Internship Report

Meghna Roy Chowdhury

Submitted to : Prof Rajat Subhra Chakraborty

Duration: 10th May 2019 to 30th June 2019

Research Topic

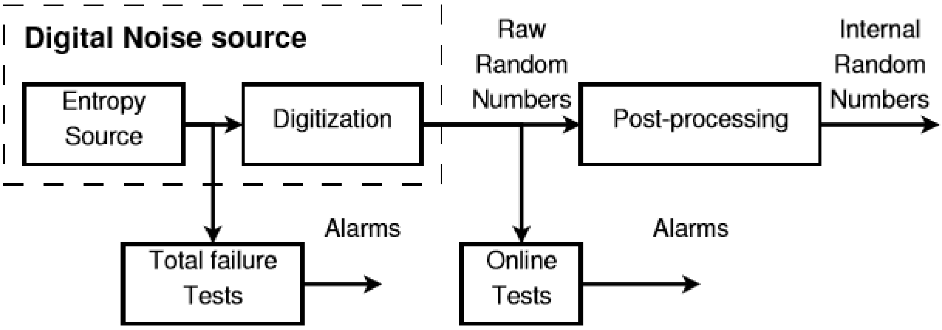
The usage of FPGA in Hardware Security

Abstract

The task given to me was to implement a TRNG in Xilinx FPGA (Nexys 4 DDR).

True randomness cannot be obtained via computational methods. Instead, physical phenomena such as noise in electronic devices should be the source of the unpredictable nature of TRNGs. Due to their importance for security, TRNGs are subjected to strict evaluations in the process of industrial certification.

True Random Number Generators (TRNGs) are essential building blocks of modern embedded security systems. They enable various cryptographic algorithms, protocols and secured implementations by providing secret keys, initialization vectors, random challenges and masks. The security of these applications relies on the uniformity and unpredictability of the utilized random numbers.



The generic architecture of a TRNG.

Details Analysis of the project:

This project was done in 2 parts:

1. TRNG Circuit – Ring Oscillators as noise source and clock, Carry 4 primitive, Bit extractor
2. BRAM and UART

<u>Part1:</u>

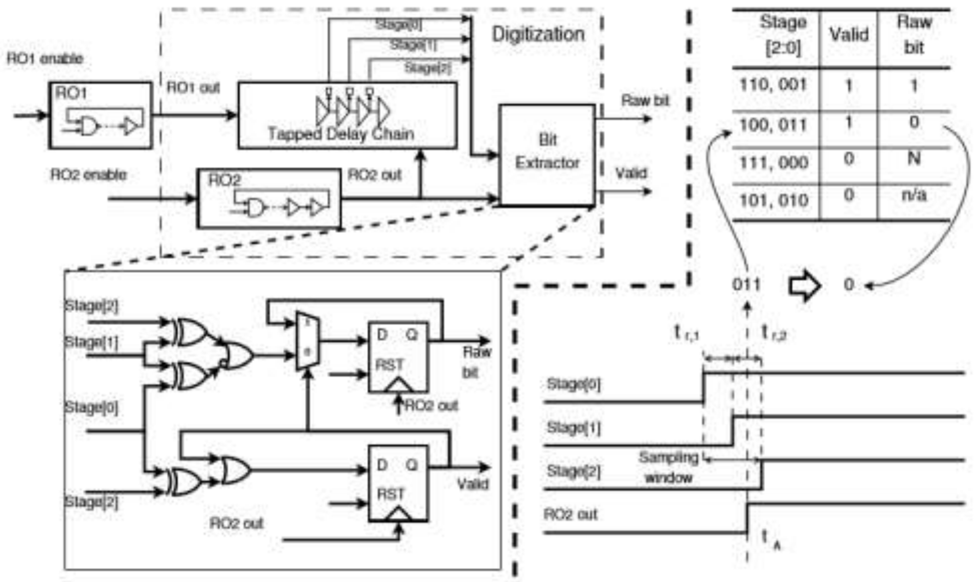
The type of TRNG implemented was an ES-TRNG: A High-throughput, Low-area True Random Number Generator based on Edge Sampling.

The basic concept of this is to sample a free running oscillator at a high frequency.

This implementation consists of 2 techniques:

1. Variable-precision phase encoding – we encode the high precision regions only.
2. Repetitive sampling – We keep sampling such that we can avoid the low precision region

Architecture:

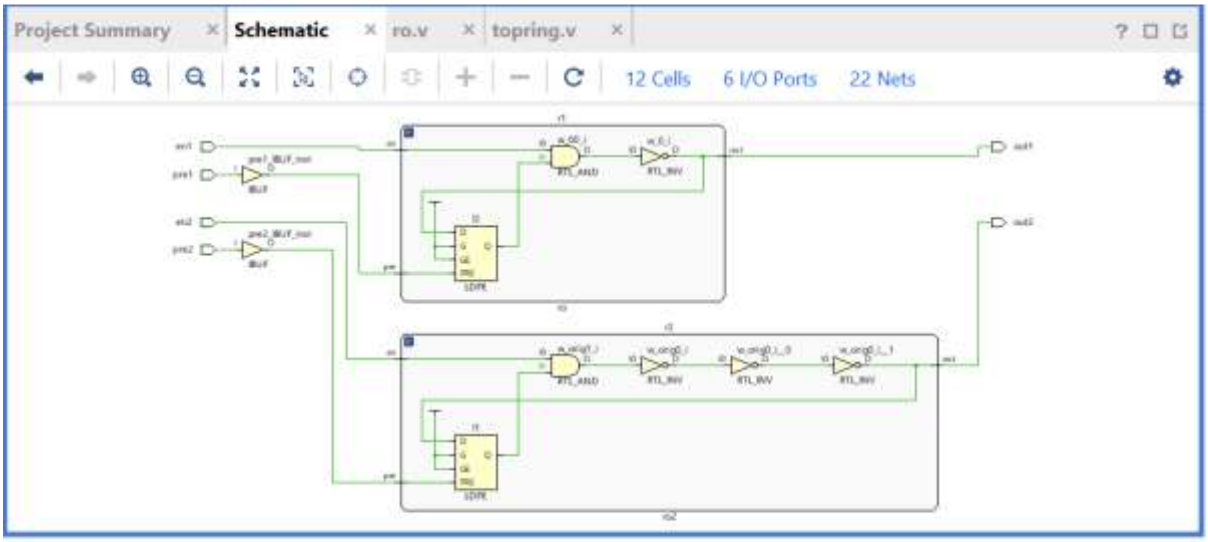


Stage [2:0]	Valid	Raw bit
110, 001	1	1
100, 011	1	0
111, 000	0	N
101, 010	0	n/a

Figure 3: The architecture and operation principle of the digital noise source.

