KU LEUVEN

FACULTY OF ENGINEERING TECHNOLOGY

DE NAYER (SINT-KATELIJNE-WAVER) CAMPUS

FPGA Performance Suite

Interim Report

Dries KENNES

Promotor: Dr. ing. Kristof Van Beeck
Co-promotoren: Nicolas Huot
Jo Van Langendonck

Master Thesis submitted to obtain the degree of Master of Science in Engineering Technology: Electronics-ICT Option ICT

Academic Year 2018-2019

Contents

List of Symbols and Abbreviations

FPGA Field Programmable Gate Array 1, 3, 5, 7, 8, 10, 11

GPU Graphics Processing Unit 3

GUI Graphical User Interface 7

HDL Hardware Description Language 1, 3–5, 7, 11

IO Input and Output 4, 9

IP Intellectual Properties 3, 5

LUT Look-Up Table 1

PAR Place and Route 8, 9

VHDL Very High Speed Integrated Circuit HDL 1, 9, 13

Chapter 1

Introduction

Antwerp Space is a Belgian satellite communications company with over 70 engineers and PhDs. It was founded in 1962 as part of Bell Telephone. In 2010 it was bought by OHB SE, a European Space and Technology group, and has been called Antwerp Space since then. It's activities include commercial ground and on-board modem and RF converters, test systems and integration of on-board communication sub-systems.[1]

As a company active in the aerospace industry, Antwerp Space is interested in FPGAs for use in small batch space grade equipment. Aerospace is a difficult industry, designs and specifications have to be fixed long before any HDL is written or boards are build. In industrial applications, the exact component choice can be delayed and designs can be revised if required.

This is obviously not the case for space, which leads to extensive analysis of every component. Often critical components -like FPGAs- are chosen with huge safety margins, if that's an option. This can be really expensive and cause other problems such as power consumption. Proper benchmarking could help substantiate this choice and cut on the required safety margin. Eliminating it entirely is not possible without having a complete HDL design. Though often a good estimation of the required components of the design can be made based on project specifications.

Today the primary source of information about FPGA performance is the manufacturers themselves. Conflict of interest is obvious here as published statistics will highlight the areas of superiority over the competition. Of course these areas may not line up with the intended use case at all. On top of that, manufactures don't handle the same standards when it comes to publishing statistics, so figures like "maximal frequency" and "LUT count and size" are not really comparable on their own.

This thesis focuses on creating a methodology and implementation that can be used to create usable benchmarks to compare devices. Those benchmarks can then be used to make a better choice. The reference implementation we create will be limited to basic VHDL building blocks. Our intention

[1] `https://www.antwerpspace.be`

is to make it easy to add to, or change those designs to fit the needs of the end user to allow them to perform a detailed and targeted suite of tests.

To summarize our main research question: *What is required to create a benchmark suite that can substantiate the choice for a specific FPGA device, architecture or manufacturer based on known characteristics of the intended use case?*

The remainder of this report is structured as follows: First we review the state-of-the-art, then we report on the progress made so far, lastly we provide a revised planning.

Chapter 2

Literature Study

In this chapter we study the state-of-the-art, examine the most current industry methodology for benchmarking FPGAs and the most used provider of IP cores for those benchmarks.

2.1 Related work

Njuguna (2008) surveys a number of benchmarks and related studies. Most of them, with notable exception of OpenFPGA.org, evaluate algorithms or implementations of algorithms rather than devices. Unfortunately OpenFPGA.org no longer exists and their website is no longer available. Since this study is now 10 years old, many of referred works are even older and only apply to much smaller devices than now available.

Selvaraj et al. (2013) evaluates more recent tools, but only for using FPGAs to accelerate general computation, augmenting or replacing GPUs. The discussed tools for generation of HDL from normal source code is interesting in the context of this thesis. This approach would allow a multitude of tests to be written in more high level languages, or have them conditionally generate the required HDL.

Vansteenkiste et al. (2015) points out that there is also a significant difference between academic and commercial results. Their conclusion is that the academic state-of-the-art is (far) behind the commercial reality, especially when comparing algorithm optimizations. This is important to consider when comparing the results from this thesis to other sources, as this thesis makes use of commercial software only.

Titan (Murray et al., 2013) utilize hybrid toolchains (commercial and academic) to reach more realistic results, but they only support one vendor.Industry players such as Intel (previously Altera) and Xilinx release white papers and technical reports on their benchmark methodology and results, such as Altera (2007), Xilinx (2017). Their focus is on showing a competitive edge, in both hardware and software. This is an obvious conflict of interest. We do however think their methodology is worth examining, which we do in the next section.

