apertus

Google Summer of Code

Google **S**ummer **o**f **C**ode 2019
Apertus Organization

Subject: Final Evaluation Documentation.

Submitted to: Apertus Organization.
Project name: **Bidirectional Packet Protocol for FPGA communication**

Student Information:
Name: Abd-ElRhman Sobhy Farahat
University: Cairo University - Faculty of Engineering - Computer Department (Egypt).
From: Cairo, Egypt.
Email: aabdosobhy@gmail.com
Code link: Bi-direction Packet Protocol.

Index:

1. Creating projects and design flow on Xilinx Vivado:

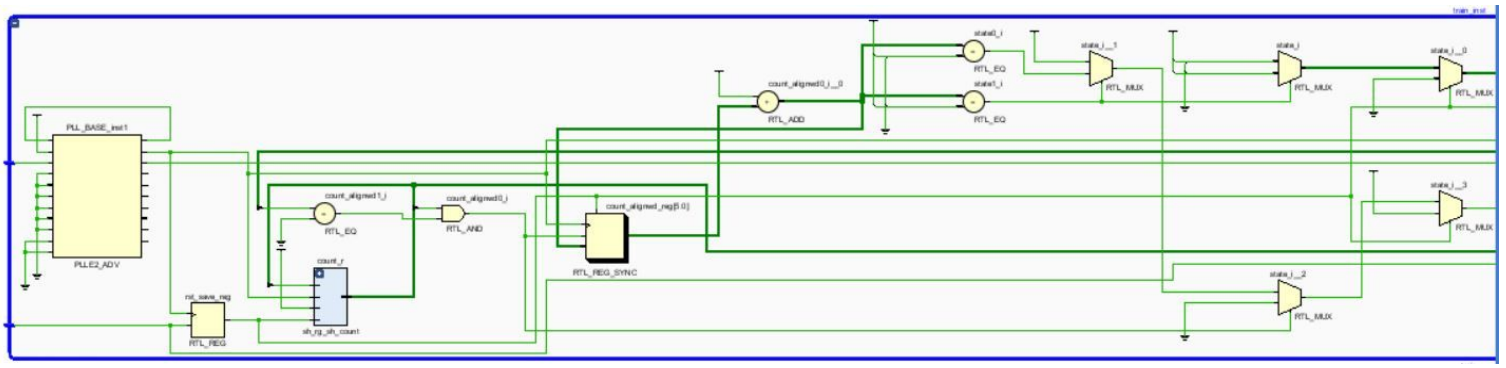
- Software version: Xilinx Vivado 2017.4
- Product Family: Zynq-7000
- Target Device: xc7z020clg400-1
- Board part: em.avnet.com:microzed_7020:part0:1.1
- Language: VHDL

2. Creating projects and design flow on Lattice Diamond:

- Software version: Lattice Diamond
- Product: MachXO2
- Target Device: LCMXO2-1200HC
- Package: “TQFP100”
- Grade: 6
- Language: VHDL

3. Link training between ZYNQ and MachXO2(ZYNQ side):

By considering the ZYNQ side it has 5 main modules:



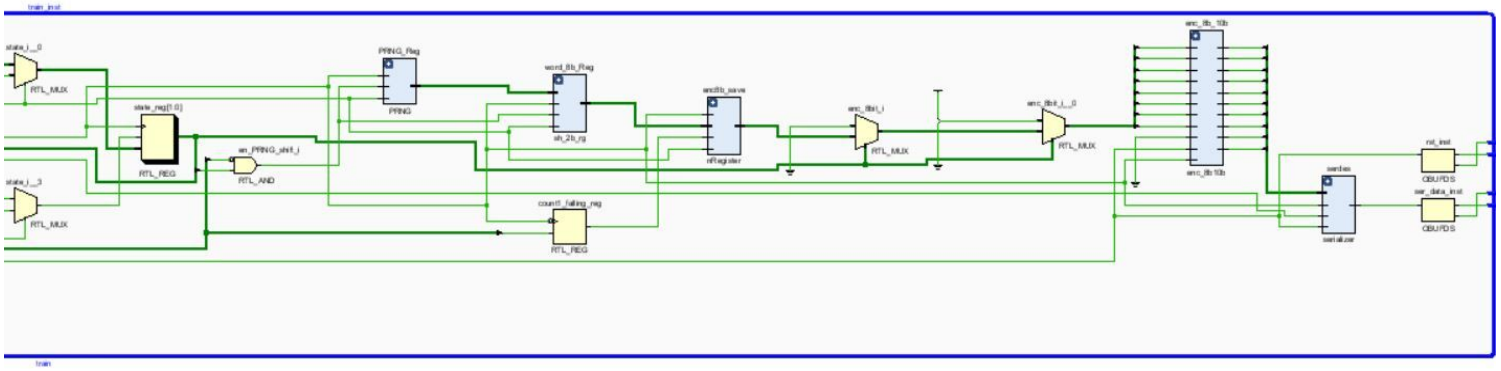


Fig 1. Schematic view - ZYNQ(1). (2)

1- **ps7_stub**: It is the interface from the PS of the ZYNQ to the PL, the used signal from it is:

ps_fclk: four clock signals from the PS, ps_fclk(0) -50 MHz- is the one is used in my module as the input clock to PLL2_Base.

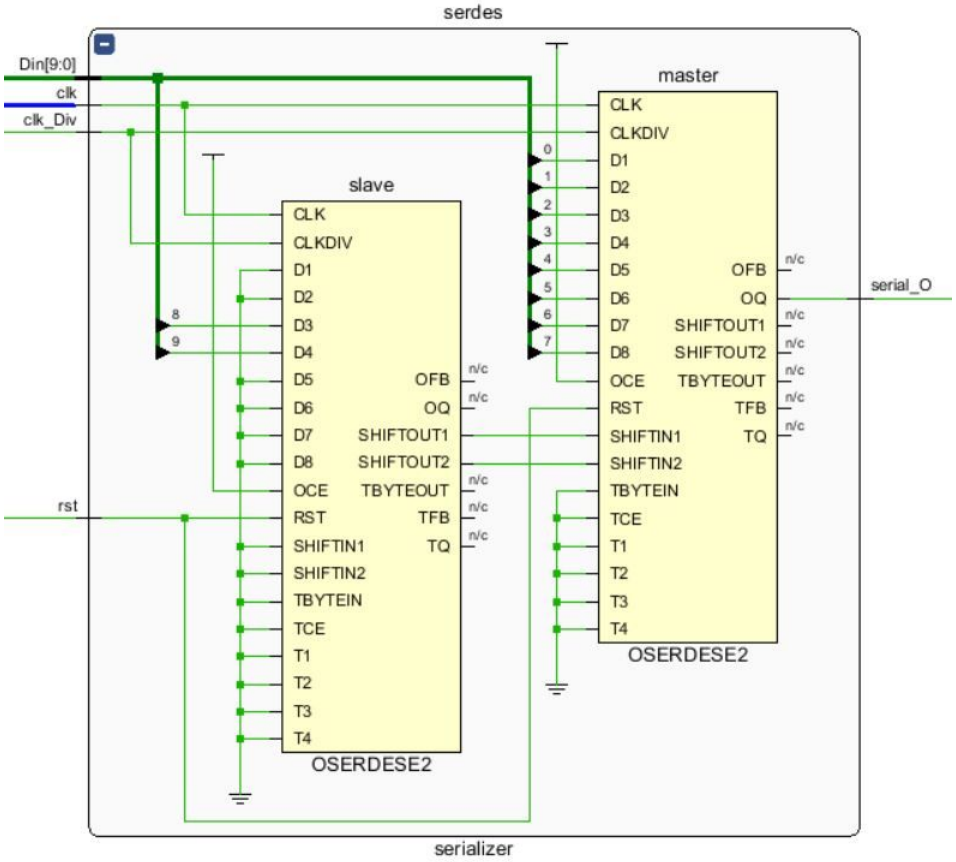
ps_reset_n: four reset signal routed to pl "FCLKRESETN[3:0]" which could be used as general purpose reset signals for PL logic, ps_reset(0) is the reset signal for both sides (ZYNQ & MachXO2).

i2c_**: i2c signals that are used to configure the machXO2 and read the values of PIC16 GPIO values.