1. Ring Oscillators

Ring oscillators usually produce deterministic signals . However , in practice , they have some noise which makes the edge of transitions unpredictable. Hence, ring oscillators in practice are non-deterministic.

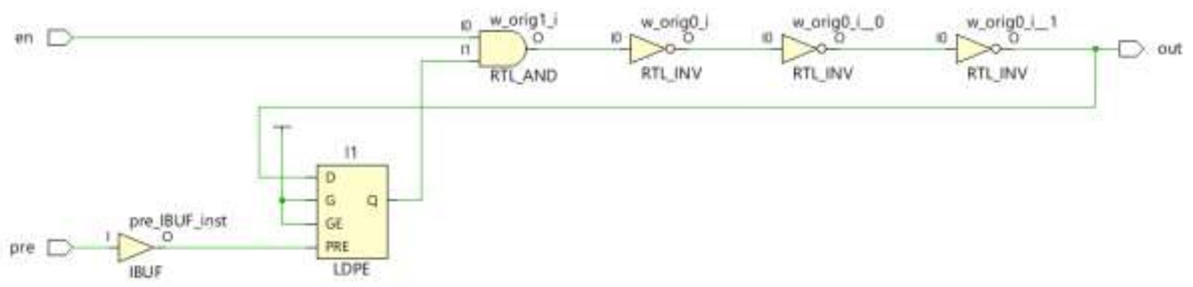


a. *Digital Noise Source (Ring Oscillator 1)*

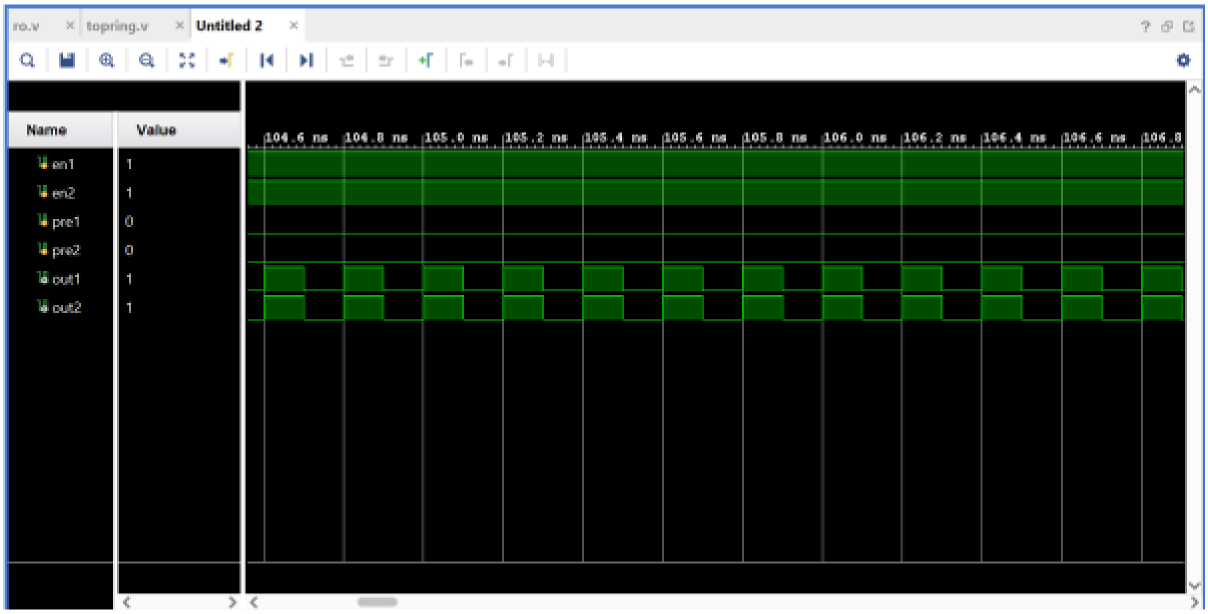
Ring oscillator 1 is the digital noise source. It uses 2 LUTs. The D Latch is just used to visualise the output of the ring oscillator in simulation

b. Clock source (Ring oscillator 2)

Ring oscillator 2 is a clock source. It uses 3 LUTs. The D Latch is just used to visualise the output of the ring oscillator in simulation



Simulation results:



Name	Value
en1	1
en2	1
pre1	0
pre2	0
out1	1
out2	1

In the actual implementation, we don't use the D Latch. The initial output depends on the noise produced.

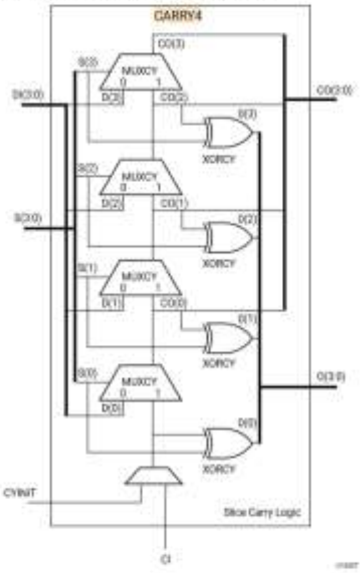
2. Tapped Delay

The Carry4 primitive is used as a Tapped Delay chain.