2.2 Stamping Methodology

Both Intel and Xilinx use the stamping methodology as described in Altera (2008). It consists of selecting a HDL design (called cores or stamps) and instantiating it multiple times in the FPGA. This can be used to test maximal speed and maximal utilization of FPGA resources, but in reality it's more of a test of the capabilities of the design software. Since in almost all cases the use of this vendor specific design software is mandatory, it is considered part of this methodology.

Shift registers are added between the IO signals of the cores and the IO pins of the FPGA. This prevents the design from being optimized away by the synthesis tools. Care must be taken to make sure that the IO wrapping is not the critical path and does not add too much overhead. This would defeat the purpose of testing the different designs. To avoid this, the wrapping logic and cores run asynchronously on separate clocks.

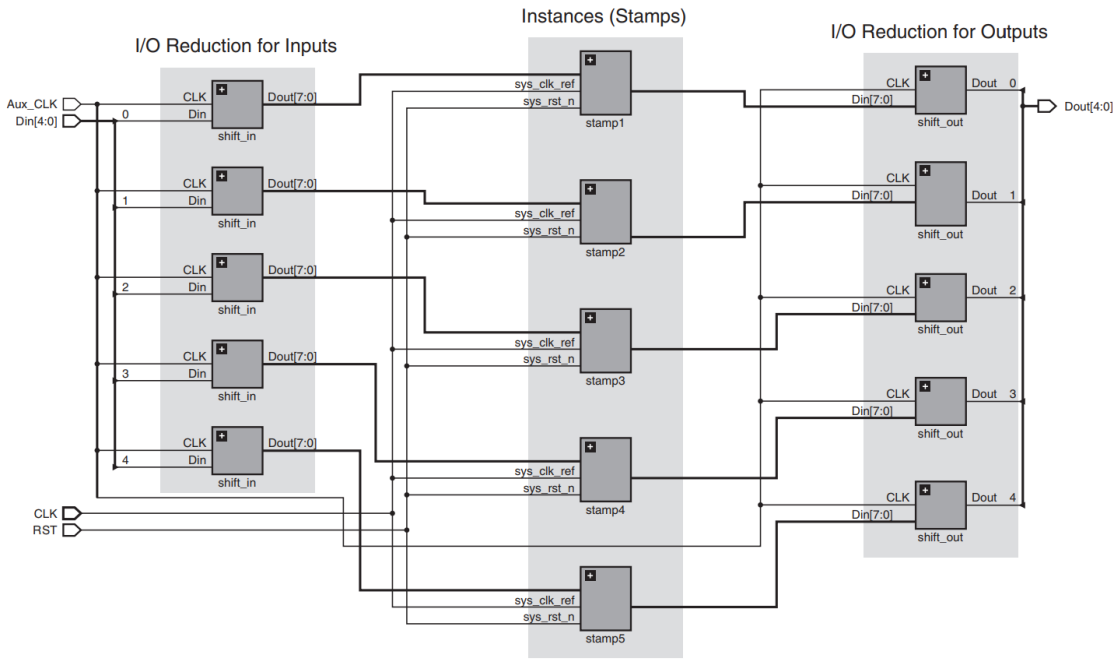


Figure 2.1: Stamping methodology as visualized by Xilinx (2017).

In most published results, the FPGA is filled to the point where the software can no longer place and route all the signals. The utilization is then measured, along with the maximal frequency. Although we think this approach may be useful to compare the efficiency of the design software, we don't think it is a good way to compare different architectures or devices. It does not provide much insight into the actual limiting factors of a device.

In the stamping the chosen design is very important. A design can, due to programmer choice or pure coincidence, be more suited for one architecture or optimization algorithm over another. This is why most published benchmark results will include a set of designs. Those designs must be sourced from somewhere, the most popular source seems to be OpenCores.

OpenCores is a website and organization which “is a community portal for professionals, amateurs, and enthusiasts interested in the field of digital design engineering. The site gives users open access to view, download, reuse, and share gateware designs. OpenCores specializes on bundles of structured files forming self-confined units, most commonly known as IP cores, coded in HDL”.[1]

Both Intel and Xilinx use designs from OpenCores in their benchmarks, as well as private designs. Those private designs are often supplied by their customers and cannot be shared publicly. This makes their results of limited value since they cannot be reproduced independently. (Altera, 2008)

The downside of stamping is that it do not often match realistic use an FPGA. A real design will never attempt to make use of all available resources, since the design software will most likely not be able to compute the placement and routing. Stamping does give a good rough indication for maximal usable capacity that can be compared with other devices or vendors, if the same parameters are used. It cannot be used to find optimal use case for one device since it offers no detailed information. Another danger is that the indented use case might not align well with chosen designs.

