Cyber-Physical Systems Engineering Project

Upgrading the

Controlling Mechanism

of the diamond experiment at
The Institute of Atomic and Subatomic Physics

Executed at
Institute of Computer Engineering
of Vienna University of Technology
assisted by
Univ.Prof.Dr. Andreas Steininger
by

Patrick Knöbel

Stachegasse 21/608
A-1120 Vienna

Vienna, at 13 June 2017

Table of Contents

Task

The research project “Diamond” at The Institute of Atomic and Subatomic Physics aims to characterize and exploit the remarkable quantum properties of defect centers in diamond.

For the experiments a very fast and precise controlling mechanism is needed. The task of this project work is to extend this mechanism. An already existing FPGA based solution should be upgraded.

1. Background

The development of quantum computers is no longer science fiction and starts to play a big role in computer science. To push on the development of such computers lots of research has to be done. There are lots of concepts and materials which can help to build better quantum computers. Impurities in the diamond for example show great potential to be used.

There are hundreds of different known diamond impurities, but the Nitrogen-Vacancy impurity (NV center) has been proven to have remarkably stable and long-lived quantum behavior [1]. These effects are still observable at room temperature. This makes it a very interesting candidate for quantum computers. An NV center is a single molecule integrated inside the diamond structure.

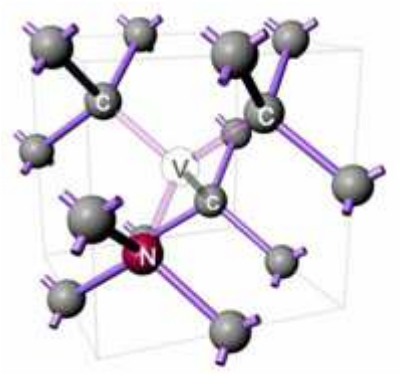

Figure 1: Diamond crystal with NV impurity [1]

In this experiment the electronic spin of the NV center molecule is entangled with light. The quantum state of this molecule is very short-lived. The stimulation must be done in a nanosecond scale. It is only possible to use an FPGA for all hardware based actions because a software based approach would be too slow. Therefore the FPGA is the time base and takes care that everything is timed correctly. It is essential to understand how much time is passed to follow the underlying physical effects.

The outputs of the FPGA are used to control lasers, laser impulse generators, photon counters, a microwave source, trigger pulses and when needed other hardware. One control sequence is some microseconds long and includes about one hundred events. The scheduling of different sequences

must be also done by the FPGA. A long delay between two sequences can lead for example to the loss of the spin of the molecule under test.

The NV center molecule has to be brought into the right quantum state. To achieve that it has to be stimulated with exactly timed laser and microwave pulses. This is called tuning. A tuning process is not guaranteed to work every time. After tuning the crystal it is also necessary to check if the NV center has the right state, otherwise the process has to be repeated. This can be achieved by counting the number of emitted photons.

On the electronical side, the check if the tuning was successful can be done by counting pulses from photon counter during a defined interval and comparing them to preset values. Furthermore, after some time the NV center loses its state and the tuning has to be repeated anyways.

Only after the successful tuning the actual measurement cycle can be performed as shown in Figure 2.

