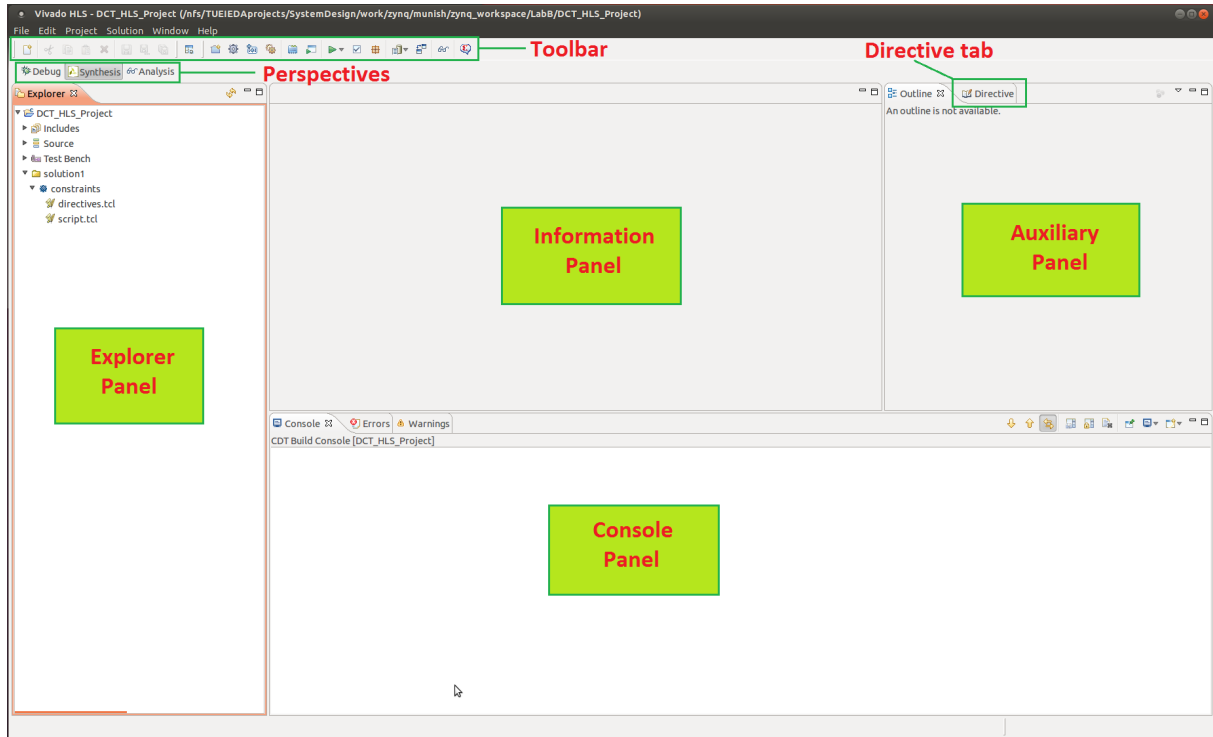


Figure 3.2.: Vivado_HLS - Description of various panels.

Figure 3.3 shows the synthesis report for this run. We can see that time period constraint of 8ns (defined during project set up) has been met, but the latency of execution is 3959 cycles and the interval is one cycle more, i.e. 3960 cycles. This is quite far from our target of having less than 300 cycles of interval time. Expanding *Detail* below *Summary*, shows that the top level instance takes the most clock-cycles and has a latency of 3668, while the read and write loops take 144 cycles each.

In this synthesis of C-code, the tool do not apply any optimization directives. We can see that interval time for the design is one cycle more than its latency. This means that the design is not pipelined. That is, data-read, DCT & data-write are executed sequentially.

To do the performance analysis of this application we will use *Analysis perspective*. Open the *Analysis perspective* by clicking *Analysis* on top-left of Vivado-HLS window. Analysis perspective is shown in Figure 3.4.

There are five available tabs to analyze the performances and areas. Module Hierarchy gives the summary of various parameters at different hierarchical levels. On the right side is the interactive graphical tool to trace various hierarchical performances of the application. It has two tabs, *Performance* and *Resource*. The *Performance Profile* gives tabular performance breakdown according to various instances and loops in the design. *Resource Profile* gives the breakdown of hardware components usage (memories, instances, IOs, registers etc.). This interactive analysis perspective gives a powerful way to analyze the performance and resource bottlenecks in the design.

How to use Analysis perspective:

- Yellow blocks, represent functional blocks which can further be expanded to elementary