2.3 Conclusion

Our research leads us to believe our approach is novel. There seems to be no prior work directly addressing the same issues as we try to tackle in this thesis. Various other studies do use benchmarking but they have other goals. The results of these studies cannot be used to compare various FPGAs or multiple vendors, or are only targeted to academic toolchains. This allows us to try and fill this gap with this thesis and provide a starting point for further research or other implementations.

[1] `https://opencores.org/about/mission`

Chapter 3

Progress So Far

In this chapter we first discuss our choice of vendor, their software and the set of parameters we will test. Then we describe the algorithm and implementation of a maximal frequency finder for this software. We explain why we use different HDL designs then most benchmarks. Finally we explain how we automated the benchmark execution and how we render the results into an interactive graph.

3.1 Selected Vendor and Parameters

So far we have worked with only one vendor and software combination: Xilinx's Vivado. We chose to start with this software suite because it is one of the biggest FPGA vendors, with a market share of ~50%.[1]

Vivado can be used as a GUI program or via TCL[2] (in scripted or interpreted mode). The TCL batch mode makes it easier to setup a system that automatically benchmarks a design with a particular set of parameters.

The statistic we are looking to benchmark is the maximum frequency a design can be made to run at. In the initial implementation phase, we keep the amount of parameters to a minimum in order to reduce complexity. The parameters are: (1) the chosen design (explained below in Choosing Designs) and (2) a single generic N. For every design, N will be increased per run of the benchmark to produce a data point. This lets us plot a simple 2D graph with a line per design.

Minimizing the parameter space is important because even a simple $N \times N \mapsto N$ unsigned adder with $N = 2$ takes over 2 minutes to produce a result. Luckily, this time does not increase much with larger values for N. Any added parameter will rapidly cause the required computational power to rise out of control.

[1] According to `http://hardwarebee.com/list-fpga-companies/`

[2] Pronounced "tickle", more information on the website: `https://www.tcl-lang.org/`

Other factors which we have not considered so far:

- Vivado's "Effort Level"

  The "Effort Level" is a measure used to indicate how much time must be spend looking for the best solution to the PAR phase and other various optimizations. This can be used to squeeze the last bit of performance out of a device, but it can also drastically increase the time required to finish a single iteration.

- Extra optimization with focus on:
  - Power consumption
  - Area of the device used
  - Timing

- Speed grade of the device. The default value is used.

  The speed grade is a vendor dependent measure for the performance of a particular FPGA. In the case of Xilinx, the speed grades are a relative measure of performance within a specific family. (barriet, 2007)

- Using multiple generics

  More complex designs can be made when multiple generics are allowed. Such as a FIR filter with an N bit wide data path and M bit wide coefficients.

3.2 Finding the Maximal Frequency in Vivado

The Vivado PAR process is constraint driven, it cannot operate without setting a target clock frequency. Because of this, benchmarking a design is not as simple as running synthesis and PAR once. After synthesis a clock constraint must be added that is strict enough so the PAR process will attempt to optimize the layout, like it would in a real design. The goal is to find the maximum frequency after all.

However, if the clock frequency is too strict, the PAR process will fail and produce a path with negative slack. The negative slack is a good indication of how much extra time is required on the clock period to get PAR to succeed. However it is not the definitive answer because given a different set of constraints, PAR will make different decisions and produce a (usually) better result.

The obvious downside of this approach is that it requires multiple passes and thus is slow. We feel this approach is justified against the potentially high error that would go unnoticed by running once and then taking the slack as truth.

To get the best result, the slack is subtracted from the previously used period. This new value is used as the period for a new PAR run. This iterative loop can be repeated until the slack is zero or within an acceptable margin of error. Sometimes the algorithms will not reach this margin, but start oscillating instead. An example in tbl. 3.1.

Table 3.1 Example of an oscillating result.

Run	Period	Slack
1	1 ns	-10 ns
2	11 ns	2 ns
3	9 ns	-2 ns
4	11 ns	2 ns
5	9 ns	-2 ns
6	11 ns	2 ns

Vivado algorithms are deterministic, this means that any oscillating run can be terminated, since it will not produce productive output. The best course of action in this case seems to mark the results as an oscillation and either dismiss the result or use the worst case maximum frequency.

