Points	Grade

Team: 6

01228774 Constantin SCHIEBER #1
0122576 Petar KOSIC #2

Digital Integrated Circuits Lab (LDIS)

384.088, Summer Term 2018

Supervisors:
Christian Krieg, Martin Mosbeck, Axel Jantsch

Task 1: Implementing a TRNG on a FPGA

Abstract

A random number generator (RNG) based on the paper from[1] had to be implemented. The paper proposed a solution based on creating a metastable flipflop. As metastabiltiy requires the violation of setup and hold times a delay line based on Lookup Tables (Specific Hardware in FPGAs to perform Lookup Table tasks) was introduced to create a phase shift between clock and data input ports of the flipflop. The reproduction of the results was not possible, as placement of the Combinatorial Logic Blocks (e.g. LUT) and the routing process influence the outcomes heavily and change from synthesis run to synthesis run. Therefore this solution foucuses an making the synthesis process more deterministic by creating various constraints on the plcaement and routing process.

[1] **Majzoobi2011**.

1 Problem statement and motivation

A true-random number generator (TRNG) based on the paper from[2] had to be implemented.

The paper shows an approach that promises a resource efficient and fast solution to the problem of generating long enough keys for cryptographic algorithms. The reproduction of the results in the paper is a challenging and important task, as the paper is very incomplete regarding information about the actual implementation.

The source of randomness is based on the metastability characteristics of a d-flipflop when violating setup or hold times of the flipflop. To create such violations the novel approach of delay lines is introduced. The data and clock input ports of the flipflop are both preceded by a series of LUTs blocks. These LUTs blocks are functionally just used as inverters - but due to the inner structe of a LUTs a delay can be introduced by applying different control signals to the LUTs ports.

2 Implementation (proposed solution)

A naive implementation of the papers approach did not lead to any reproduceable output. Therefore a list of stepstones that were encountered while working on a solution is compiled. The most important takeway from the following is that the top and bottom delay lines need to be symmetric in terms of placement, routing and ports.

The paper authors omit any details on the actual implementation, but as this question in the comments of a blog post (most likely from the paper author) shows it was of interest for them as well[3] and that they also struggled with optimization during synthesis.

The following constraints provide us with a solid ground for further work as placement, routing and pin assignments are now mostly consistent with every synthesis run.

2.1 Unwanted Optimization of Logic Elements

This was mainly a problem for the LUT blocks and sometimes a problem for the d-flipflop. The synthesis tool tries to minimize the use of LUT blocks as it rightfully detects the redundancy in the logic. It therefore minimizes the LUT6 (6 inputs) blocks to combinations of smaller LUT Blocks - controlling the delay is therefore not possible anymore (on the one hand due to the missing delay of the mutliplexers in the lut and on the other hand due to the possible asymmetry in the top and bottom delay paths).

2.1.1 Mitigation

The Xilinx / Vivado Documentation states on this topic that the DONT_TOUCH attribute is the best option.[4] It should ensure that the signal is kept, in reality this does not work reliably, or the author didn't understand it fully.

Listing 1: Attribute Declaration in VHDL, for the flipflop

```
        attribute DONT_TOUCH : string;
        attribute DONT_TOUCH of flipflop: label is "TRUE";
```

2.2 Unwanted Optimization of Logic Element Placement

The luts would be placed in many different ways for every synthesis run, even if nothing else changed. Fixing the location of the luts to specific slices solved this problem. A spreadsheet was created to ease the creation of the assignments.

2.2.1 Mitigation

Listing 2: Set location of a lut

```
set_property LOC SLICE_X85Y125 [get_cells gen_bot_coarse_plds[1].i_bot_coarse_pld/g0_b0]
```

[2] **Majzoobi2011**.
[3] **Mehrdad2010**.
[4] **XilinxAttributes**.

2.3 Unwanted Optimization of Routing Paths

As the routing tool is - rightfully - trying to minimize setup and hold times in the routed design it creates asymmetric paths with delays that can be longer than the logic delay introduced by the lut blocks. Net Delays of up to 126ns could be observed, while the logic delay on the same path would only account for 7ns. This in itself would not cause a problem if the top and bottom path would show the same delay. But already a 5% deviation of top and bottom path would render the delay of the delay lines useless.

2.3.1 Mitigation

Timing analysis and optimization can be disabled for net paths or cells. For the sake of simplicity all cells in in the delay path very excluded from the timing analysis. This lead to more direct and more consistent routings.