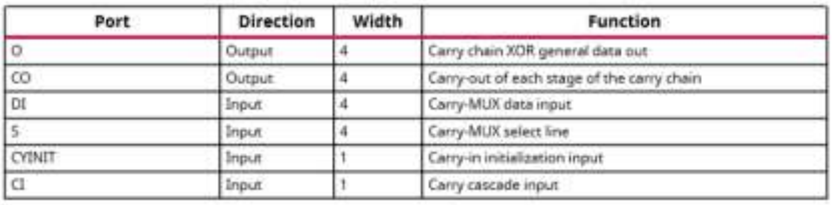
The Carry 4 primitive consists of MUXs and XOR gates:

CARRY4

Primitive: Fast Carry Logic with Look Ahead

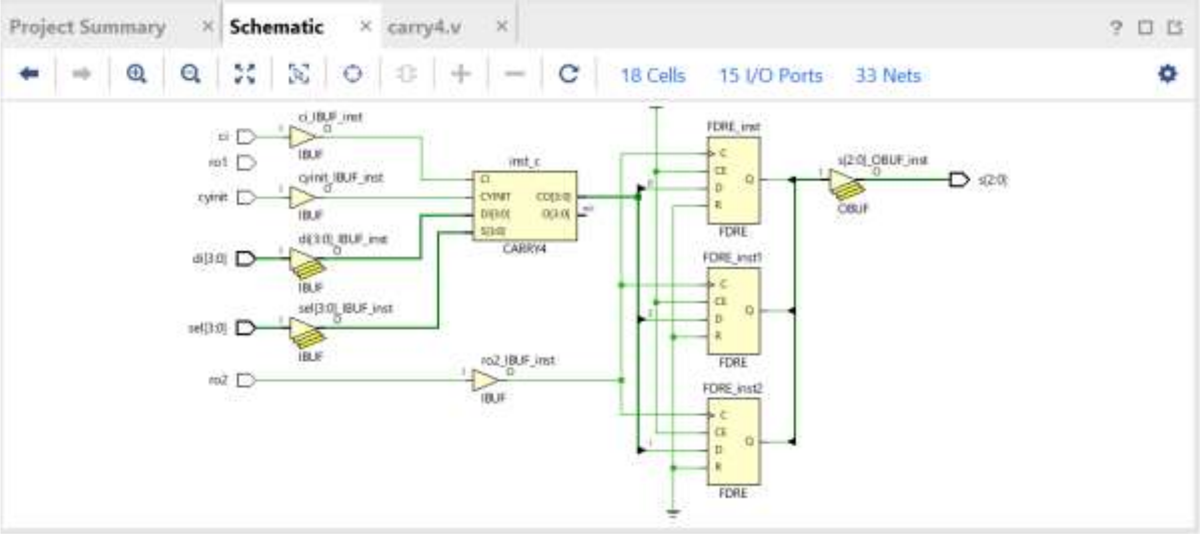


Port Descriptions



Port	Direction	Width	Function
O	Output	4	Carry chain XOR general data out
CO	Output	4	Carry-out of each stage of the carry chain
DI	Input	4	Carry-MUX data input
S	Input	4	Carry-MUX select line
CYINIT	Input	1	Carry-in initialization input
CI	Input	1	Carry cascade input

- > CI was set as RO1 (free running oscillator)
- > The select lines(S[3:0]) were set as 1
- > DI[3:0] were set as 0
- > CYINIT was set as 0
- > We sample the output of the first 3 MUXes (CO[0], CO[1] and CO[3]) using primitive D flip flop.

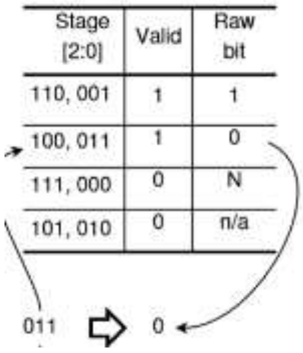


3. Bit extractor

This is an encoding circuit. It uses the output of the Carry 4 chain as its input , R02 as clock and gives 2 bits (Raw , Valid) as output.

The 3 bit input can have any one of the 8 combinations. However, the occurrence of all the combinations are not possible.

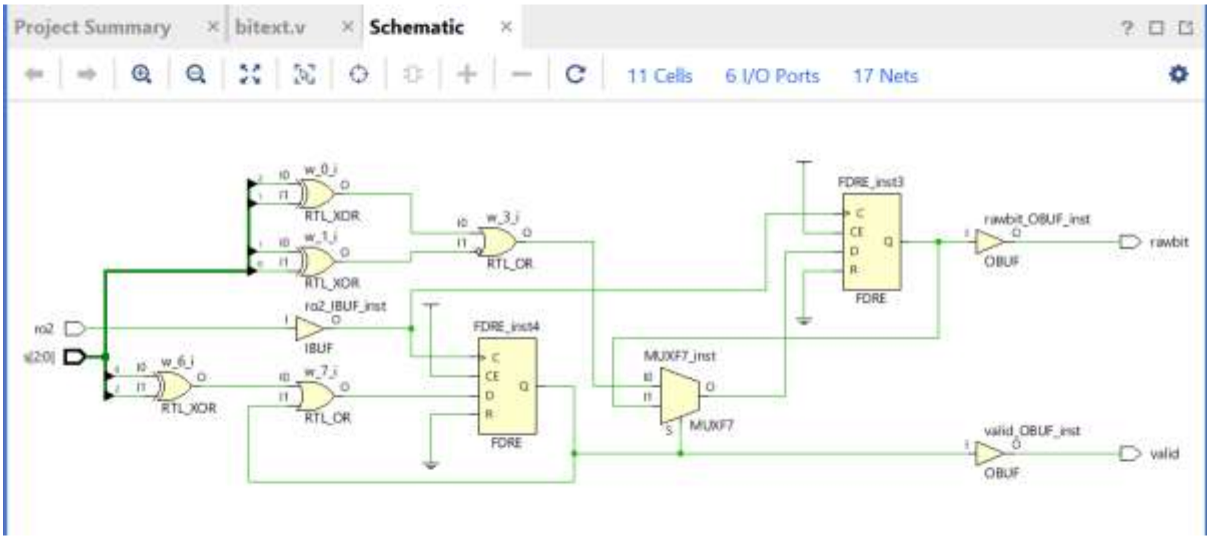
The combinations of valid and invalid bits are given in the table below:



Stage [2:0]	Valid	Raw bit
110, 001	1	1
100, 011	1	0
111, 000	0	N
101, 010	0	n/a

011 ⇨ 0

When valid=0 => Low precision region ; valid=1 => High precision region



Operation:

i. ***Ring Oscillator 1 is the noise source and is enabled from the beginning.***

ii. ***Ring Oscillator 2 is the clock source. It is enabled 250 ns after RO1 is enabled.***

iii. ***The output of R01 is given to the tapped delay chain , and the output of R02 is the clock. The tapped delay chain gives 3 bits as output.***

iv. ***The output of the tapped delay chain is given to the bit extractor which basically encodes the 3 bit output of the***

1. *Variable Precision encoding:*

This technique is enabled by using both the tapped delay chain and the bit extractor.