3. Lab B: High Level Synthesis using Vivado-HLS

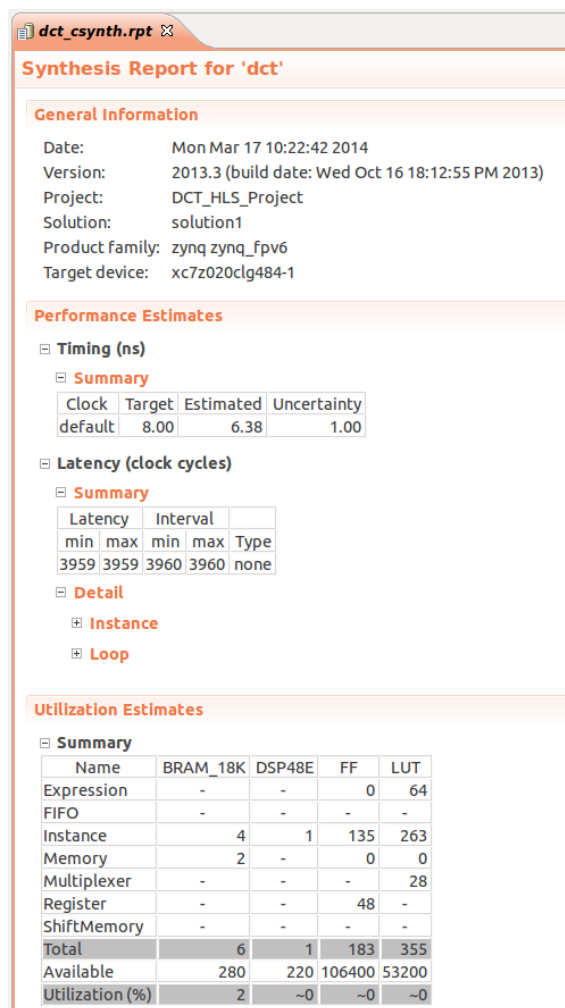


Figure 3.3.: Analysis report for solution1.

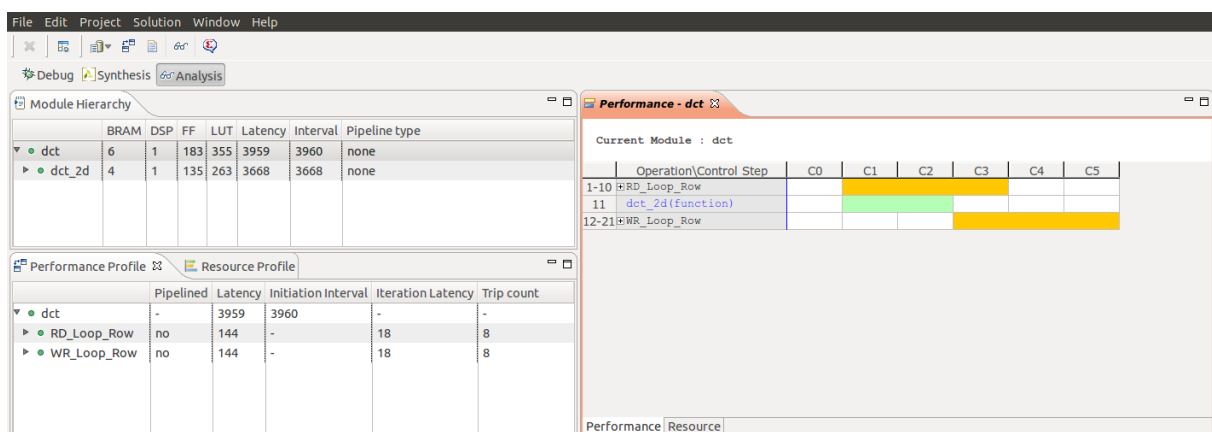


Figure 3.4.: Analysis perspective for dct (like figure144).

operations (on clicking on yellow blocks).

- Green blocks, represent sub-blocks at lower level of hierarchy.
- Violet blocks, represent the elementary operations.
- Keeping the cursor on these blocks shows the performance summary.
- By right-clicking on the block and selecting *Goto Source* shows its implementation in the source code.

Performance analysis:

- For the implementation of *solution1* in the Analysis perspective, we can notice RD_Loop_Row and WR_Loop_Row both have latency of 144 clock-cycles. Inside *dct > dct 2d*, Row_DCT_Loop and ColDCT_Loop have latency of 1688 clock-cycles each (Figure 3.5).
- In the Module Hierarchy, latency for *dct_2d* is 3668 and for *dct_1d* is 209.

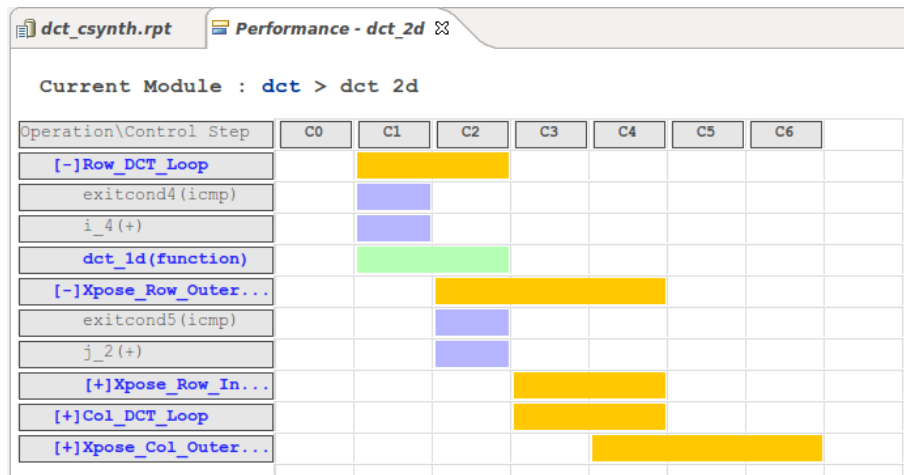


Figure 3.5.: Performance for dct 2d.