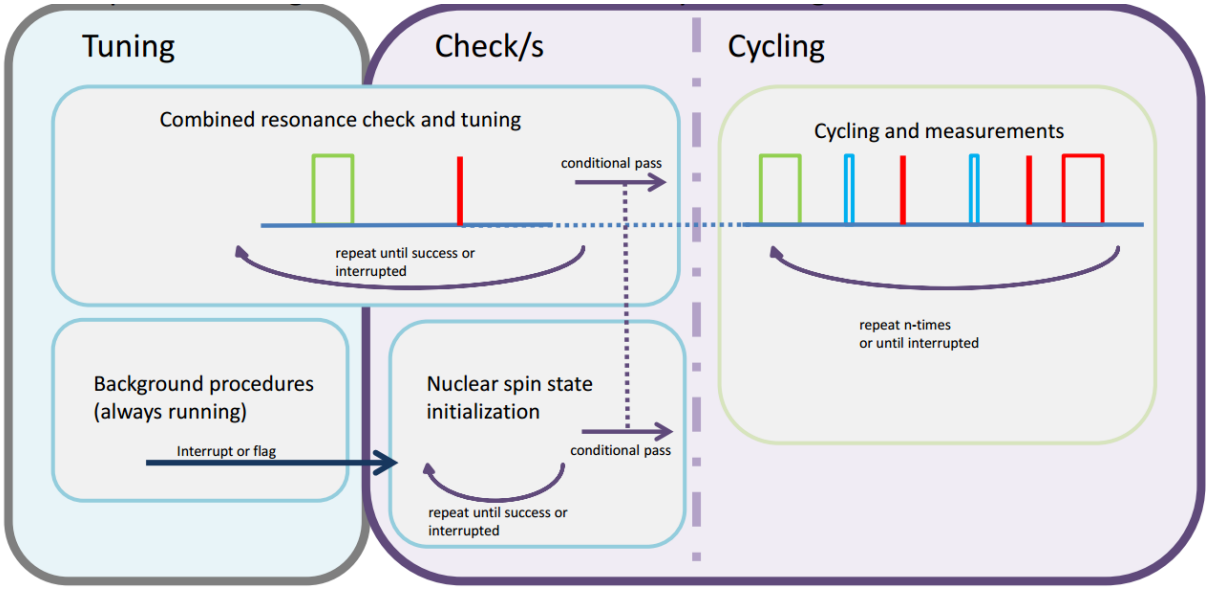

Figure 2: Overview of the experiment

2. Platform

At the Diamond Research Group at the Institute of Atomic and Subatomic Physics Dipl.-Ing. Georg Wachter already built a Test System and provided a FPGA design for the controlling mechanism for first calibrations and testing of the measurement system.

For the FPGA Platform, the system already used in other experiments, called “STEMLAB 125-14” sold by the company Redpitaya [2] was chosen. The main reasons for that are the low cost of the platform compared to others and the low effort to program the FPGA.