This entire process is done inside a single Vivado batch run, by using a complex custom TCL script that does the iterative loop until it either reaches a result within a given margin of error, produces an error, runs out of iterations or detects an oscillation. The source code of the TCL script is included in the appendix Code as `vivado_fmax.tcl`. The default values for the parameters have been chosen based on experimentation. A starting period of $1ns$, an error tolerance of 1% and an iteration limit of 10 were chosen because they produce good results. The iteration limit is almost never reached, thanks to the error tolerance and oscillation detection.

3.3 Choosing Designs

The designs required to accurately predict device performance depend on the eventual use case. This makes it hard to recommend a fixed set of designs, but there are some primitives that often return in most designs. This includes multipliers, adders and compare logic. With these building blocks it is possible to create more complex blocks such as filters.

It is also important to convince the software not to optimize the design away. Vivado (and presumably other software) can detect and remove unused gates, registers and signals and removes them. One option is to bind the IO to external pins, but this limits the design to roughly[3] the numbers of pins on the device. In low IO count designs the speed of the IO buffers can actually be the limiting factor. In higher count IO designs the routing to and from the IO buffers becomes the limiting factor. This approach should thus be avoided.

Vivado (and most other software) allows VHDL attributes to be used to inform the synthesis and PAR algorithms to not remove parts of a design. A minor downside to this approach is that it makes the VHDL code less portable, because the attributes have not been standardized. VHDL was chosen as design language over Verilog because it is the dominant language in Europe. If properly accounted for in the TCL script, Verilog designs should also be usable.

[3]The clock pin must be external on most devices, and most also need a reset pin for example.

For the initial run, a simple $N \times N \mapsto N$ unsigned adder was chosen. After verification of the methodology outlined above, a number of designs were added, as both signed and unsigned:

- Add and subtract ($N \times N \mapsto N$)
- Multiply ($N \times N \mapsto 2N$)
- Divide and remainder (and modulus) ($N \times N \mapsto N$)
- Greater or less than with "or equal" variants, equal and not equal ($N \times N \mapsto 1$)
- Shift and rotate, left and right. ($N \times log_2 N \mapsto N$)

No extra directives are added, these designs are all implemented in the naive approach. This lets the benchmark provide insight into the synthesis software's ability to optimize and correctly infer the use of dedicated hardware blocks. In our limited tests so far, Vivado did this consistently. For example, by using DSP blocks in the multiplication tests.

As previously stated, by using a single generic (N) instead of 2 separate variables, the required amount of runs is substantially reduced. A number of these operations are classically assumed to be implemented as the same logic elements. We confirmed this by looking at the outcome of a test run.

3.4 Batching Runs

To allow easy integration of other vendor's toolchains, a custom Python module was created to control the batching of different runs. The design of the module is still in progress, as the required structure will become more clear as soon as another vendor is integrated. For now, the primary purpose of the Python module is to generate a work queue, dispatch runs to processes and aggregate the results.

Initially we tried to let Vivado handle parallelization and only run one instance at the same time, but not all of the processing can be parallelized. It is thus more efficient to force Vivado to only use one thread and start up as many parallel instances as desired. Once an instance finishes a next one starts until the script runs out of work.

For each instance, the Python module stores the Vivado log output and parses it to retrieve relevant output information from the TCL script. We chose this approach because the Python script is required to manage the batch anyway, and the TCL scripts are more limited. The result aggregation happens independent of the running instances at a set interval. This reduces the loss if a batch is interrupted.

The Python code written to control Vivado and benchmark a batch of designs so far is available in the appendix under Code as `vivado.py` and `autorun.py`.

3.5 Implementation Test Run

The FPGA we used is a Kintex-7 part number XC7K410T. It has 63550 logic slices, 406720 logic cells, 508400 CLB flip-flops, 5663Kb maximum distributed RAM , 1540 DSP48 slices, 500 user

usable IO pins and 795 block RAMs of 36Kb each with ECC. In Vivado HLx 2017.1, its part name is `xc7k410tfbg900-1`. An important note is that there is no actual hardware involved in any of this thesis, Vivado is simply instructed to target this specific part number. The results for $f_{max}(N)$ are shown below in fig. 3.1 and fig. 3.2, the complete dataset and interactive graph is linked in the appendix Raw Result Data.