2- **PRNG**: Pseudo-Random Number Generation which has a Fibonacci LFSR of polynomial 5.3.2.0 of the xor-ed bits with a seed of “11100111” that gives a 255 generated words and loops again. The only word that didn’t generate is “00000000” as its the reset value of the receiver, sending it once the rest signal of Transmitter is fired(after encoding).

3- **enc_8b_10b**: Its an open source code based on IBM encoding that encodes the 8 bits generated from PRNG into 10 bits that have a balanced no. of zeros and ones.

4- **Serializer**: a pair of OSERDESE2 primitives are used in a master & slave configuration in DDR mode for 10:1 serialization ratio, it takes the 10 bits encoded from 8b_10b as input to be serialized.



5- **PLL2_BASE**: generates the two main clocks at sender's side, the clocks are:

clk_50: the faster clock -50 MHz- that most of the modules use.

clk_10: the sampling clock -10 MHz-that captures the encoded value every 5 cycles of clk_50.

Rest of modules and signals:

- **OBUFDS**: The output of the LVDS signals used for sending reset and serial data by two pairs of LVDS signal

- **count_r**: synchronous counter of size 5 bits that shifts a value of '1' every clk cycle that enables PRNG to shift the new value at the first 4 counts and disable it for 1 cycles to calculate the encoded 10 bits.

- **rst_save_reg**: register of 1 bit that save the value of the input reset to my module to be last sampled by the sampling clock clk_10.
- **sh_2b_rg:** shift register that shift more than 1 bit at the rising edge of clk_50, that will act like ddr registers in my module as It shifts 2 bits every rising edge.
- **enc8b_save**: ordinary register that captures the PRNG data to be not changed until the encoding finishes, and refreshes its value every

Logic Description:

After getting the input clk and reset from ps7 instant and prepare the two clocks that training module will run with it. The train module has three states described as follows:

- state "00" : Word alignment state. Transmitting a same word "11110000" -before encoding- for 40 times.

- Stae “01”: reset word. Transmit the word “00000000” -before encoding- that is the reset signal for the receiver side that reset some modules to start BER .
- State “1x” : transmitting the PRNG value forever.

Analysis of Design:

Timing analysis:

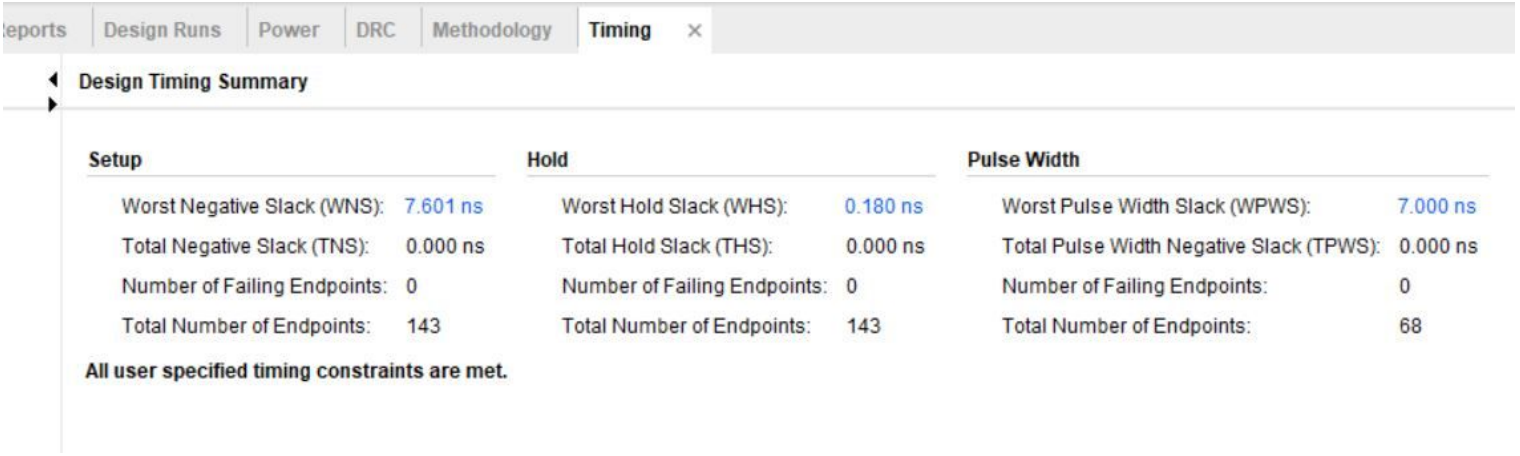


Fig 3. Time analysis - ZYNQ.

Power Analysis:

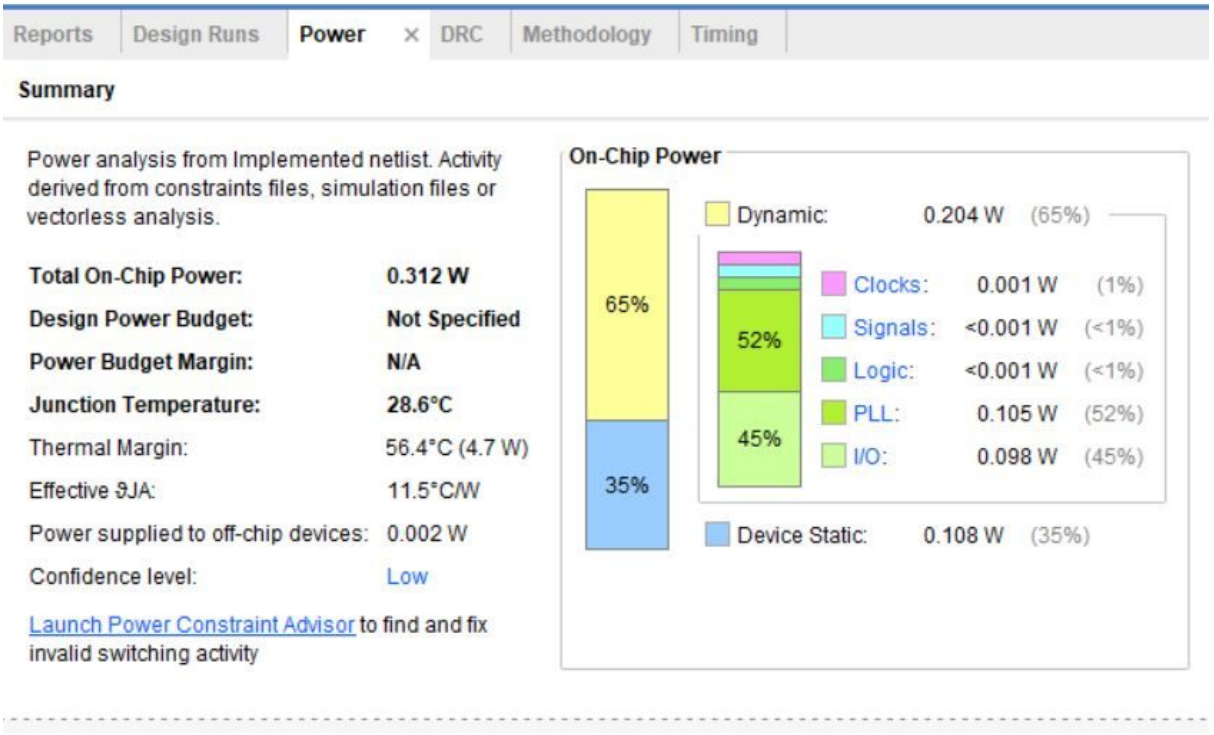


Fig 4. Power analysis - ZYNQ.

