FINAL PROJECT

Powering (x^y) Architecture Using Hyperbolic CORDIC

Abdulraheem **Al**J**J**arrah
Karam **A**bu**G**halieh

Instructor: Daniel Llamocca

Reconfigurable Computing
ECE 5900 – Fall 2017

OUTLINE:

- Introduction
- Hyperbolic CORDIC
- Powering x^y
- Powering x^y Implementation using Hyperbolic CORDIC
- AXI4, FIFOs and Power Interface
- Simulation
- Experimental Setup
- Results

❖ Introduction

- Powering is broadly used in wide rang of fields like economics, biology, chemistry, physics, and computer science, with applications such as compound interest, population growth, chemical reaction kinetics, wave behavior, and public-key cryptography.

- Powering can be computed using hyperbolic CORDIC which can be build using shifters adders and LUTs.

- The high cost of powering computation in addition to showing our deep understanding of problem solving and ability to design embedded systems design using FPGA were the motivations behind this project.

EXPANDED HYPERBOLIC CORDIC

- Hyperbolic CORDIC is used to compute hyperbolic functions in efficient and fast way

- In the expanded CORDIC the index of iterations will be from –M to N, two sets of equation for positive and negative iterations.

$$i \le 0: \begin{cases} x_{i+1} = x_i + \delta_i y_i (1 - 2^{i-2}) \\ y_{i+1} = y_i + \delta_i x_i (1 - 2^{i-2}) \\ z_{i+1} = z_i - \delta_i \theta_i, \quad \theta_i = Tanh^{-1}(1 - 2^{i-2}) \end{cases}$$

$$i > 0: \begin{cases} x_{i+1} = x_i + \delta_i y_i 2^{-i} \\ y_{i+1} = y_i + \delta_i x_i 2^{-i} \\ z_{i+1} = z_i - \delta_i \theta_i, \quad \theta_i = Tanh^{-1}(2^{-i}) \end{cases}$$

- Iterations with i=4,13, ... k, 3k+1 must be repeated to guarantee convergence

$$Rotation: \quad \delta_i = -1 \; if \; z_i < 0; \; +1 \, , otherwise$$
$$Vectoring: \quad \delta_i = -1 \; if \; x_i y_i \ge 0; \; +1 \, , otherwise$$

EXPANDED HYPERBOLIC CORDIC

- x_n, y_n, z_n converges to the following

$$Rotating: \begin{cases} x_n = A_n(x_0 coshz_0 + y_0 sinhz_0) \\ y_n = A_n(y_0 coshz_0 + x_0 sinhz_0) \\ z_n = 0 \end{cases}$$

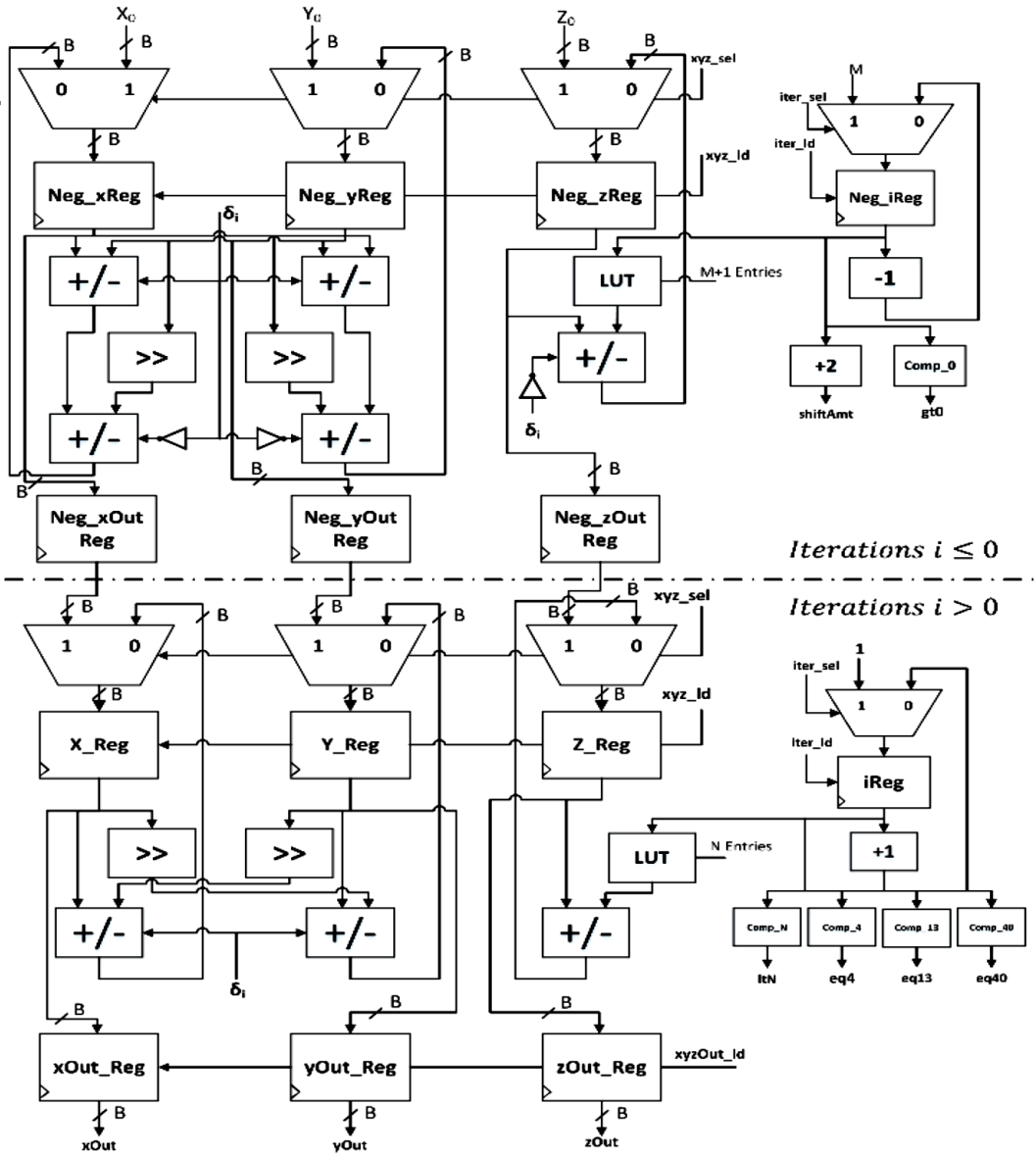
$$Vectoring: \begin{cases} A_n\sqrt{x_{in}^2 + y_{in}^2} \\ y_n = 0 \\ z_n = z_{in} + Tanh^{-1}\dfrac{y_{in}}{x_{in}} \end{cases}$$