The position of the captured edge of the noise source is encoded into a raw bit. Since the delays tr/f,1 and tr/f,2 of the tapped delay chain are much smaller than the oscillation period T01, the digitization module captures the oscillator phase with high precision around signal edges when the phase value is around 0 or D. This region is called the high precision region. The samples from this region are encoded by either 0 or 1 depending on the phase. In the remainder of the cycle, the edge is captured with low precision, i.e. only the correct half-period can be determined from the captured data. This region is called the low precision region. The samples from this region are not used (encoded as N). The sampling of the delay chain is triggered by the rising edge of the signal RO2 out. Due to the accumulated timing jitter, the relative sampling position follows a Gaussian distribution. Increased accumulation time tA leads to a wider jitter distribution.

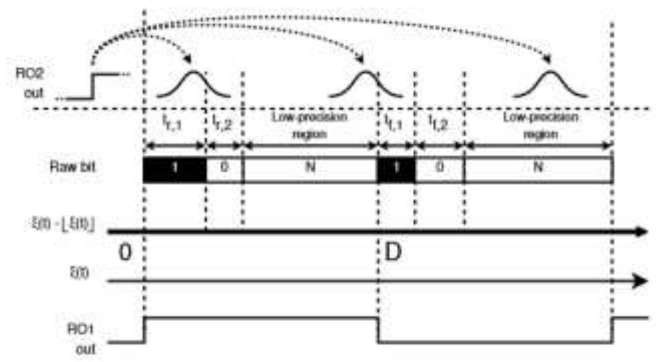


Figure 4: Variable-precision phase encoding.

2. *Repetitive Sampling:*

The proposed digitization module has a small critical path, which enables the digital noise source operating at a higher frequency than other components. Repetitive sampling is synchronized to the high frequency signal RO2 out, aiming to reduce the time needed to hit the high-precision region, thereby improving the throughput. Once the high-precision region is hit, a Valid signal is generated.

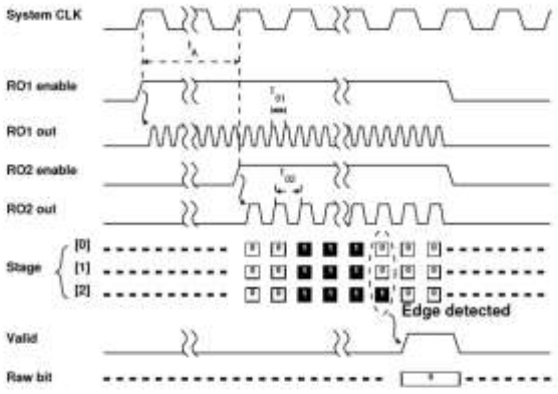
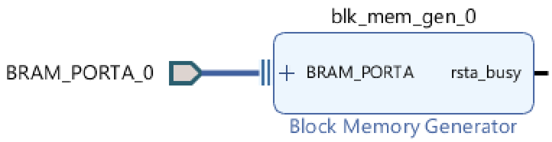
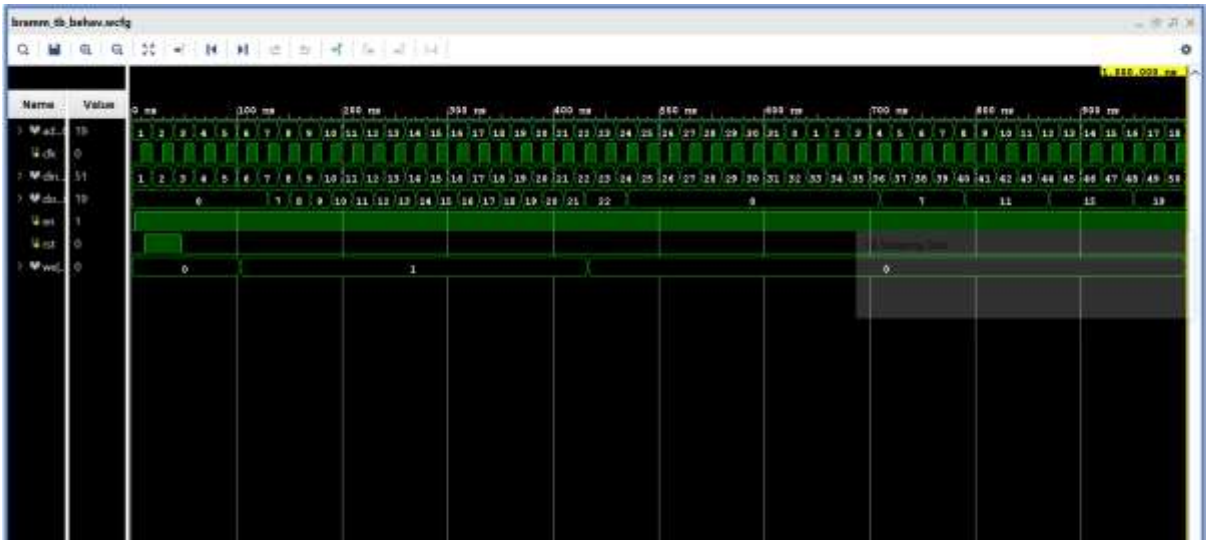


Figure 5: Timing diagrams of repetitive sampling.

Part2:

BRAM





1. UART

This was done using IP catalog – axiuartlite.