From this test, run we learned that only the following designs make sense to run:

- Add and subtract
- Multiply (implemented with DSP slices automatically)
- Divide
- Greater than and equal
- Shift left and right and one rotate (we chose left).

There are many more potential designs to try out, a few we believe might be interesting:

- A FIR filter
- Memory
  - Inferred and implied
  - ROM and RAM
- A state machine
- (Fixed point) Saturation logic

3.6 Displaying the results

Benchmarking produces a lot of data. In this case we are only interested in the maximal frequency. Other data, such as utilization of FPGA resources and software runtime might be of interest as well. Due to the density of the information of a large run with many designs creating a normal, static, graph is not practical. There are many frameworks to create interactive graphs, we chose plotly.js[4] because it's free and open source, powerful and easy to use.

A single custom HTML page with limited embedded JavaScript can easily interpret the aggregated results from the Python batch script. Live updating is done by polling for new results on a regular interval. The code is available in the appendix under Code as `plot.html`

3.7 Conclusion

The progress made so far indicates we are on the right track. The next step is implementing another vendor, as comparing between vendors is one of the coals of the thesis. The workload will depend on which vendor we pick (not yet determined at time of writing) and how their software fairs in an automated environment. After this, if time permits, we would like to research which parameters are worth investigating or add more complex HDL designs.

[4] `https://plot.ly/javascript/`

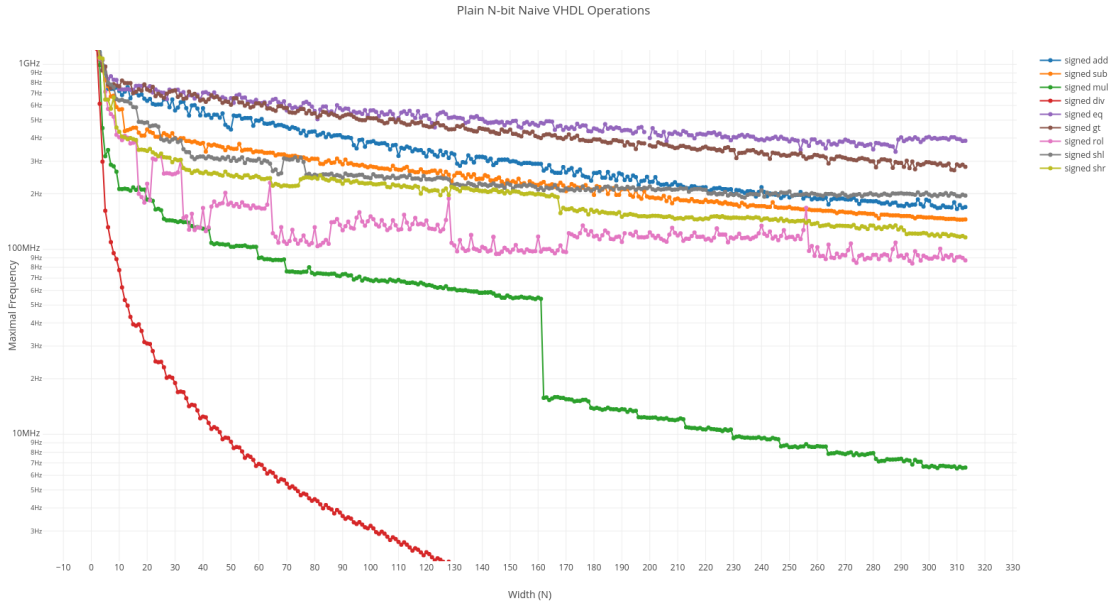


Figure 3.1: A graph of the results produced by the Implementation Test Run. For clarity the designs that share the same outcome have already been removed. An interactive version of this graph is linked in appendix Raw Result Data.