4. Link training between ZYNQ and MachXO2(MachXO2 side):
   By considering the MachXO2 side it has 7 main modules:

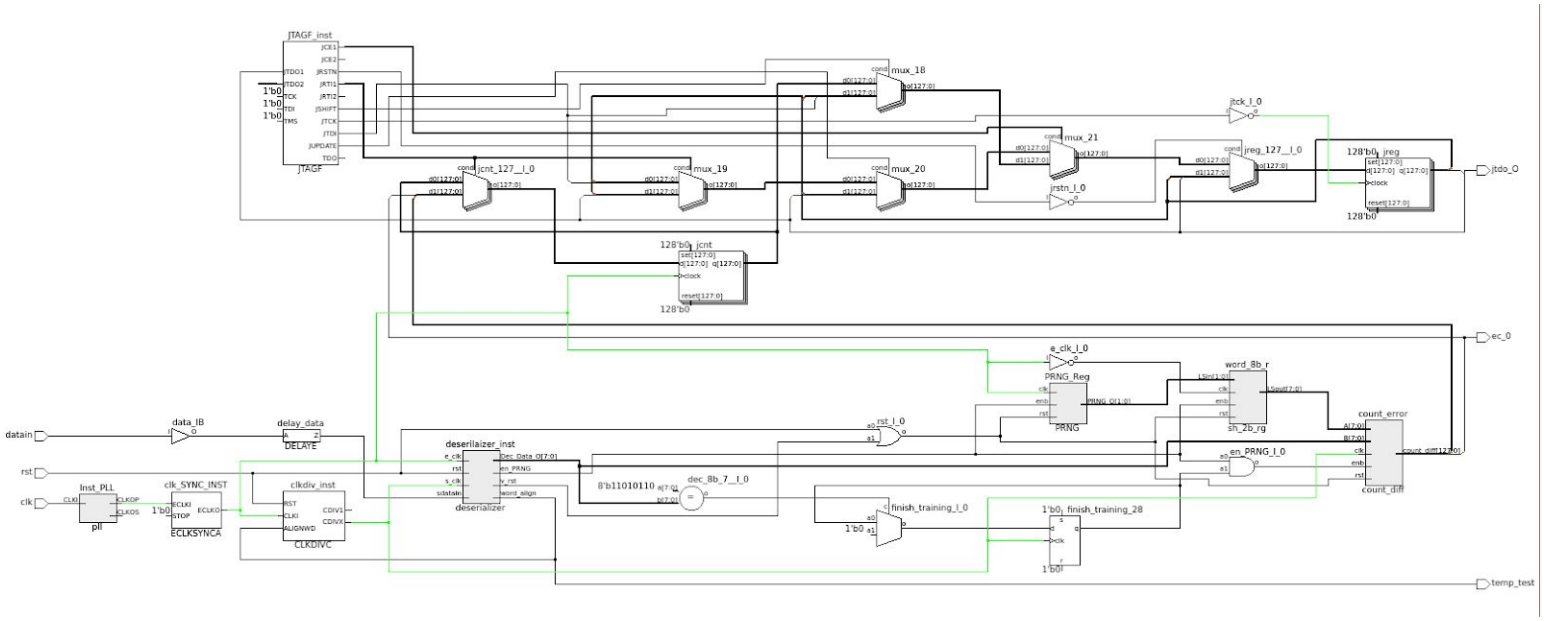


Fig 5. Schematic view of MachXO2.

1. **pll**: pll module that generated from IPexpress from diamond lattice that has an input of clock 50 MHZ and generate a clock with the same frequency. Its main purpose is to route the clock signal to e_clk.

2. **CLKDIVC**: generates the sampling clock of 12.5 MHz that will be the s_clk for sampling (4 * e_clk = 1 s_clk).

3. **IDDRX4B**: receives the input data and generate 8 bits every 4 cycles of e_clk that will be the received data word.

4. **dec_8b10b**: Its an open source code based on IBM encoding that decodes the 10 bits received from LVDS (transmitter) into the original

8 bits.

5. **PRNG**: Pseudo-Random Number Generation which has a Fibonacci LFSR of polynomial 5.3.2.0 of the xor-ed bits with a seed of “11100111” that gives a 255 generated words and loops again. The only word that didn’t generate is “00000000” as its the reset value of the receiver.