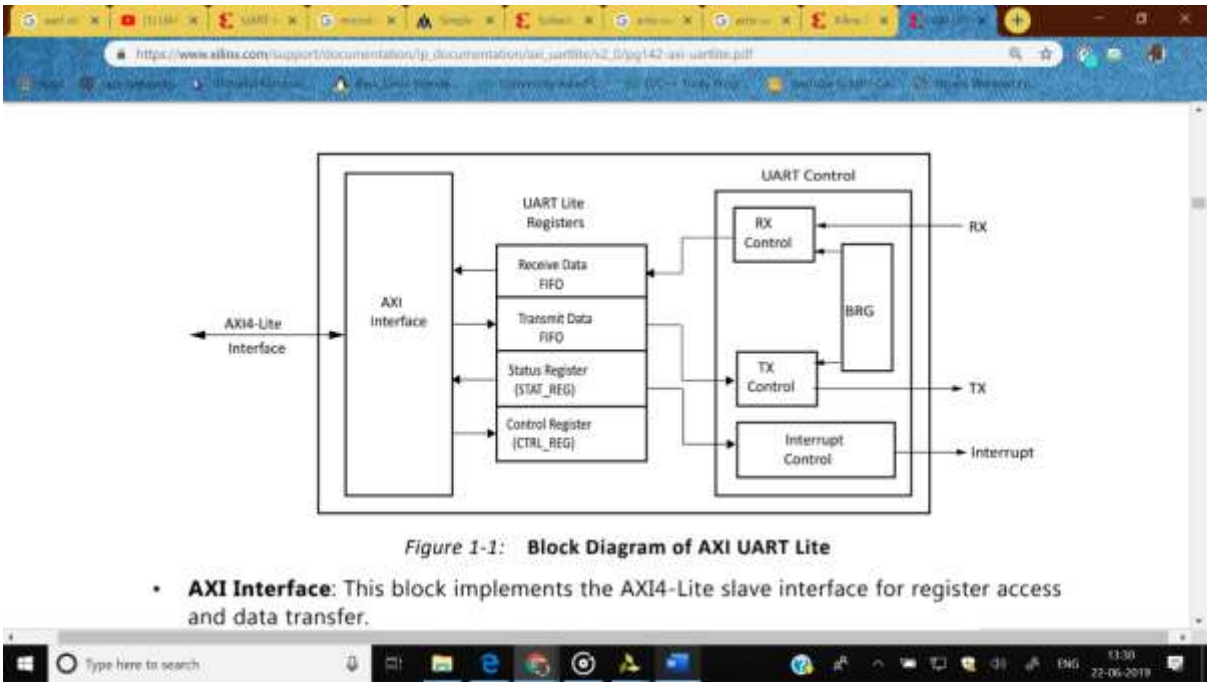
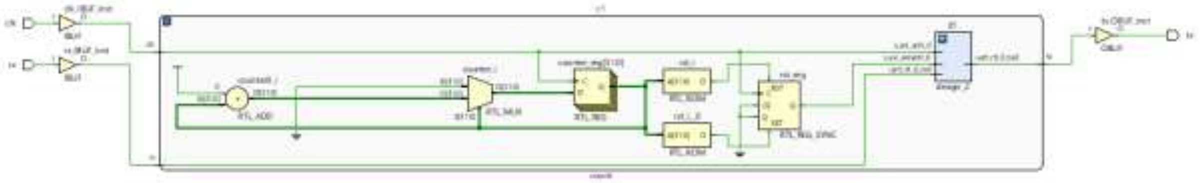
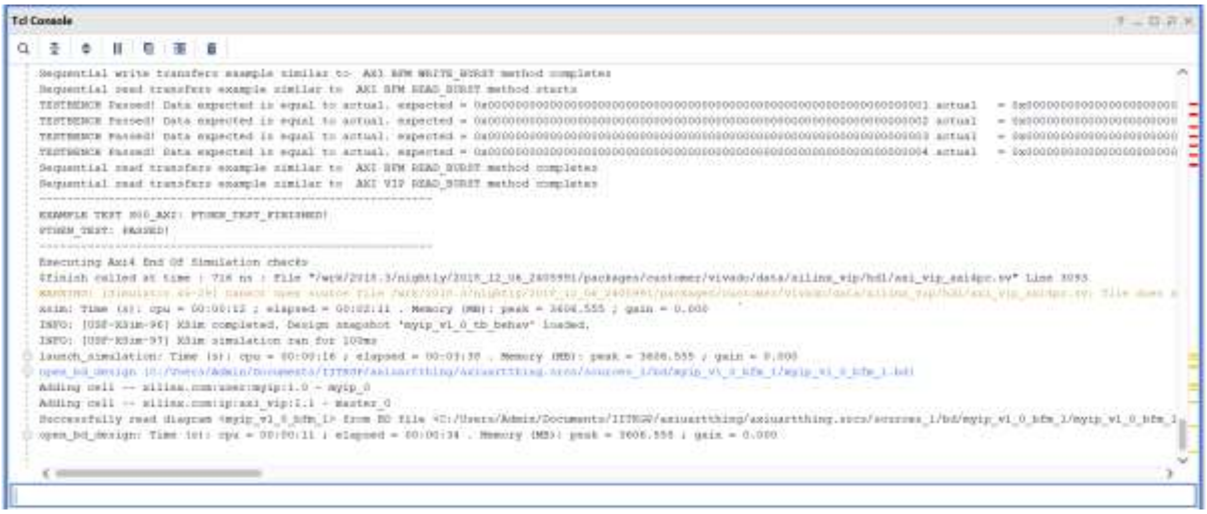


Figure 1-1: **Block Diagram of AXI UART Lite**

- **AXI Interface**: This block implements the AXI4-Lite slave interface for register access and data transfer.