Listing 3: Disable the timing analysis for luts

```
set_disable_timing [get_cells {gen_top_coarse_plds[*].i_top_coarse_pld/g0_b0}]
set_disable_timing [get_cells {gen_top_fine_plds[*].i_top_fine_pld/g0_b0}]

set_disable_timing [get_cells {gen_bot_coarse_plds[*].i_bot_coarse_pld/g0_b0}]
set_disable_timing [get_cells {gen_bot_fine_plds[*].i_bot_fine_pld/g0_b0}]
```

Another option that has to be considered is to route the delay path nets first and only after that the rest. No specific improvements could be obsereved, alltough other groups reported some.

Listing 4: Route critical nets first

```
set preRoutes [get_nets  {*lut_t_* *lut_b_*}]
route_design -net [get_nets \$preRoutes] -delay
route_design -preserve
```

2.4 Unwanted Optimization of Port Elements

The synthesis / routing tool can decide how to arrange the port mapping of the luts, e.g. port A6 of the lut can be mapped to signal I0 while the expected mapping would be A0:I0. This again causes asymmetry in the delay paths as certain ports (with A6 being the fastest and A0 the slowest) of the lut react faster than others (as per the design of the lut).

2.4.1 Mitigation

Ports of logic elements can be fixed by the following commands.

Listing 5: Disable the timing analysis for luts

```
set_property LOCK_PINS { I0:A1 I1:A2 I2:A3 I3:A4 I4:A5 I5:A6 } [get_cells *i_top_coarse_pld/g0_b0 ]
set_property LOCK_PINS { I0:A1 I1:A2 I2:A3 I3:A4 I4:A5 I5:A6 } [get_cells *i_top_fine_pld/g0_b0 ]
set_property LOCK_PINS { I0:A1 I1:A2 I2:A3 I3:A4 I4:A5 I5:A6 } [get_cells *i_bot_coarse_pld/g0_b0 ]
set_property LOCK_PINS { I0:A1 I1:A2 I2:A3 I3:A4 I4:A5 I5:A6 } [get_cells *i_bot_fine_pld/g0_b0 ]
```

2.5 Top Module State Machine

For the main implementation we used a state machine for all the logic.

STATE_UART_WELCOME:
The Logic starts in STATE_UART_WELCOME, which sends an welcome message to the user via UART. Afterwards it goes into STATE_IDLE and waits for new receiving data.

STATE_IDLE:
When it receives new data the state moves forward to STATE_PROD_RN.

STATE_TEST_RN:
Dummy State

STATE_PROD_RN:
In this state the state machine waits until the TRNG is ready to create next random number and then goes into STATE_PROD_RCV.

STATE_PROD_RCV:
In this state it saves the random number and switches into STATE_UART_RN.

STATE_UART_RN:
The STATE_UART_RN sends the random number over UART and then switch the state to STATE_SSEG_-RN.

STATE_SSEG_RN:
The STATE_SSEG_RN state sends the random number data to the seven segment module, which shows the lower 8 nibbles of the random number in hex on the display.

2.6 UART

For the UART we used the given UART IP and slightly modified it for our purposes.

2.7 Seven Segment Display

The Seven Segment Display implementation was in theory straight forward. There were some problems with variables but we fixed it by using signals. Also in the beginning we used way too fast refreshing periods. The problem occurred when we connected the package with the implementation of the RNG part. The seven segment display uses a refresh period of 1ms to 16ms as stated in the datasheet. We've chosen 8ms as the period with 1ms refresh time per segment. Timing Diagram from https://reference.digilentinc.com/reference/programmable-logic/nexys-4-ddr/reference-manual for better understanding:

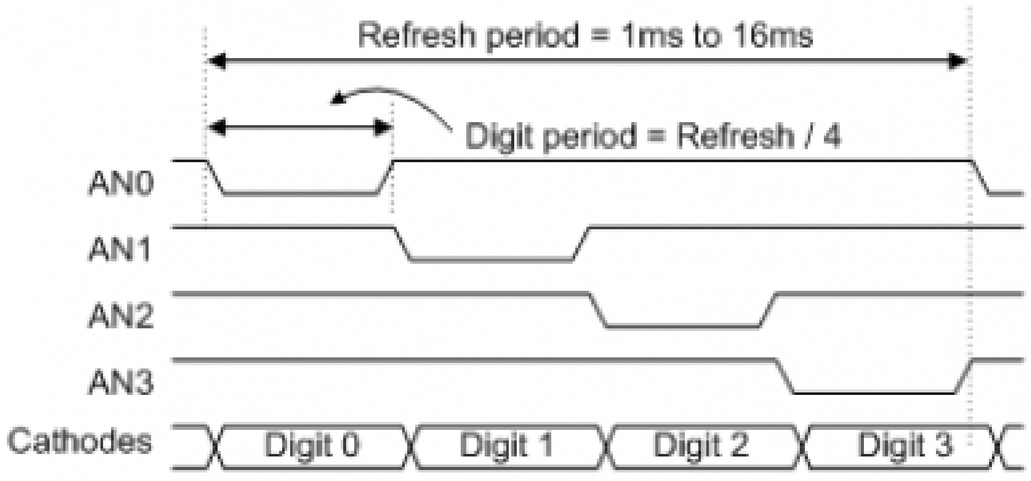

Figure 19. Four digit scanning display controller timing diagram.