- Function *dct_1d* is called in both Row_DCT_Loop and ColDCT_Loop.
- Expanding *dct_1d*, we see there are nested loops of DCT_Outer_Loop and DCT_Inner_Loop (Figure 3.6)
- One possible optimization which we can do is pipelining in these nested loops, which will eventually improve the performance of the function.
- So, as next step we will create new *solution2*, and add loop optimization directives.

Go back to the *Synthesis perspective* for the next optimization step.

3.4. Solution2: Loop Optimization Directives

We found that one way of optimization is to do loop pipelining in the nested loops. This is done by adding directives to the synthesis flow.

1. In the *Synthesis perspective*, select Project>New Solution. In the window set Solution Name: "solution2", Period: 8; Uncertainty: <blank>; select both the Options: *Copy existing directives* and *Copy existing custom constraints to solution1*. Click *Finish*.

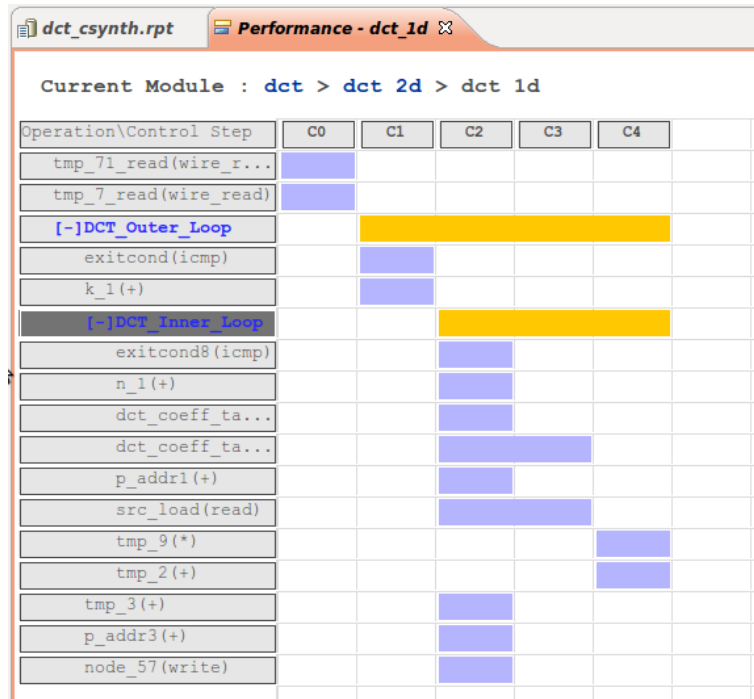


Figure 3.6.: Nested loops of dct_1d.

2. Make sure that *solution2* is active (bold). If not, right-click on *solution2* and select *Set Active Solution*.
3. Under *Source* in *Explorer* panel, double-click *dct.cpp* to make it visible on *Information* panel.
4. In the *Auxiliary* panel, activate *Directive* tab. Here you will notice all the functions and loops which we have seen in the *Analysis perspective* in the last section.
5. To add *loop pipelining* directive to DCT_Inner_Loop, of function *dct_1d* follow these steps,
 - a) Right-click on DCT_Inner_Loop and select *Insert Directive..*
 - b) In the *Directive Editor* window, select *Directive: PIPELINE*, and click *OK*.
6. Repeat Step-5 for the following functions,
 - a) Xpose_Row_Inner_Loop and Xpose_Col_Inner_Loop functions in *dct2d*.
 - b) RD_Loop_Col function in *read_data*.
 - c) WR_Loop_Col function in *write_data*.

The *Directive* panel will now look like in Figure 3.7.

7. Synthesize the design by clicking *C-Synthesis*, or selecting *Project>Run C Synthesis>Active Solution*.
8. In the log report search for **XFORM-541** messages, which logs about the flattening of nested loops. The synthesis report (Figure 3.8) shows that clock timing is still met and the latency has gone down to 1982 clock-cycles.
9. To compare the results for this run with the previous run, select *Project>Compare Reports...* . Select the solutions and click *Add*. Click *OK* to finish.