- Where A_n is defined as:

$$A_n = (\prod_i^0 = -M\sqrt{1-(1-2^{i-2})^2}\,)\prod_i^N \sqrt{1-2^{-2i}}$$

EXPANDED HYPERBOLIC CORDIC

- Two Stages
- FSM & LUT For each state
- B is 32 bit floating point format



❖ Powering x^y

- The goal is to compute x^y , which could be achieved by getting e^{ylnx}
- Two functions required ln x and exp(x)
  - ln x is defined as: $\ln x = 2\tanh^{-1}\frac{x-1}{x+1}$
  - By setting $x_0 = x + 1, y_0 = x - 1, z_0 = 0$, in vectoring mode the output z_n converges to $\frac{1}{2}lnx$
  - Exp(x) is defined as: $e^x = \cosh x + sinh\, x$
  - By setting $x_0 = y_0 = \frac{1}{A_n}, z_0 = x$, in rotating x_n converges to e^x

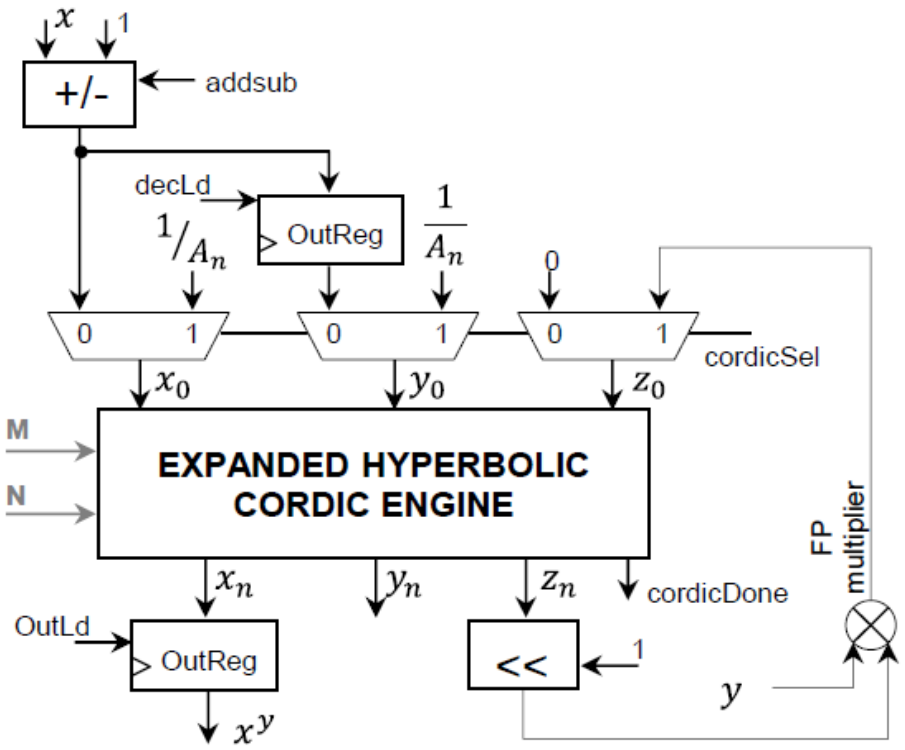
❖ Powering x^y

- Required input to get x^y
  - By setting $x_0 = x + 1, y_0 = x - 1, z_0 = 0$, in vectoring mode the output z_n converges to $\frac{1}{2} lnx$