6. **Count_diff**: compare the decoded value from dec_8b10b with the word generated from RPNG. count the difference bits and counting that result to every word will lead to the BER result.

7. **JTAGF**: JTAG interface that enable us to communicate with the MachXO2 with JTAG protocol. JTAGF has 2 private instructions, ER1 (0x32) and ER2 (0x38). If the ER1 instruction is shifted into the JTAG instruction register, JRTI1 will go high when the TAP controller is in the Run-Test/Idle state. The same as ER2. ER1 instruction in my module is responsible for reading the BER register value after transmit the corresponding .

Rest of modules and signals:

- DELAYE: delay module for the serial data input to be center aligned with the e_clk.

- ECLKSYNCA: element is associated with the ECLK and must be used to drive the ECLK.

- sh_2b_rg: shift register that shift more than 1 bit at the rising edge of e_clk, that will act like ddr registers in my module as It shifts 2 bits every rising edge.

- IB: Input buffer for the serial data input.

Logic Description:

First let's describe how the word is received:

The IDDRX4B receives 8-bit word every s_clk cycle by concatenate 5 words together will generate 40 bits that's equal to 4 encoded words the transmitter has transmitted. So there are 5 states in receiving the word in order to generate the 10 bits to be decoded.

- state 0: the received word is 8 bits so their can’t be decoded pausing the PRNG at this state and not count the difference from the PRNG and decoder at this state.
- State 1: the received word became 16 bits so decoding the first 10 bits will be possible.
- State 2, 3, 4, same as state 1 as their are a 10 bits available to be decoded.

Second the process of training:

The process of training is based on the received serial data that control the current phase:

- Word alignment phase: after resetting the MachXO2 module it will remain in the word alignment phase until the transmitter brake that by sending the reset value “00000000” after decoding.
  Word alignment signal is fired at state 2 in receiving the words that will make the IDDRX4B to shift the sampling edge by 1 bit, after 8 times from firing that signal the sampling edge will return to the first position. So by sending 40 words of “11110000” alignment-word will lead the sampling point to return back to its first position 2 times.

- BER counting phase: after receiving the reset value “00000000” -after decoding- the train module will be in counting BER for each decoded word comparing it with the word generated from the PRNG and adding the result until the final word “11010110” that will pause the BER register to count more.

- BER final result: after the 2 past phases done the BER register would be ready to read by JTAGF using the private instruction ER1.

Analysis of Design:

Timing analysis:

	A	B	C	D	E	F	G	H	I	J	K	L
1	Device Family	MachXO2										
2	Device	LCMXO2-1200HC										
3	Package	TQFP100										
4	Setup Performance Grade	Default										
5	Hold Performance Grade	Default										
6	Preference File	/home/aabdosobhy/finalrun/Untitled.tpf										
7	Check Unconstrained Connections	No										
8	Check Unconstrained Paths	No										
9	Report Asynchronous Timing Loops	No										
10	Report Style	Verbose Timing Report										
11	Full Name	No										
12	Clock Domain	No										
13	Worst-Case Paths	10										
14	Number of Unconstrained Paths	0										
15												
16												
17	Preference	FREQUENCY NET "clk_50" 50.000000 MHz										
18												
19												
20	Preference											
21												
22												
23	Delay Table											
24												
25	Source	Destination	Weighted Slack	Arrival	Required	Data Delay	Route %	Levels	Clock Skew	Setup/Hold	Jitter	Color
26	deserilaizer_inst/decoderIn_i7	deserilaizer_inst/decoder_10b_8b/CO_225	5.628	7.186	10	6.685	64	6	0.368	0.133	0	
27	deserilaizer_inst/decoderIn_i8	deserilaizer_inst/decoder_10b_8b/CO_225	6.224	6.888	10	6.755	64.4	6	0	0.133	0	
28	deserilaizer_inst/decoderIn_i6	deserilaizer_inst/decoder_10b_8b/DO_226	6.254	6.873	10	6.372	68.6	5	0.368	0.133	0	
29	deserilaizer_inst/decoderIn_i6	deserilaizer_inst/decoder_10b_8b/CO_225	6.29	6.855	10	6.354	62.1	6	0.368	0.133	0	
30	deserilaizer_inst/decoderIn_i7	deserilaizer_inst/decoder_10b_8b/DO_226	6.384	6.808	10	6.307	61.8	6	0.368	0.133	0	
31	deserilaizer_inst/decoderIn_i7	deserilaizer_inst/decoder_10b_8b/BO_224	6.388	6.806	10	6.305	61.8	6	0.368	0.133	0	
32	deserilaizer_inst/decoderIn_i7	deserilaizer_inst/decoder_10b_8b/CO_225	6.448	6.776	10	6.275	61.6	6	0.368	0.133	0	
33	deserilaizer_inst/decoderIn_i8	deserilaizer_inst/decoder_10b_8b/CO_225	6.554	6.723	10	6.59	63.5	6	0	0.133	0	
34	deserilaizer_inst/decoderIn_i7	deserilaizer_inst/decoder_10b_8b/CO_225	6.602	6.699	10	6.198	61.2	6	0.368	0.133	0	
35	deserilaizer_inst/decoderIn_i7	deserilaizer_inst/decoder_10b_8b/BO_224	6.602	6.699	10	6.198	61.2	6	0.368	0.133	0	
36												