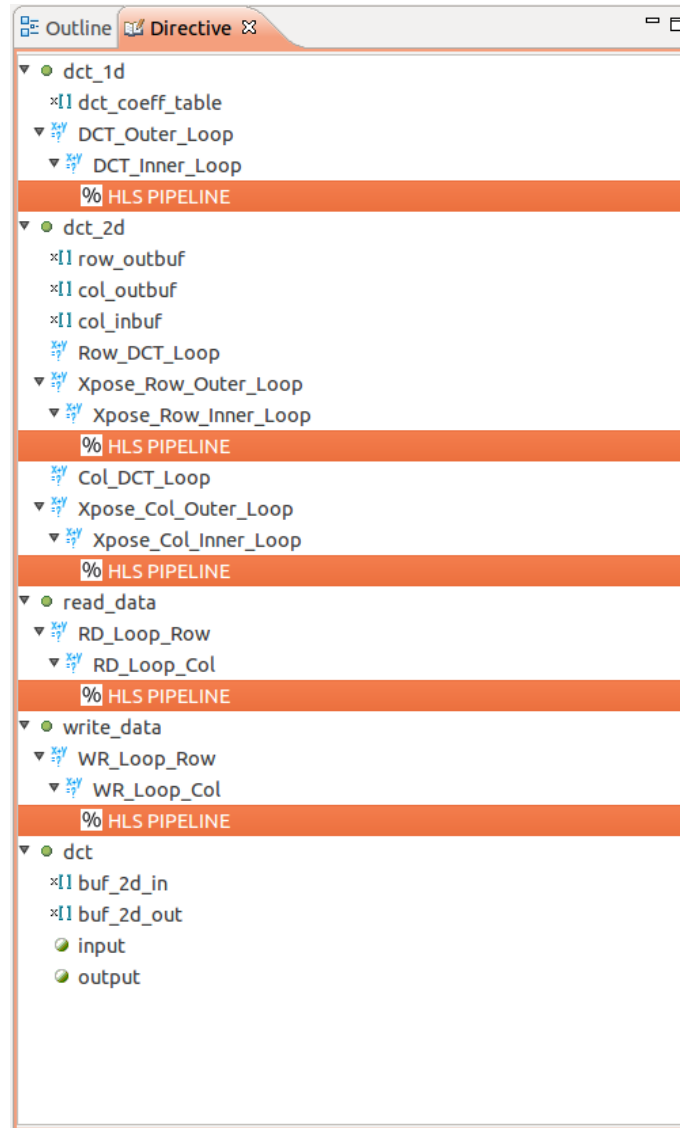


Figure 3.7.: Directive panel for solution2.

In the comparison report we see that the latencies have almost halved and the FF (Flip flops) and LUT usage has increased. This is because of additional FF and logic required for pipelining the loops.

Select *Analysis perspective* to do performance analysis. Here are the findings:

- We see that latencies for read and write have reduced to 65 cycles from earlier 144 cycles.
- In `dct_2d`, latencies for `Row_DCT_Loop` and `Col_DCT_Loop` have reduced to 856 each.
- Going inside `dct_2d` to `dct_1d`, gives some more insight into performance bottlenecks.
- We see that `DCT_Inner_Loop` is still not flattened. To analyze this, right-click node `<>`(write) and select *Goto Source*.
- In the C Source, notice that write to `dst[k]` statement is outside the inner loop and is

3. Lab B: High Level Synthesis using Vivado-HLS



Figure 3.8.: Synthesis report for solution2.

preventing the DCT_Inner_Loop to flatten (Figure 3.9)

- The next potential performance gain we can get if we also flatten the DCT_Outer_Loop.

10. Return to the *Synthesis perspective* for the next optimization step.

Questions:

1. What is C-code validation and C-code synthesis in Vivado-HLS?
2. What are *Directives* in *Vivado-HLS*?

3. Lab B: High Level Synthesis using Vivado-HLS

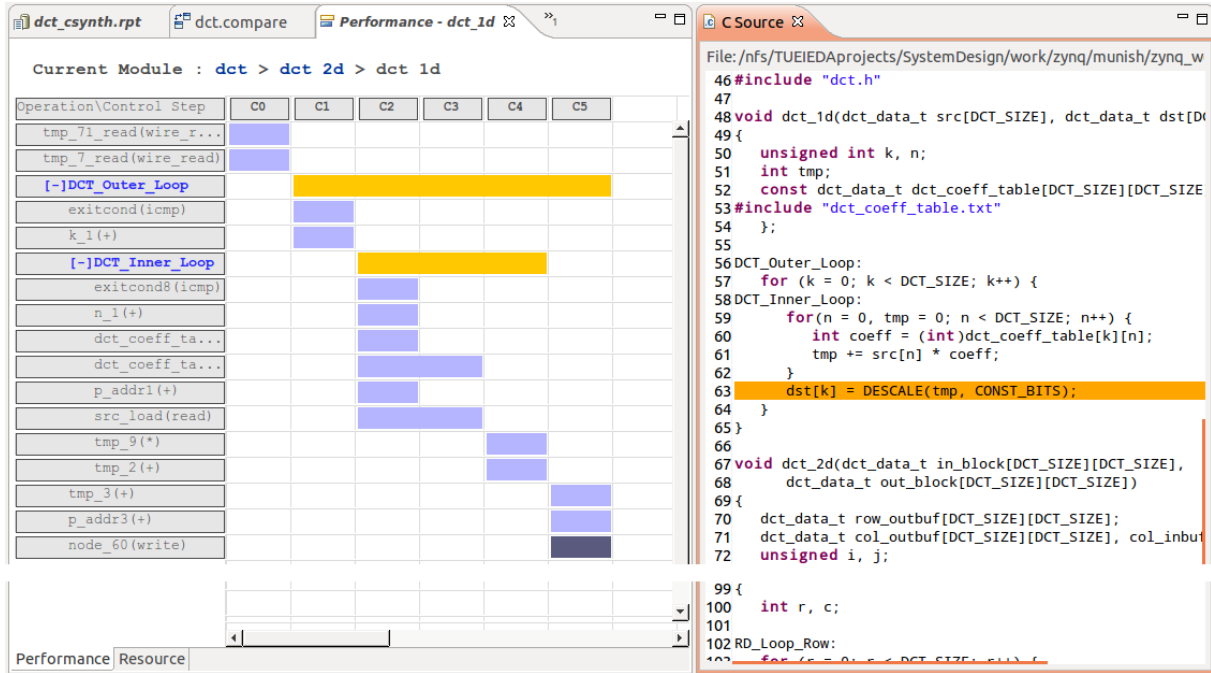


Figure 3.9.: Data dependencies preventing the flattening of nested loops.