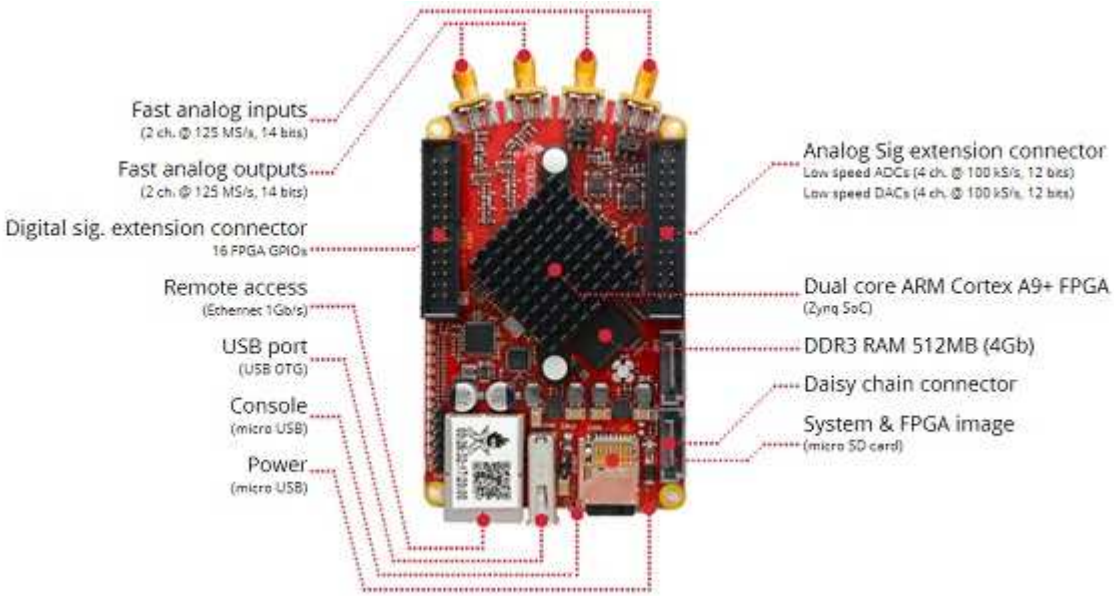

Figure 3: STEMLAB 125-14 Board

The board is based on the Xilinx Zynq 7010 FPGA which includes a dual core ARM Cortex A9 processor, 28k logic cells and 2.1Mb of Block RAM. It also has different ADC and DAC converters which are not used in this project.

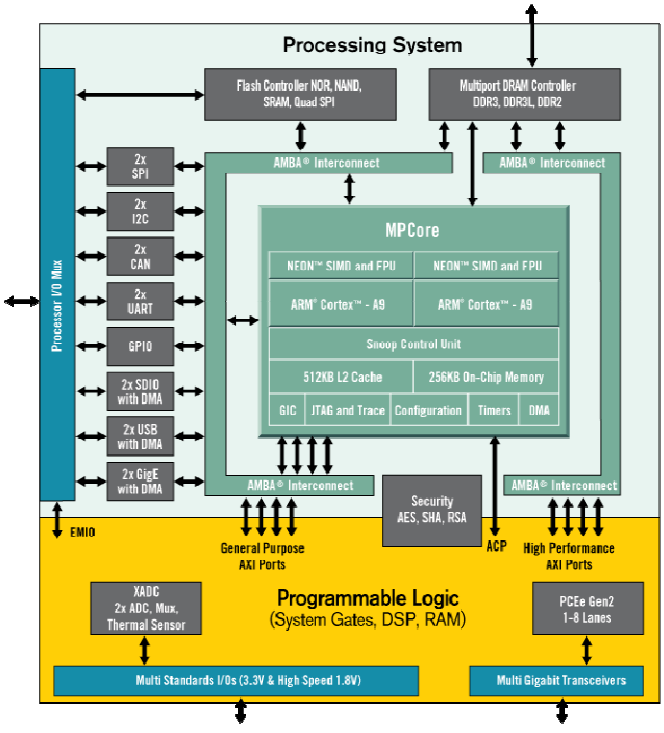

Figure 4: Zynq-7000 Platform

The STEMLAB platform provides an out-of-the-box running setup with a Linux environment running on the ARM processor and a Vivado example project. Redpitaya also develop their own internal bus system which is easy to use and directly accessible over a memory mapped interface in Linux.

3. Specifications

Based on the experiment the following specifications arise:

- Programmable sequence to be used for tuning and measurement:
  - minimal 5
  - duration of minimal 4s
  - timeout setting to schedule a rerun
  - priority order settings (a tuning sequence must run before a measurement sequence)
  - constant time delay between two sequences
  - logging the sequence changes
  - resolution:
    - minimal: 4ns (250 MHz)
    - desired: 1ns (1 GHz)
  - Patterns per sequence:
    - minimal 32 different patterns
    - a pattern consists of 14 outputs
    - programmable idle output pattern
    - as jitter free as possible
- Digital inputs for the decision if a sequence was successfully executed:
  - 2 digital inputs
  - pulse counter up to 10MHz
  - programmable time window for the pulse counter
  - programmable upper and lower counter limits for the sequence rerun decision
  - logging the results

The existing FPGA Design from Dipl.-Ing. Georg Wachter only supported one sequence with a resolution of 4ns and had no digital inputs.

4. Implementation

The specification that the pattern should be as “jitter free” as possible and the time between two sequences should be constant restrict the use of the ARM Processor. Therefore the management and sequencing of the patterns are all implemented in hardware and done by the FPGA. The ARM processor is only used for programming, monitoring and logging.