38	Data Path Details					
39						
40	Name	Fanout	Delay	Source	Destination	Resource
41	REG_DEL	0	0.367	R8C9C.CLK	R8C9C.Q0	deserilaizer_inst/SLICE_13
42	ROUTE	16	1.197	R8C9C.Q0	R9C10C.A1	deserilaizer_inst/decoderIn_7
43	CTOF_DEL	0	0.408	R9C10C.A1	R9C10C.F1	deserilaizer_inst/SLICE_126
44	ROUTE	1	0.761	R9C10C.F1	R9C10B.A1	deserilaizer_inst/decoder_10b_8b/n4_adj_1042
45	CTOF_DEL	0	0.408	R9C10B.A1	R9C10B.F1	deserilaizer_inst/SLICE_115
46	ROUTE	5	0.508	R9C10B.F1	R9C9B.D1	deserilaizer_inst/decoder_10b_8b/n2683
47	CTOF_DEL	0	0.408	R9C9B.D1	R9C9B.F1	deserilaizer_inst/SLICE_121
48	ROUTE	2	0.359	R9C9B.F1	R9C9B.C0	deserilaizer_inst/decoder_10b_8b/n3670
49	CTOF_DEL	0	0.408	R9C9B.C0	R9C9B.F0	deserilaizer_inst/SLICE_121
50	ROUTE	1	1.453	R9C9B.F0	R8C11D.B0	deserilaizer_inst/decoder_10b_8b/n3497
51	CTOF_DEL	0	0.408	R8C11D.B0	R8C11D.F0	deserilaizer_inst/decoder_10b_8b/SLICE_1
52	ROUTE	1	0	R8C11D.F0	R8C11D.DI0	deserilaizer_inst/decoder_10b_8b/CO_N_530
53			Total=6.685			
54						
55						
56	Source Clock					
57						
58	Name	Fanout	Delay	Source	Destination	Resource
59	ROUTE	42	1.714	BECLKSYNC0.ECLKO	R8C9C.CLK	e_clk
60			Total=1.714			
61						
62						
63	Destination Clock					
64						
65	Name	Fanout	Delay	Source	Destination	Resource
66	ROUTE	42	1.346	BECLKSYNC0.ECLKO	R8C11D.CLK	e_clk
67			Total=1.346			
68						

Fig 6. Time analysis -MachXO2(1). (2)

Power Analysis:

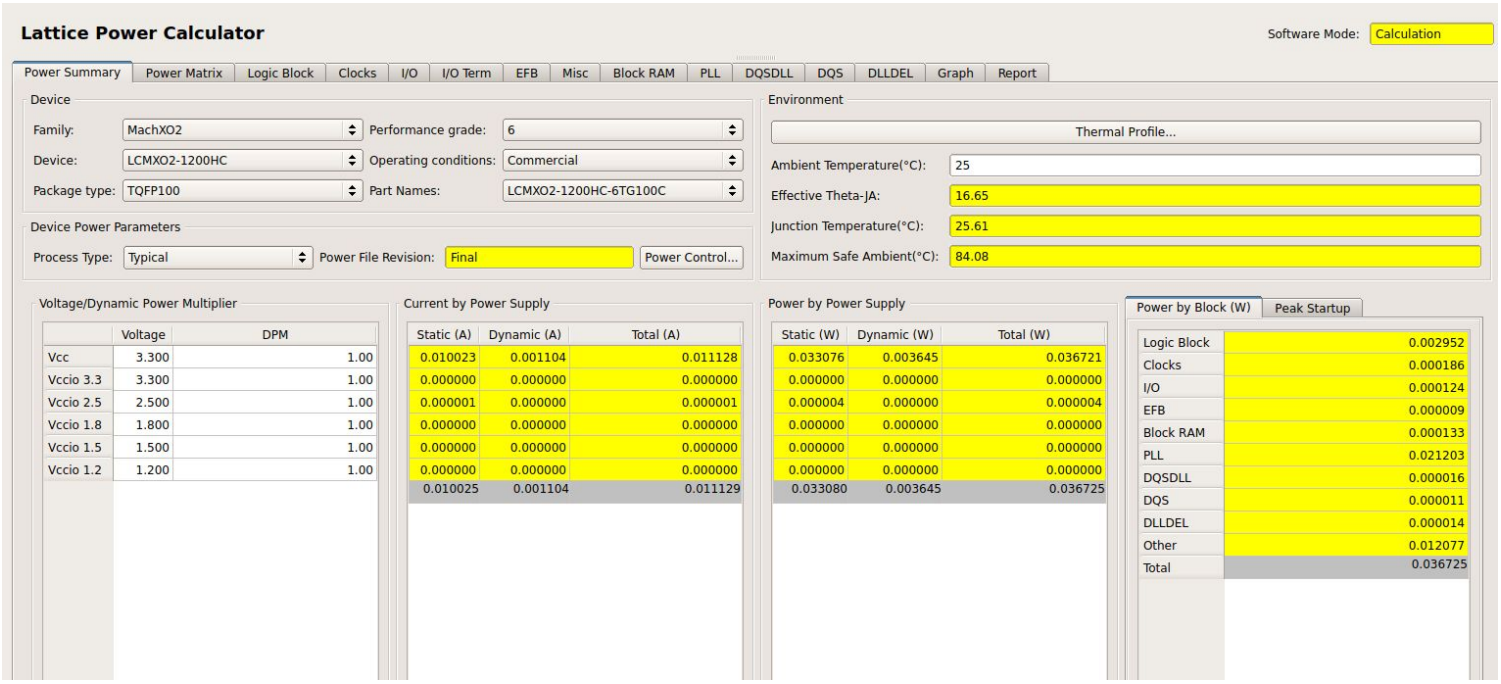


Fig 7. Power analysis - MachXO2.

4. Running Link Training on beta_e hardware (assuming you have access on beta_e HW):

- Transfer the two files *.bit that generated from running both make file for MachXO2 and TCL file for ZYNQ side.
- From terminal on beta_e run the following commands:
    - ./prep_icsp.sh
        Initialize the power, PS clocks, GPIO and I2C.
    - ./prep_train_lvds.sh
        Load the bit file of training on ZYNQ side
    - ./rf_sel.py B
        Select the PIC16 that connected to the RFE(Routing Fabric East).
    - ./pic_jtag_bitload.py /root/train_lvds_ps.bit
        Load the bit file of training to the MachXO2 using the help of PIC16.
    - devmem 0xF8000240 32 0xF
        Set the rst signal = ‘0’ as it is negated.
    - devmem 0xF8000240 32 0x0
        Fire the rst signal = ‘1’.
    - devmem 0xF8000240 32 0xF
        Set the rst signal = ’0’ so the rst signal is fired
    - ./pic_jtag_er1.py
        Read the final BER register.