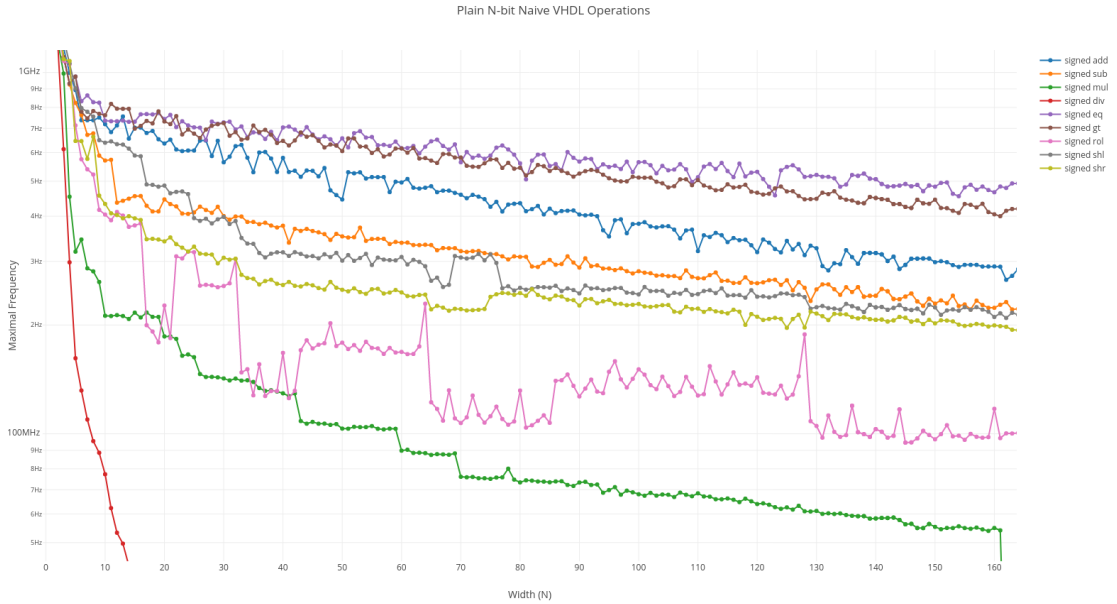


Figure 3.2: The same graph, but zoomed in.

Chapter 4

Planning

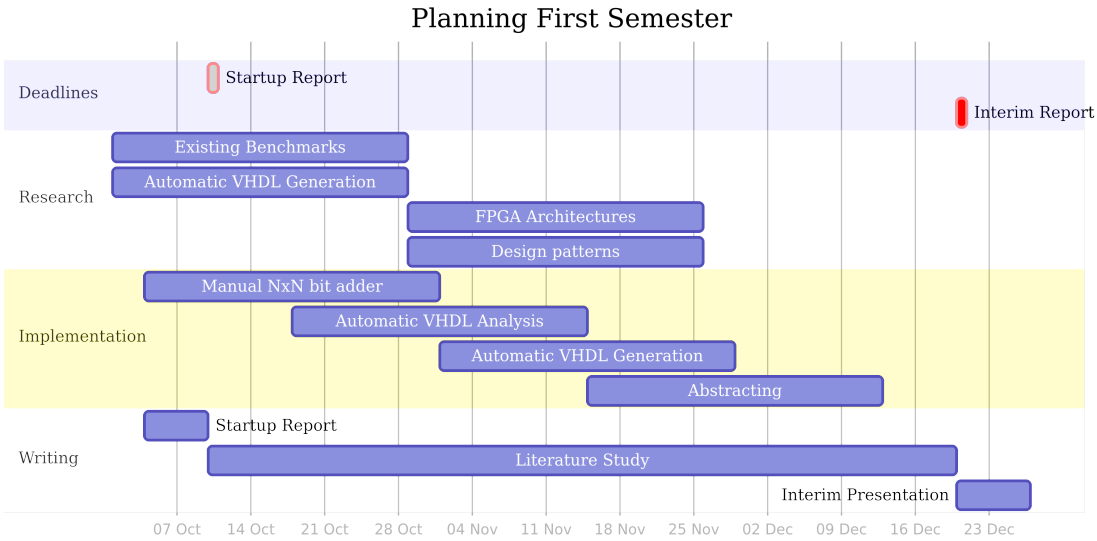
No plan survives contact with the enemy. (von Moltke, 1900, pp. 33–40) This apparently also applies to this thesis, where the enemy is time. In this case, I suspect this is primarily due to my underestimation of how long writing takes compared to implementing and research.