To reduce the complexity of the design it is separated into the following parts:

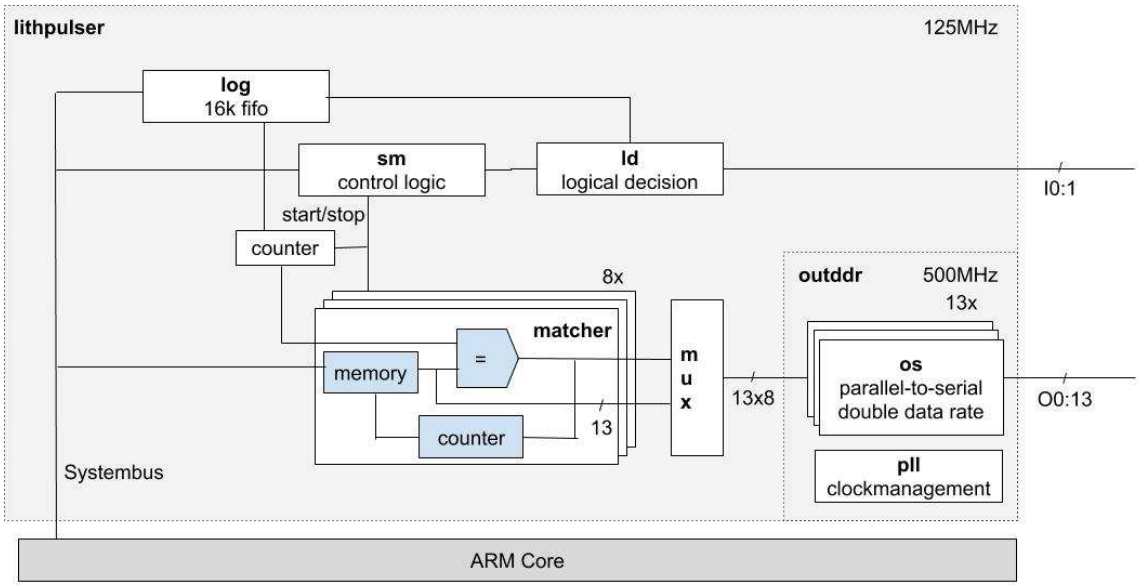

Figure 6: Implementation overview

The ARM Processor is connected over the Systembus. How the processor is controlling the hardware is described in the Appendix. First all the sequences are programmed. Then the ARM Processor starts the experiment. During the experiment the log fifo is frequently read out by the ARM processor to reduce the likeliness of an overrun.

In the following sub-sections all modules will be described.

5.1. Matcher

The matcher is the heart of the design. It has its own organized memory and is built to work in parallel units. This is necessary to achieve higher output rates at a lower overall clock rate.

The matcher can output one pattern per clock cycle. It allows a continuous burst of output pattern changes. To achieve this, the address calculation for the memory access must be one clock ahead.

```
--RAM adress calc must be one clock earlier:
caddres: process(we, rcs , le, t_l, p, setram_add)
begin
    if(we='0') then --Not writing RAM => prepair for read:
        if(le.t=t_l) then
            ram_add <= std_logic_vector(TO_UNSIGNED(rcs*EMAX+p+1,11));
        else
            ram_add <= std_logic_vector(TO_UNSIGNED(rcs*EMAX+p,11));
        end if;
    else
        ram_add <= setram_add;
    end if;
end process;
```

Figure 7: Matcher RAM address calculation

To indicate that a pattern change has to be done, the additional output bit called “valid” is used. The signal is then used by the mux module to generate a continuous output stream.

The whole design should support different sequences. Therefore, the matcher keeps track of the active sequence number. The internal memory is split into parts, one for each sequence. To change the number of sequences or the maximum number of pattern changes per sequence, this fragmentation of the memory must be re-evaluated.