The modul has an internal buffer for the numbers, that should be displayed on the segments. The seven segment module triggers the data copy with an enable signal from the TRNG. Therefore the Module has only a simple state machine with two states: Idle and a state for copying data to the buffer.

3 Results (verification plan)

The goal of implementing a working TRNG was not reached during this lab assignment.

The number generation was extremely deterministic, as the p-values obtained by the NIST Test Suite score all a 0. The Test Suite (and also its wrapper) throw many igamc: UNDERFLOW errors that were at first wrongfully interpreted as errors due to the used compiler and the used math libraries. After further investigation this just means that the p-value is so low that an exception is thrown and the p-value gets replaced by 0.

As a good p value should be around 0.5 (equal chance of guessing and not guessing the correct sequence) the conclusion must be drawn that the numbers are not suitable for cryptography at all.

Interestingly enough a run with the dieharder test suite yields good results on the same data. Further investigation is needed on the why.

3.0.1 Design and Testing

A random number takes 128 cycles to be generated, at 100MHz one 128bit random number therefore needs 1.28us. Considering the UART Overhead and the wait time of 0.01ms in the python script that reads from

Table 1: Dieharder results for experiment TRNG

Test	**tSample**	**P-value**	**Passed**
sts_monobit	100000	0.18544306	PASSED
sts_runs	100000	0.95997337	PASSED
sts_serial	100000	0.13818581	PASSED
sts_serial	100000	0.99575888	WEAK

Table 2: NIST-STS results for experiment XYZ

Test	**P-value**
Frequency	0.0
Block Frequency	0.0
...	0.0
Complexity	0.0

UART the generation of 100000 random numbers takes around 10 minutes. Testing with the dieharder testsuite takes around 10 minutes, the NIST Test Suites are finished after 1 minute.

Table 3: Utilization of the project

Site Type	Used	Fixed	Available	Util%
Slice LUTs	517	128	63400	0.82
LUT as Logic	517	128	63400	0.82
LUT as Memory	0	0	19000	0.00
Slice Registers	442	1	126800	0.35
Register as Flip Flop	442	1	126800	0.35
Register as Latch	0	0	126800	0.00
F7 Muxes	5	0	31700	0.02
F8 Muxes	0	0	15850	0.00

4 Discussion

Our main findings were that working with very incomplete data from the authors of the study (how was the design actually implemented) is very hard. We tried to contact the authors but got now response in weeks.

The current state of the TRNG requires further investigation as it is not working at the moment.

Gladly we could get the project to a point where working with a solid basis would be possible (e.g. deterministic routing). From now on a more intense timinga analysis of the path and optimized placement by hand would be the way to got.

5 Conclusions

The has a better understanding on how constraints in vhdl and vivado work. Sadly we couldn't provide a working solution.

6 Assessment

This is the place for the teaching staff to add notes for team assessment.

#	**Issue**	**Yes**	**No**
1 Implementation			
1.1	Does the implementation conform to the specification?		
1.2	Is the implementation resource-efficient?		
1.3	Is the implementation's hardware description language (HDL) complexity low?		
1.4	Is the implementation well-documented?		
1.5	Is the file structure's complexity low?		
2 Coding style			
2.1	Is the line width of code limited to 80 characters?		
2.2	Is white space appropriately used?		
2.3	Are tabs used for indentation?		
2.4	Are separators used to logically divide the file contents?		
2.5	Are meaningful comments given?		
3 Code reuse			
3.1	Is publicly available code re-used?		
3.2	Is non-publicly available code re-used?		
3.3	Are the sources of re-used code cited?		
4 Interaction			
4.1	Was the specification unclear to the team?		
4.2	If yes, did the team contact the teaching staff to make the specification clear?		
5 Report			
5.1	Are there typos?		
5.2	Is the report grammatically correct?		
5.3	Is there redundant information?		
5.4	Is the report's format consistent?		
5.5	Are captions properly used and numbered? Page numbers?		
5.6	Are figures and tables properly referenced in the body text?		
5.7	Are resources properly referenced?		