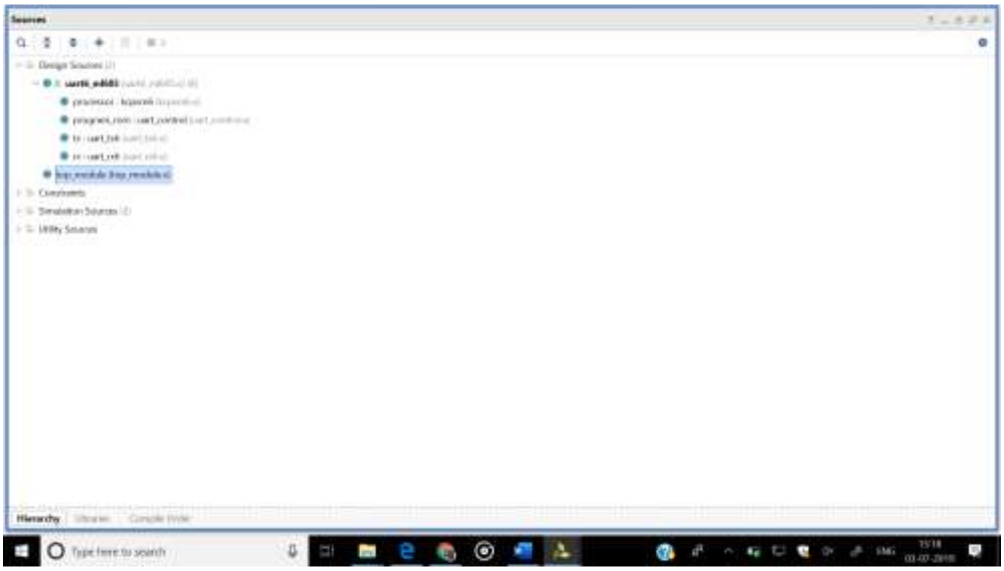
```
Sequential write transfers example similar to  AXI BFM WRITE_BURST method completes
Sequential read transfers example similar to  AXI BFM READ_BURST method starts
TESTBENCH Passed! Data expected is equal to actual. expected = 0x000000000000000000000000000000000000000000000000000000000000000001 actual   = 0x0000000000000000000000000
TESTBENCH Passed! Data expected is equal to actual. expected = 0x000000000000000000000000000000000000000000000000000000000000000002 actual   = 0x0000000000000000000000000
TESTBENCH Passed! Data expected is equal to actual. expected = 0x000000000000000000000000000000000000000000000000000000000000000003 actual   = 0x0000000000000000000000000
TESTBENCH Passed! Data expected is equal to actual. expected = 0x000000000000000000000000000000000000000000000000000000000000000004 actual   = 0x0000000000000000000000000
Sequential read transfers example similar to  AXI BFM READ_BURST method completes
Sequential read transfers example similar to  AXI VIP READ_BURST method completes
---------------------------------------------------------

EXAMPLE TEST S00_AXI: PTGEN_TEST_FINISHED!
PTGEN_TEST: PASSED!
---------------------------------------------------------

Executing Axi4 End Of Simulation checks
$finish called at time : 718 ns : File "/wrk/2018.3/nightly/2018_12_06_2405991/packages/customer/vivado/data/xilinx_vip/hdl/axi_vip_axi4pc.sv" Line 3093
WARNING: [Simulator 45-29] Cannot open source file /wrk/2018.3/nightly/2018_12_06_2405991/packages/customer/vivado/data/xilinx_vip/hdl/axi_vip_axi4pc.sv: File does n
xsim: Time (s): cpu = 00:00:12 ; elapsed = 00:02:11 . Memory (MB): peak = 3606.555 ; gain = 0.000
INFO: [USF-XSim-96] XSim completed. Design snapshot 'myip_v1_0_tb_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 100ms
launch_simulation: Time (s): cpu = 00:00:16 ; elapsed = 00:03:30 . Memory (MB): peak = 3606.555 ; gain = 0.000
open_bd_design {C:/Users/Admin/Documents/IITGW/axiuartthing/axiuartthing.srcs/sources_1/bd/myip_v1_0_bfm_1/myip_v1_0_bfm_1.bd}
Adding cell -- xilinx.com:user:myip:1.0 - myip_0
Adding cell -- xilinx.com:ip:axi_vip:1.1 - master_0
Successfully read diagram <myip_v1_0_bfm_1> from BD file <C:/Users/Admin/Documents/IITGW/axiuartthing/axiuartthing.srcs/sources_1/bd/myip_v1_0_bfm_1/myip_v1_0_bfm_1
open_bd_design: Time (s): cpu = 00:00:11 ; elapsed = 00:00:34 . Memory (MB): peak = 3606.555 ; gain = 0.000
```

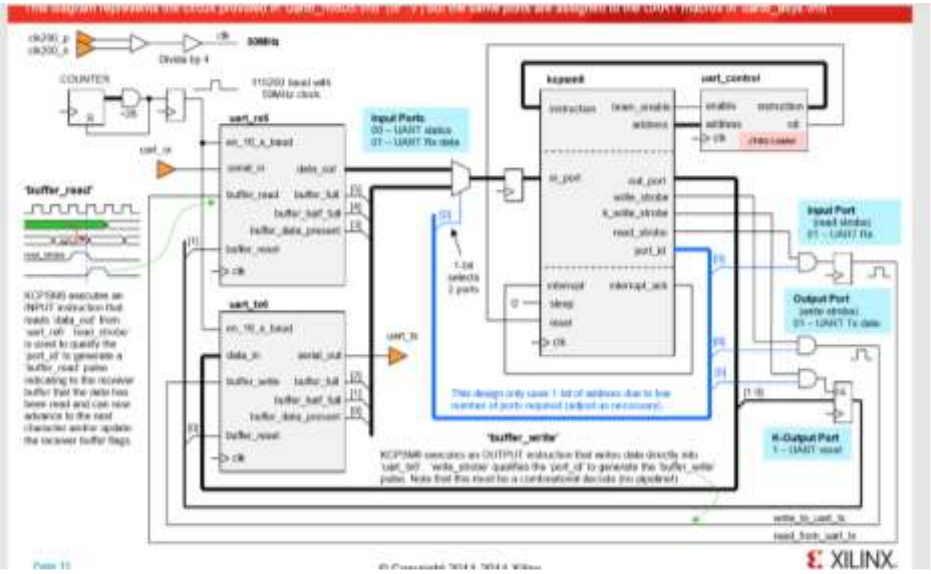
2. <u>Picoblaze</u>

The following design was implemented In VIVADO in Verilog.



Sources

- Design Sources (2)
  - uart6_ml605 (uart6_ml605.v) (4)
    - processor : kcpsm6 (kcpsm6.v)
    - program_rom : uart_control (uart_control.v)
    - tx : uart_tx6 (uart_tx6.v)
    - rx : uart_rx6 (uart_rx6.v)
  - top_module (top_module.v)
- Constraints
- Simulation Sources (2)
- Utility Sources



KCPSM6 executes an INPUT instruction that reads 'data_out' from 'uart_rx6'. 'read_strobe' is used to qualify the 'port_id' to generate a 'buffer_read' pulse indicating to the receiver buffer that the data has been read and can now advance to the next character and/or update the receiver buffer flags.

This design only uses 1 bit of address due to low number of ports required (adjust as necessary).