3. How does *Loop Optimization (loop unrolling)* work? (ref. to SDS Theory lectures)
4. Which are the various *Directives* available in *Vivado-HLS* and give one liner description for each? (ref. to Vivado-HLS User Manual, ug902)

3.5. Solution3: Loop Pipelining for Outer Loops

1. Create a new solution *solution3* and make sure that solution3 is active.
2. Remove the previously added PIPELINE directive to DCT_Inner_Loop of dct_1d by right-clicking and selecting *Remove Directive*.
3. Select DCT_Outer_loop and add PIPELINE directive.
4. Directives tab should now look like in Figure 3.10.
5. *C Synthesize* the design and open report compare perspective (Figure 3.11).

In this solution, the latency has almost halved to 894 clock-cycles (these numbers can vary slightly across different runs) and number of FF has increased to more than double. Since dct_1d is called in multiple sub-blocks, we have gained significant improvements.

Select the *Analysis perspective* to do the design analysis. Here are the findings,

- Expanding the read and write sub-blocks, shows a quite vertical stack of operations. This means that these blocks have been fairly optimized and are pipelined.
- Going into *dct>dct_2d>dct_1d* and expanding DCT_Outer_Loop shows a spread of operations, which goes from the top left corner to the bottom right corner.
- This spread can be because of data-dependences or from data bottleneck at the IOs.

3. Lab B: High Level Synthesis using Vivado-HLS

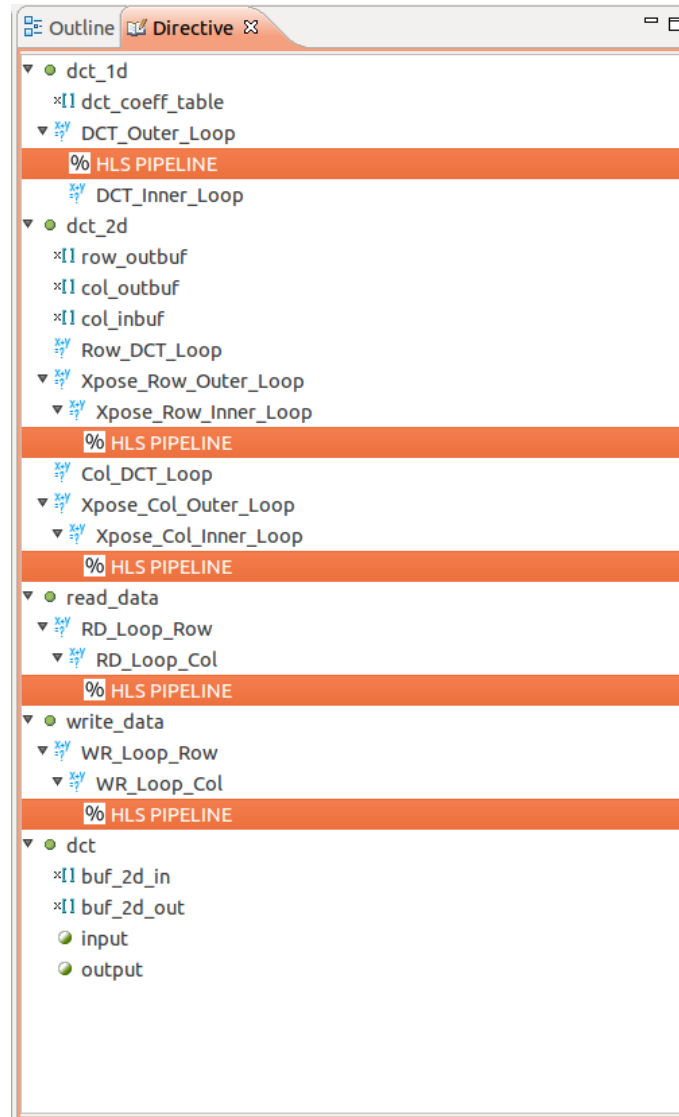


Figure 3.10.: Directive tab for solution3.

- Now activate Resource panel, and expand the Memory Ports. Right-click on one of the *read block* for *src* and select *Goto Source*.
- Here we notice that eight read operations are stacked one after the other. This is because BRAM allows two read operations in one cycle.
- Since the same BRAM is used to read the data serially, the read operations from BRAM are delaying the execution of operations (Figure 3.12).

Till now we have optimized the design at a specific sub-block level. In the next step, we will enable these individual sub-blocks to operate in parallel, to do the global optimizations.