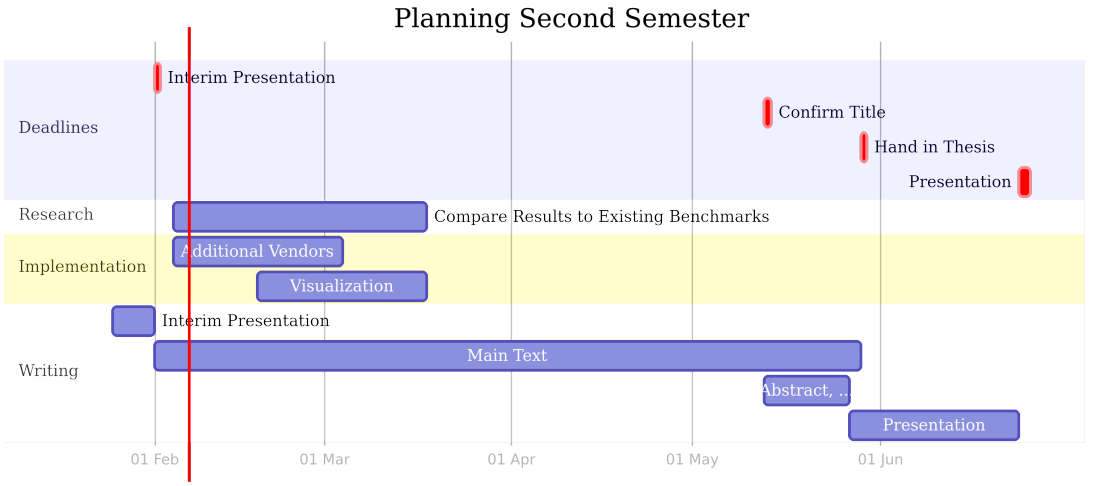
I have included the original planning and the revised planning, which also reflects the actual time usage of the past weeks. My primary lessen is that I should spend more time writing so it doesn't all pile up at the end.

A number of important revisions:

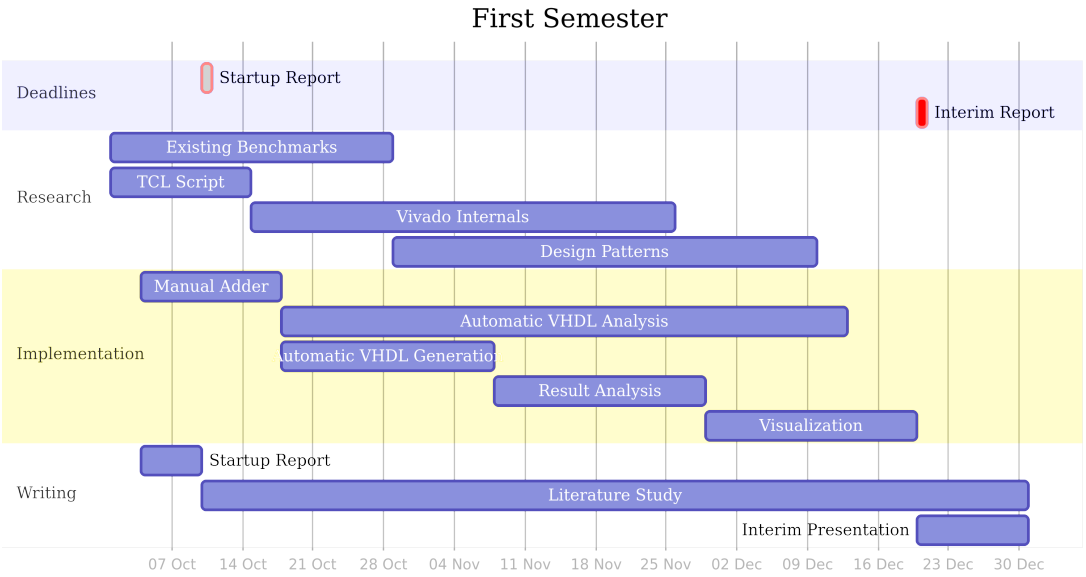
1. Implementing automatic VHDL generation is not feasible and was replaced by using generics and template VHDL files.
2. There are no benchmarks with the same approach or parameters, so I scrapped the "Compare Results to Existing Benchmarks" task.
3. Implementation of multiple vendors will likely take longer than initially anticipated. But some other tasks, such as visualization, have already been implemented.
4. Extra time to read more research and refine the literature study has been planned.