KCPSM6 executes an OUTPUT instruction that writes data directly into 'uart_tx6'. 'write_strobe' qualifies the 'port_id' to generate the 'buffer_write' pulse. Note that this must be a combinatorial decode (no pipeline).

Page 11

© Copyright 2011-2014 Xilinx

Implementation:

Specifications:

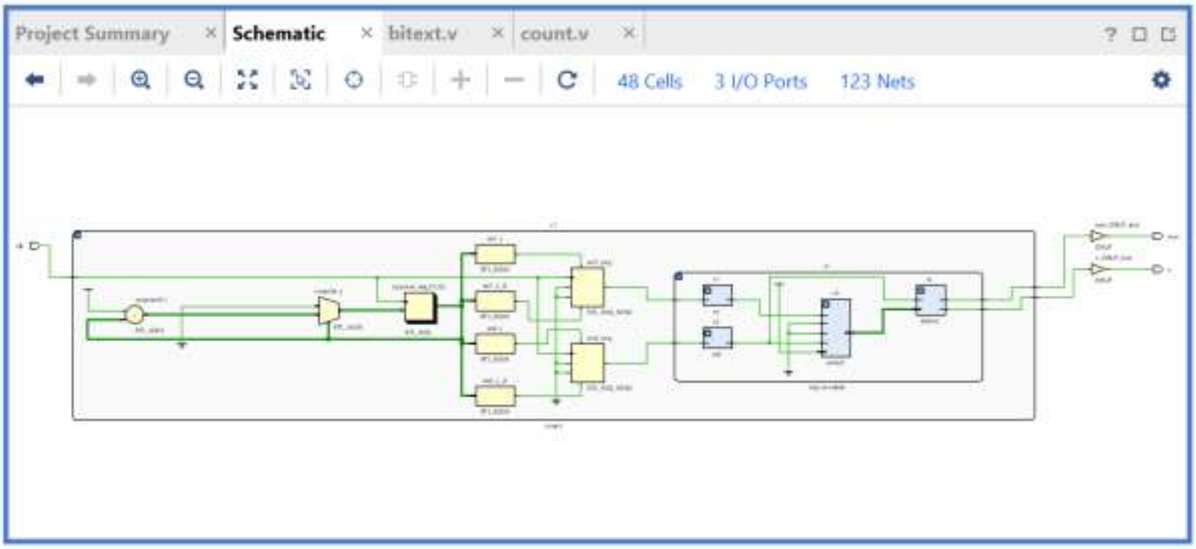
- Xilinx Nexys4 DDR (Artix 7)
- Clock – 100MHz
- Software: Vivado 2018.3

To visualise the randomness in on board LEDs, we increased the sampling time to 4s.

For this we used a slow clock on software.

The design was implemented using 5 modules:

i. Control
ii. Count
iii. Top module
iv. Ring oscillator 1
v. Ring oscillator 2
vi. Tapped Delay
vii. Bit Extractor



To get higher samples , we eliminated the slow clock and used a BRAM to store the random bits generated and UART was used to write the generated bits to the PC.

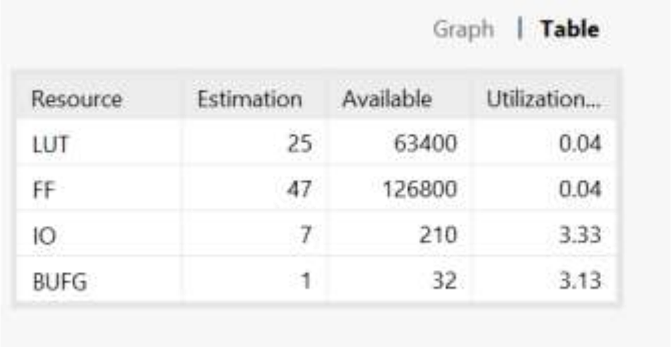
Picoblaze (KCPSM6) block was used to transfer data from the FPGA to the PC using UART.

RESULTS

On sampling the bits at 4s , 1’s were observed on the LEDs at random intervals , at:

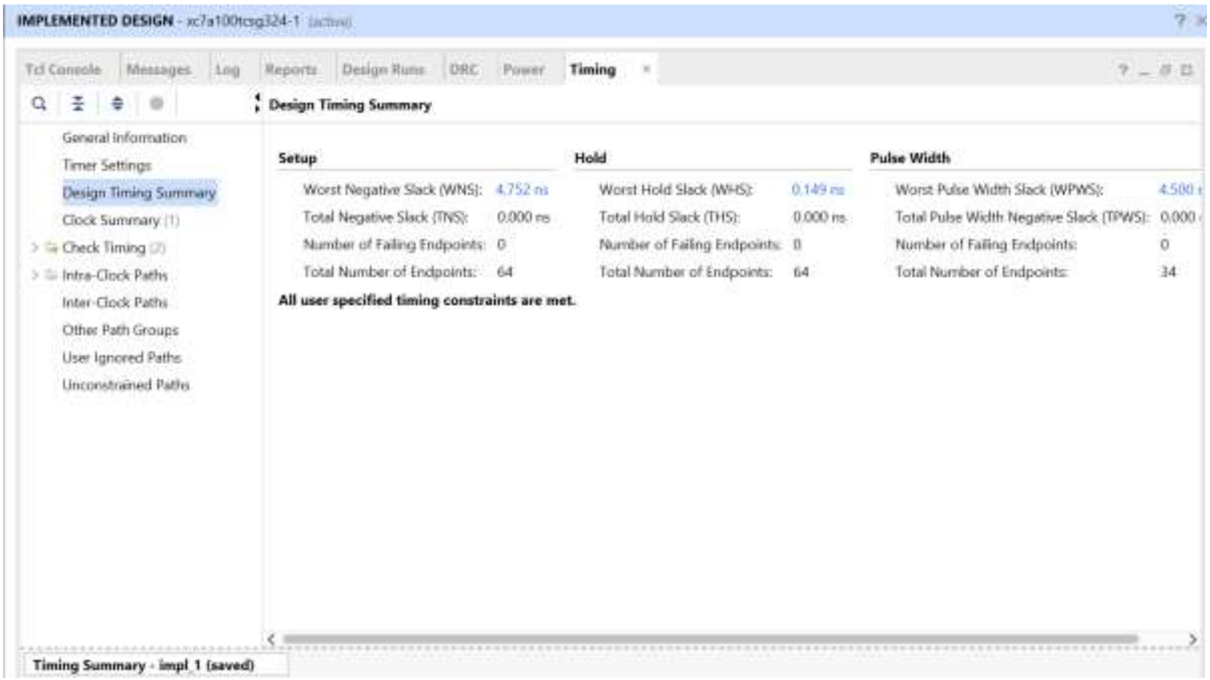
i. 22s
ii. 180s
iii. 3s
iv. 70s
v. 39s
vi. 12 minutes
vii. 2s