3. Lab B: High Level Synthesis using Vivado-HLS

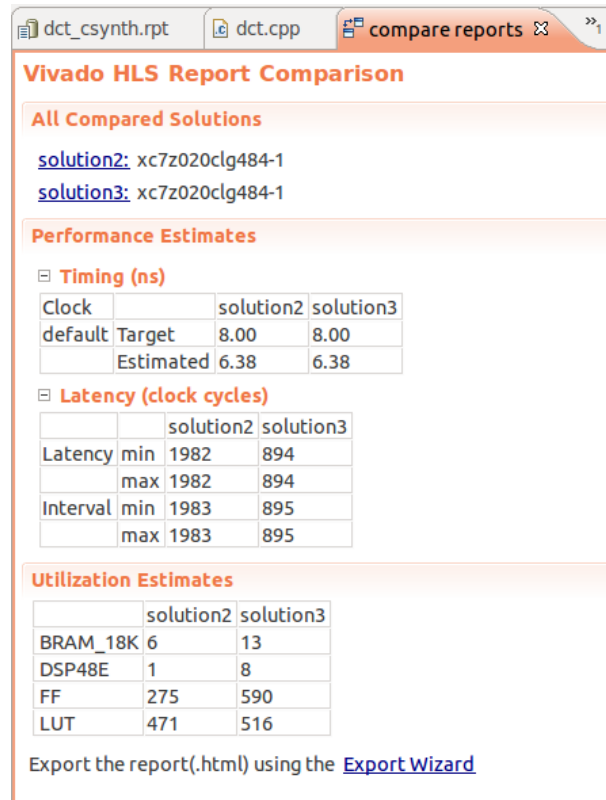


Figure 3.11.: Report comparison for solution3 vs solution2.

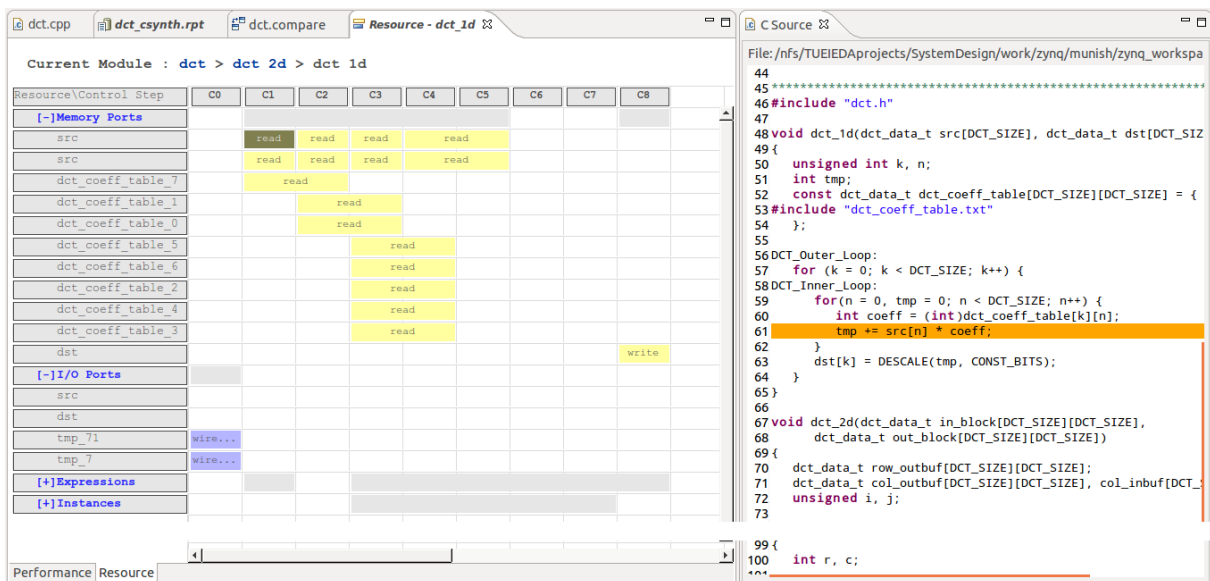


Figure 3.12.: Performance bottleneck because of IO operations.

3.6. Solution4: Dataflow Optimization for the Complete Design

1. Create new solution: *solution4*.

3. Lab B: High Level Synthesis using Vivado-HLS

2. Activate the *Directive* tab, right-click *dct* function and select *Add Directive*..
3. In the *Directive Editor* select *Directive: DATAFLOW* and click *OK*.
4. Figure 3.13 shows the directive tab for the current solution.