To write a sequence, the pattern changes must be programmed in incessant order by time. The matcher handles internal pointers for each sequence to save the number of stored changes. To write new sequences, these pointers can be cleared by the input signal “clear_all”.

5.2. Mux

The mux is the most time critical module. It takes the output from the eight matchers and calculates the final output patterns for the parallel to serial converter.

To reduce the critical path length the mux is implemented in nine stages. The first eight shift the pattern changes and the last one fills the unchanged outputs with the last known pattern.

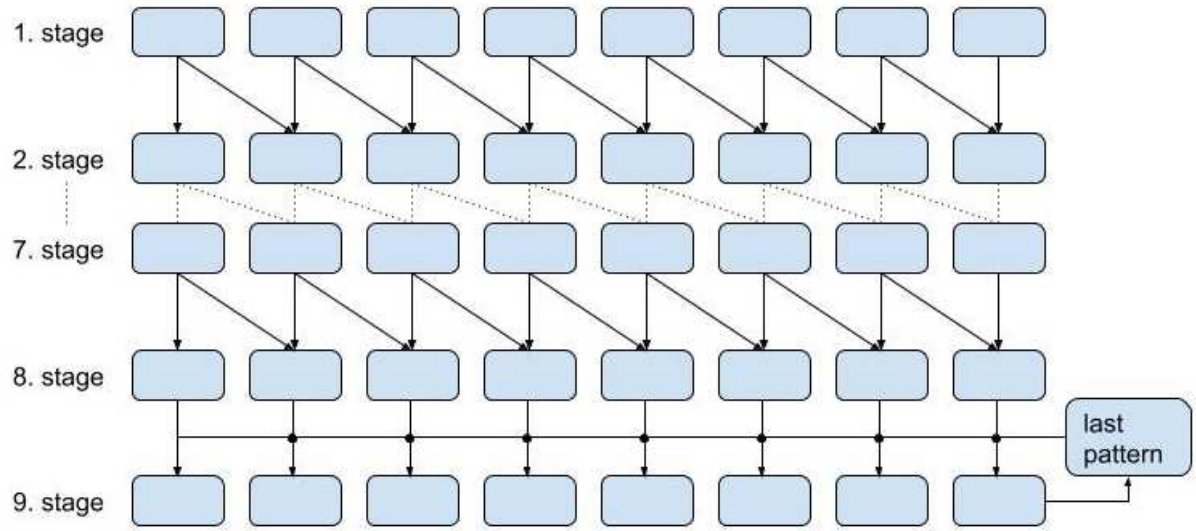

Figure 8: Mux internal structure

With the help of the “valid” bit from the matcher the outputs with pattern changes get shifted to the right till another output with a pattern change is found. It is possible that there will be outputs left with no pattern change. Therefore, in the last stage all outputs left are set to the last output pattern. The new pattern from the last output is then taken for the next replacement.

For example if only the input 2 has a valid pattern. This pattern will be copied to the right in each stage. Therefore in stage 8 input 2 to 8 have a valid pattern, but input 1 is still invalid. In stage 9 the last pattern as shown in Figure 8 is then used for the input 1 in stage 9.

After the mux module all outputs have a valid pattern. This parallel information is then sent to the parallel to serial converter in the outDDR mode.

5.3. OutDDR

In this module the parallel stream of the output data from the mux module is serialized to the 1 GHz output signal. As shown in the overview diagram, the outDDR module is running at 500 MHz. To get

to the 1 GHz output signal the Double Data Rate feature is used and therefore output changes are possible at the falling and rising edge of the clock signal.

To achieve this functionality the resource “OSERDESE2” is used. This component is capable of serializing up to 14bits in one clock cycle and is mainly used for interfacing DDR3 SDRAM. A detailed description of this hardware component can be found at [3, p161ff].

To synchronize the mux output signals with the “OSERDESE2” we need an additional PLL. This is described in more detail in the section “Key Challenges”.