Resource utilisation:



Resource	Estimation	Available	Utilization...
LUT	25	63400	0.04
FF	47	126800	0.04
IO	7	210	3.33
BUFG	1	32	3.13

Timing constraints:



Design Timing Summary

Setup

Worst Negative Slack (WNS): 4.752 ns
Total Negative Slack (TNS): 0.000 ns
Number of Failing Endpoints: 0
Total Number of Endpoints: 64

Hold

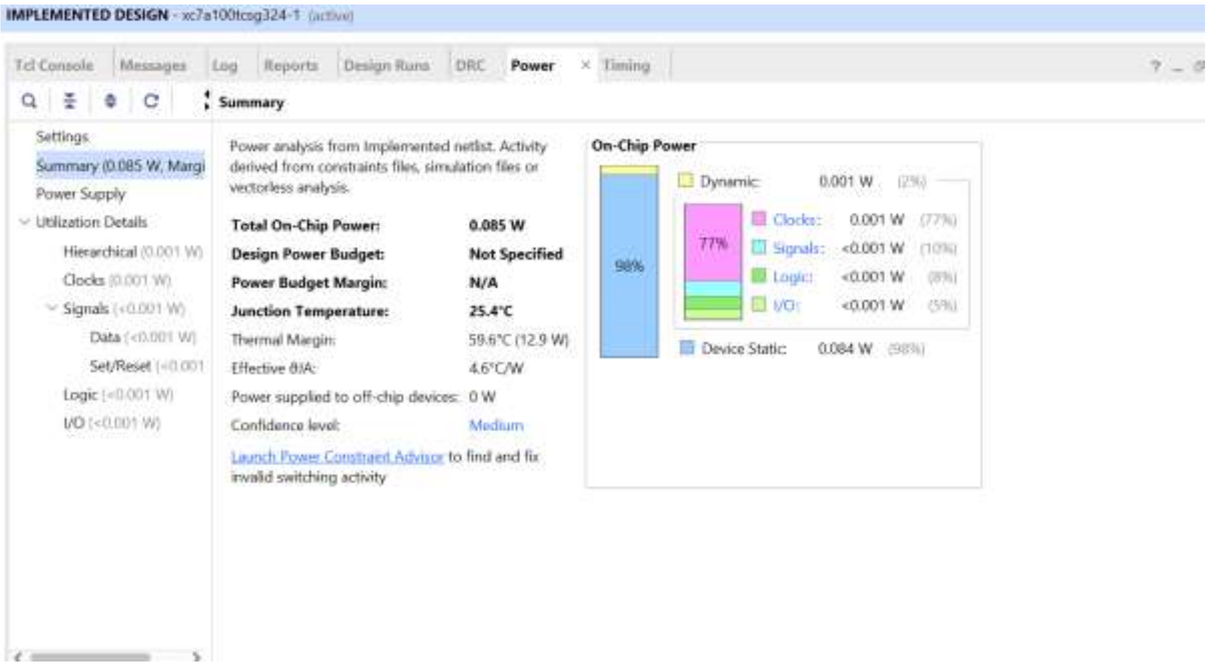
Worst Hold Slack (WHS): 0.149 ns
Total Hold Slack (THS): 0.000 ns
Number of Failing Endpoints: 0
Total Number of Endpoints: 64

Pulse Width

Worst Pulse Width Slack (WPWS): 4.500
Total Pulse Width Negative Slack (TPWS): 0.000
Number of Failing Endpoints: 0
Total Number of Endpoints: 34

All user specified timing constraints are met.

Power:



Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power: **0.085 W**
Design Power Budget: **Not Specified**
Power Budget Margin: **N/A**
Junction Temperature: **25.4°C**
Thermal Margin: 59.6°C (12.9 W)
Effective θJA: 4.6°C/W
Power supplied to off-chip devices: 0 W
Confidence level: Medium

Launch Power Constraint Advisor to find and fix invalid switching activity

On-Chip Power

Dynamic: 0.001 W (2%)
Clocks: 0.001 W (77%)
Signals: <0.001 W (10%)
Logic: <0.001 W (8%)
I/O: <0.001 W (5%)
Device Static: 0.084 W (98%)

CONCLUSION

Classroom education is one part of learning. However, the largest part lies in the application of the knowledge gained in the classroom. This can only be done by doing real-time projects. This internship helped me gain more practical knowledge apply it for real time projects. Not only did this internship make me realise the importance and use of Embedded Systems in real-life scenarios, but also taught me how to debug and test the devices. This internship also taught me time management and working independently .

Overall, this experience was a good one.

ACKNOWLEDGEMENT

I wish to thank Prof. Rajat Subhra Chakraborty for giving me this opportunity to intern at IIT Kharagpur and work on this project from which I learnt a lot. I would also like to thank my mentor Mr Sreekanth for his constant guidance. I would also like to thank my parents, for being my motivation to take up this internship; and last, but not the least, the faculty and management at Vellore Institute of Technology (VIT), Chennai, for providing me with such an avenue to help realize how interesting it is to work in today's industry.

Bibliography

1. ES-TRNG: A High-throughput, Low-area True Random Number Generator based on Edge Sampling Bohan Yang, Vladimir Rožić, Miloš Grujić, Nele Mentens and Ingrid Verbauwhede
2. Ultra-Compact UART Macros for Spartan-6, Virtex-6 and 7-Series with PicoBlaze (KCPSM6) Reference Designs
3. PicoBlaze for Spartan-6, Virtex-6, 7-Series, Zynq and UltraScale Devices (KCPSM6)
4. Nexys4 DDR™ FPGA Board Reference Manual
5. Xilinx Websites

*****End*******