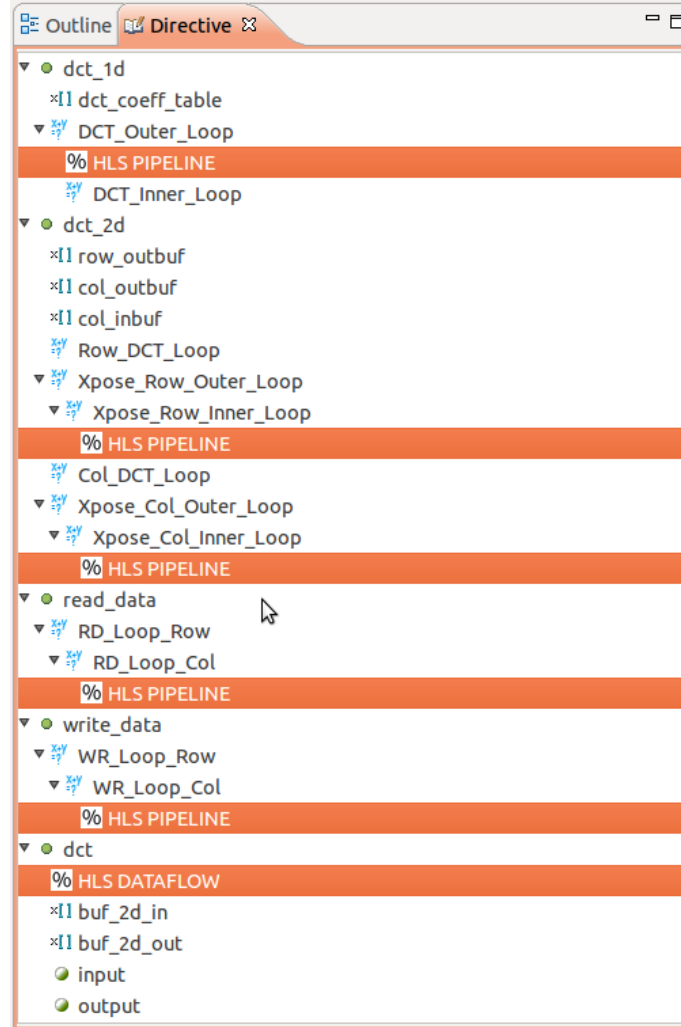


Figure 3.13.: Directives for solution4.

5. *C Synthesize* the new solution and open the compare reports.

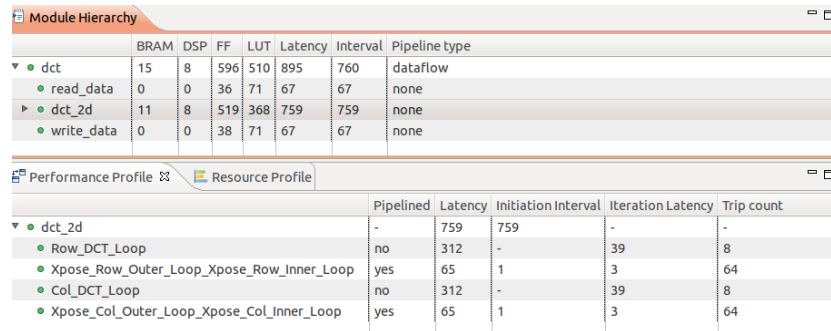
In the comparison report we see that the latency has not improved much, this is because as we have seen earlier the design is bottlenecked by IOs. But due to parallel execution of sub-blocks, the interval has been reduced to 760 clock-cycles.

Select the *Analysis perspective* to do further performance analysis. Here are the findings,

- In the *Module Hierarchy* tab, we see that *dct_2d* consumes most of the cycles.
- Since *dct_2d* has same number of cycles as the interval for *dct*, rest of the sub-blocks are operating in parallel. So *dct_2d* determines the interval of the design.
- In the *Performance* profile for *dct_2d*, we notice that latency and interval for *dct_2d* are equal. This means that none of the operations in *dct_2d* are parallelized (Figure 3.14).

3. Lab B: High Level Synthesis using Vivado-HLS

This is because, dataflow optimization only operates on top-level loops and functions, so it has restricted the parallelization only at the top-level loops and functions. One way to overcome this is to push all the loops from lower-level to top-level. Other alternative is to do the pipelining of the entire design, which can lead to large increase in the design area.



The screenshot displays two panels from the Vivado-HLS analysis perspective for solution4. The top panel, 'Module Hierarchy', shows a tree structure with 'dct' expanded to show 'read_data', 'dct_2d', and 'write_data'. The bottom panel, 'Performance Profile', shows a table of performance metrics for the expanded 'dct_2d' hierarchy.

	BRAM	DSP	FF	LUT	Latency	Interval	Pipeline type
dct	15	8	596	510	895	760	dataflow
read_data	0	0	36	71	67	67	none
dct_2d	11	8	519	368	759	759	none
write_data	0	0	38	71	67	67	none

	Pipelined	Latency	Initiation Interval	Iteration Latency	Trip count
dct_2d	-	759	759	-	-
Row_DCT_Loop	no	312	-	39	8
Xpose_Row_Outer_Loop_Xpose_Row_Inner_Loop	yes	65	1	3	64
Col_DCT_Loop	no	312	-	39	8
Xpose_Col_Outer_Loop_Xpose_Col_Inner_Loop	yes	65	1	3	64

Figure 3.14.: Analysis perspective (MH and PP) for solution4.

In the next optimization step we will push the loops from `dct_2d` hierarchy to higher level to apply dataflow optimizations.

3.7. Solution5: Flattening the Sub-blocks to Top-level

1. Create a new solution *solution5* using default settings.
2. Activate the Directive tab, right-click `dct_2d` function and select *Add Directive..*
3. Select *Directive:INLINE*, keep other default settings and click *OK*.
4. Figure 3.15 shows the directive tab for the current solution.
5. *C Synthesize* the new solution and open the compare reports.

Now, we are down to an interval period of about 262 cycles with the latency of 795 cycles. For hardware usage, we see the jump on number of FF, LUTs, as well as on BRAM and DSPs. Now we have achieved the design objectives of the design. Next step is to package this IP to use it further in Vivado/PlanAhead.