5.4. LD

One additional requirement for the design is the ability to react to external events, for example to check if a tuning was successful or needs to be repeated. Furthermore, a person with less experience in developing FPGA designs should be able to add additional decision logic if necessary. Therefore, in the ld module only the counting and gating is done. The decision calculation is done in the sm module.

The output signal is generated in the matcher and is then processed through the mux and the outDDR modules. The matcher and ld module share the same time counter. Therefore the ld gate window is 128ns ahead of the output pattern.

5.5. SM

This module includes the main state machine of the whole design. The state machine has the following structure:

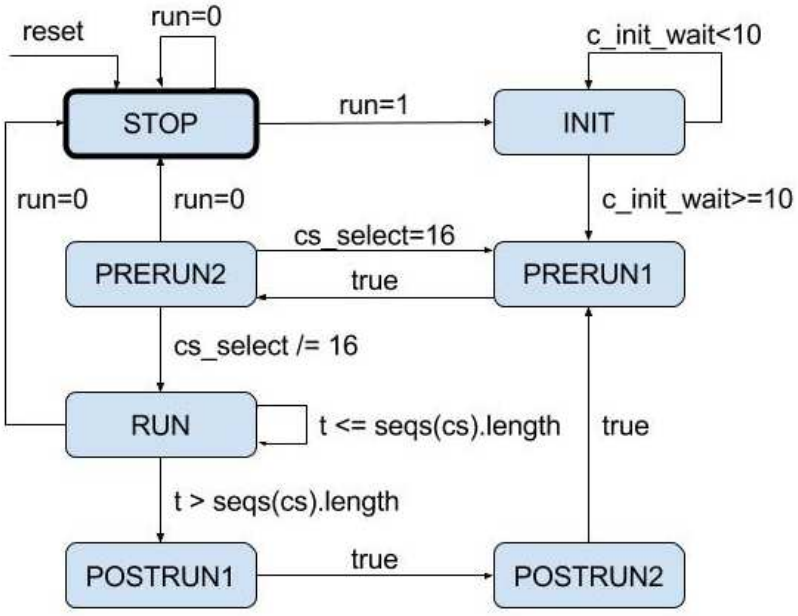

Figure 9: Main state machine

Some states are split to reduce the length to the most critical path.

State STOP

This state is corresponds to the configuration register run. When run is set to zero it is guaranteed that the state machine will jump into this state. Also the global counter is set to zero. When run is set to 1 the sate machine will jump to the state INIT.

State INIT

By entering the INIT state all counters and the log buffer are reset. To reset the log buffer properly a delay of 10 clock cycles is introduced.

State PRERUN1

In this state the next sequence to be executed is selected. The priority order and the last execution time are taken into account. Therefore, only the sequence with the highest priority which has to be re-run is selected.

State PRERUN2

If in state PRERUN1 a sequence is found, all initializations to run the sequence are done in this state. The sequence number and start time are added to the log buffer if desired. If no sequence is found (cs_select=16) the state is changed to PRERUN1 to search again for a new sequence.

State RUN

This state represents a running sequence. At the end of the sequence the state is changed to POSTRUN1.

State POSTRUN1

In this state the “logical decision” logic is executed. The next state will be POSTRUN2.

State POSTRUN2

On the result of the logical decision the sequence parameters are calculated to schedule a re-run if necessary. The ld counter values are added to the log buffer if desired. The next state will be PRERUN1.

5.6. Log

The log module is a standard FIFO IP core from the Vivado Toolchain. The memory size is as large as possible for the left BRAM in the design to reduce the risk of losing any log data.

5.7. Systembus

The mapping of the internal registers, memory and log buffer is made as simple as possible and should be self-explanatory. How the memory has to be configured is described in the Appendix.

5. Key Challenges

Output resolution challenge

In the first stages of evaluation it looked like that the chosen FPGA did not have the capability for a sequence resolution of 1 ns. The used FPGA supports only a maximum of 450 MHz clocking frequency over the global clock distribution network.