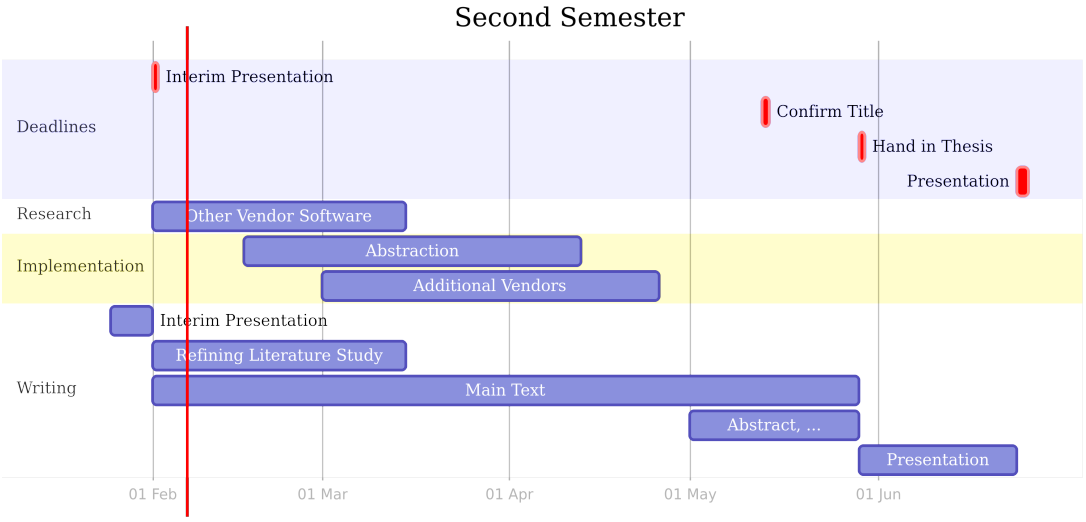
4.1 Initial Planning





4.2 Revised Planning





References

Altera. Fpga performance benchmarking methodology. Technical report, 08 2007. URL https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/wp/wpfpgapbm.pdf.

Altera. Opencore stamping and benchmarking methodology. Technical report, 05 2008. URL https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/tb/tb-098-opencore-stamping-and-benchmarking-methodology.pdf.

barriet. Speed grade, Nov 2007. URL https://forums.xilinx.com/t5/CPLDs-Archived/Speed-Grade/m-p/3115/highlight/true#M163.

Umer Farooq, Zied Marrakchi, and Habib Mehrez. Fpga architectures: An overview. In *Tree-based heterogeneous FPGA architectures*, pages 7–48. Springer, 2012.

Wenyi Feng, Jonathan Greene, and Alan Mishchenko. Improving fpga performance with a s44 lut structure. In *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '18, pages 61–66, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-5614-5. doi: 10.1145/3174243.3174272. URL https://people.eecs.berkeley.edu/~alanmi/publications/2018/fpga18_s44.pdf.

Kevin E Murray, Scott Whitty, Suya Liu, Jason Luu, and Vaughn Betz. Titan: Enabling large and complex benchmarks in academic cad. In *23rd International Conference on Field programmable Logic and Applications*, Sept 2013. doi: 10.1109/FPL.2013.6645503. URL http://www.eecg.utoronto.ca/~kmurray/titan.html.

Raphael Njuguna. A survey of fpga benchmarks, Nov 2008. URL https://www.cse.wustl.edu/~jain/cse567-08/ftp/fpga/.

Henry Selvaraj, Luka Daoud, and Dawid Zydek. A survey of high level synthesis languages, tools, and compilers for reconfigurable high performance computing. volume 240, 09 2013. doi: 10.1007/978-3-319-01857-7_47. URL https://www.researchgate.net/publication/257430335_A_Survey_of_High_Level_Synthesis_Languages_Tools_and_Compilers_for_Reconfigurable_High_Performance_Computing/download.

Elias Vansteenkiste, Alireza Kaviani, and Henri Fraisse. Analyzing the divide between fpga

academic and commercial results. 12 2015. doi: 10.1109/FPT.2015.7393137. URL https://people.eecs.berkeley.edu/~alanmi/publications/other/fpt15_xilinx.pdf.

Helmuth von Moltke. *Militārische Werke*. Number vol. 2, pt. 2 in Militārische Werke. E. S. Mittler, 1900.

Xilinx. Measuring device performance and utilization: A competitive overview. Technical report, 08 2017. URL https://www.xilinx.com/support/documentation/white_papers/wp496-comp-perf-util.pdf.

Appendix A

Code

The code is available at `https://thesis.dries007.net/interim-report/`.

- `vivado_fmax.tcl`
- `vivado.py`
- `autorun.py`
- `plot.html`

Appendix B

Raw Result Data

All raw data is available at `https://thesis.dries007.net/interim-report/data/`.

Interactive version of fig. 3.1: `https://thesis.dries007.net/interim-report/plot.html`.

FACULTY OF ENGINEERING TECHNOLOGY
DE NAYER (SINT-KATELIJNE-WAVER) CAMPUS
Jan De Nayerlaan 5
2860 SINT-KATELIJNE-WAVER, België
tel. + 32 16 30 10 30
fet.denayer@kuleuven.be
www.fet.kuleuven.be