Questions:

5. What were the indicators which lead to the conclusion that data dependency was blocking the loop pipelining for *Solution3*?
 6. What is *Local vs. Global optimization*? (ref. to SDS Theory lectures)
 7. What prevented the *DATAFLOW* optimization for *solution4* to improve the design?
-

3.8. RTL Verification

1. Make sure that *solution5* is active. Run RTL verification of the design we generated, by clicking *Run C/RTL Cosimulation* or select *Solution>Run C/RTL Cosimulation*.



Figure 3.15.: Directives for solution5.

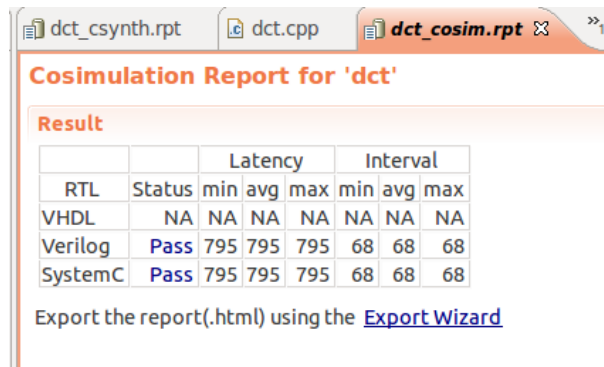
2. Keep the default settings (only *SystemC* selected in *RTL Selection*) and click *OK*.
3. At the end of the verification, you should see *Test passed!* message printed on the console and you will see the co-simulation report with status message as *Pass*.
4. Similarly, for doing the validation with Verilog/VHDL, select the appropriate option in the C/RTL Co-simulation dialog box and run the *Co-simulation*.

Figure 3.16, shows the co-simulation report for RTL verification. In the RTL verification, input vectors are generated based on the C test bench. These vectors are used in the RTL simulations and outputs are fed back to compare with C-testbench.

3.9. IP Creation

In this step, the design is exported as IP to be used in the Xilinx design flow (Vivado/PlanAhead IP-library).

3. Lab B: High Level Synthesis using Vivado-HLS



The screenshot shows the Vivado-HLS interface with three tabs: `dct_csynth.rpt`, `dct.cpp`, and `dct_cosim.rpt`. The `dct_cosim.rpt` tab is active, displaying the "Cosimulation Report for 'dct'". Under the "Result" section, there is a table with the following data:

RTL	Status	Latency			Interval		
		min	avg	max	min	avg	max
VHDL	NA	NA	NA	NA	NA	NA	NA
Verilog	Pass	795	795	795	68	68	68
SystemC	Pass	795	795	795	68	68	68

Below the table, there is a link: "Export the report(.html) using the [Export Wizard](#)".

Figure 3.16.: Cosimulation report for dct.

1. Make sure that *solution5* is active. Click *Export RTL* or select *Solution>Export RTL*.
2. In the *Export RTL* dialog, select *IP Catalog* as *Format*, keep other settings default. Then click *OK*.
3. For other export IP formats, appropriate format can be selected in the *Export RTL* dialog.

If you expand the *impl* folder under *solution5*, you will see *xilinx_com_hls_dct_1_0.zip*. This zipped file contains the exported IP for the design and can further be taken to next steps of the design flow in Vivado/PlanAhead.

With this we are now at the end of LabB section.

Questions:

-
8. Describe briefly about *Xilinx Pcore* for *IP-Creation*? (ref. Vivado-HLS on Xilinx website)
-

Bibliography

- [1] ITRS-2011, *www.itrs.net*
- [2] ANSI, IEEE Standards Board, IEEE Standard VHDL Language Reference Manual: IEEE Std 1076-1993, New York, 1988, ISBN 1559373768
- [3] Peter J. Ashenden, Designer's Guide to Vhdl, Morgan Kaufmann Publishers, 1995, ISBN 1558602704
- [4] Giovanni De Micheli, Synthesis and Optimization of Digital Circuits, McGraw-Hill Book Company, 1994, ISBN 0070163332
- [5] Jayaram Bhasker, A VHDL Primer, Prentice Hall, Englewood Cliffs, 1998, ISBN 0130965758
- [6] Zedboard, *www.zedboard.org*.
- [7] Zedboard Hardware User's Guide v1.1, Aug 2012, Digilent.
- [8] LogiCORE IP AXI Video Direct Memory Access v6.1, PG020, Dec 2013.
- [9] Xilinx, *www.xilinx.com*.
- [10] Vivado Design Suite Tutorial, UG871, June 2013, Xilinx.
- [11] OV7670 (2006): OV7670/OV7171 CMOS VGA (640x480) CAMERACHIPT M with OmniPixel Technology, OmniVision Technologies Inc.
- [12] PG, ADV7511 (2012): Low-Power HDMI Transmitter Programming Guide, Analog Devices, Inc.
- [13] Zynq-7000 All Programmable SoC TRM v1.7, UG585, Feb 2014, Xilinx.
- [14] Vivado Design Suite User Guide, High-Level Synthesis, UG902(v2013.2), June 19, 2013, Xilinx.
- [15] Introduction to Zynq-7000TM All Programmable SoC, Speedway.
- [16] Umair Razzaq, Design Space Exploration for Embedded Vision Applications on Zynq FPGA using HLS, 2013, Master Thesis, EDA-TUM.