The FPGA has also a parallel to serial converter hardware called “OSERDESE2”. This hardware is mainly used to interface DDR3 SDRAM and is capable of serializing up to 14 bits in one clock cycle. But is also needs to be clocked with half the output frequency. To generate an output stream with a

resolution of 1ns, this component therefore has to be clocked with 500 MHz. Additionally this 500 MHz has to be in phase the parallel input signal.

These conditions introduced a big challenge. We could have used an additional PLL channel and generated the required 500 MHz easily. But to guarantee that both clocks are in phase they have to go over the same clock buffer and as mentioned above this is not possible.

It turned out that the “OSERDESE2” hardware can be connected directly to the PLL output with a special IO clock buffer. Therefore, the solution is to use an additional PLL component and sync the new generated 500 MHz clock to the 125 MHz input clock. This ensures that both clocks are in phase. This is the only way to use the parallel to serial converter at higher speeds than the global clock distribution network allows.

Synthesis Problems

One problem with the Vivado tool [4] is the lack of a default variable range check in the behavioral simulation. Additionally, there is no warning in the synthesis if one always has a false or true comparison based on the range of variables. Furthermore, it was not possible to run a post synthesis simulation in this special case [5]. The binary mapping of the state of a state machine is also not documented during the synthesis.

To initialize the log buffer, it must be held in reset for some clock cycles. After testing the initialization in the simulation it emerged that the reset period must be longer. To correct this problem the counter value was increased without correcting the range of the variable. In the simulation everything looked good, but in the synthesized design the state machine was trapped in the INIT state and did not do anything. All the problems mentioned above resulted in a long search for the cause of this fault.

6. Conclusion

All specifications were satisfied for the final design. The possible number of programmable sequences could be increased from 5 to 16. Also 127 patterns per sequence were achieved instead of the specified minimum of 32. Additionally the desired resolution of 1ns could be achieved.

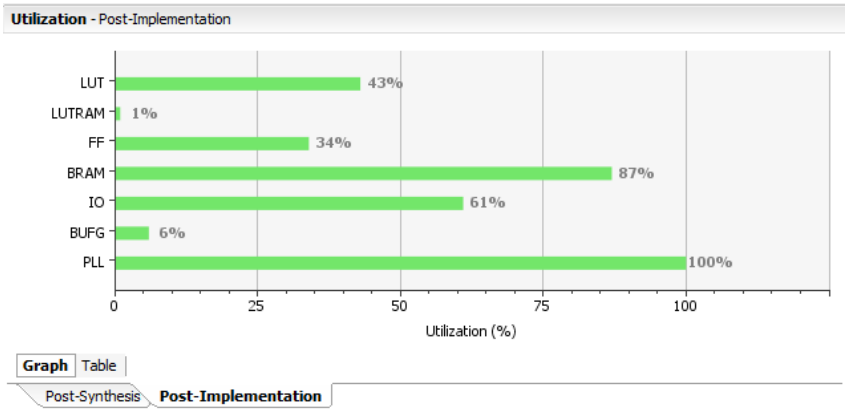

Figure 10: Post Implementation Utilization

In general, the whole design is optimized for timing to reduce the length of the most critical path. Also the resource utilization is around 50% as shown above. This makes it harder for the synthesis to meet the timing constrains.

The theoretically reachable maximum output resolution is 833ps. This limit is caused by the hardware serializer. Therefore, it is possible to increase the resolution in the future, but there is not much more performance to gain.

List of illustrations

Sources

[1]: http://atomchip.org/diamonds/

[2]: http://redpitaya.com/

[3]: https://www.xilinx.com/support/documentation/user_guides/ug471_7Series_SelectIO.pdf

[4] : https://www.xilinx.com/support/answers/67551.html

[5] : https://www.xilinx.com/support/answers/64097.html

[6]: https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html#productAdvantages