  - Multiply $\frac{1}{2} lnx$ by 2y to $ylnx$
  - By setting $x_0 = y_0 = \frac{1}{A_n}, z_0 = ylnx$, in rotating mode x_n converges to x^y

❖ Powering x^y Implementation using Hyperbolic CORDIC

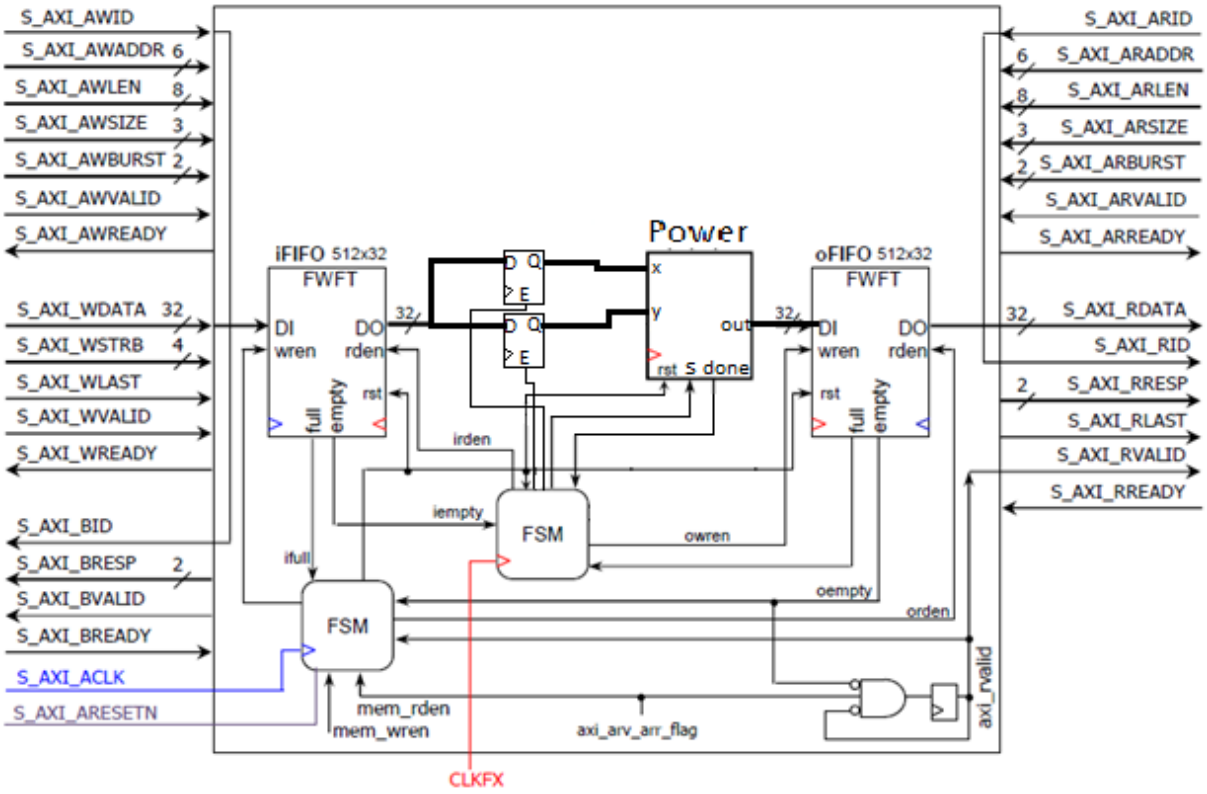
Expanded CORDIC twice as in the following steps:

1. Use vectoring mode to get $z_n = \ln x/2$ by loading $x_0 = x + 1, y_0 = x - 1, z_0 = 0$, this is done using the addsub unit on the top left corner and the multiplexers at the input stage.
2. To get $yln\,x$, first z_n is multiplied by 2 using the shifter , then the floating point multiplier is used to multiply $2z_n$ output by y to get $yln\,x$.
3. Use the current output of z_n as the next input to the CORDIC block in rotating mode. With $x_0 = y_0 = 1/A_n$, this is again done using the multiplexers at the input stages. The output $x_n = e^{ylnx} = x^y$ is obtained in OutReg.



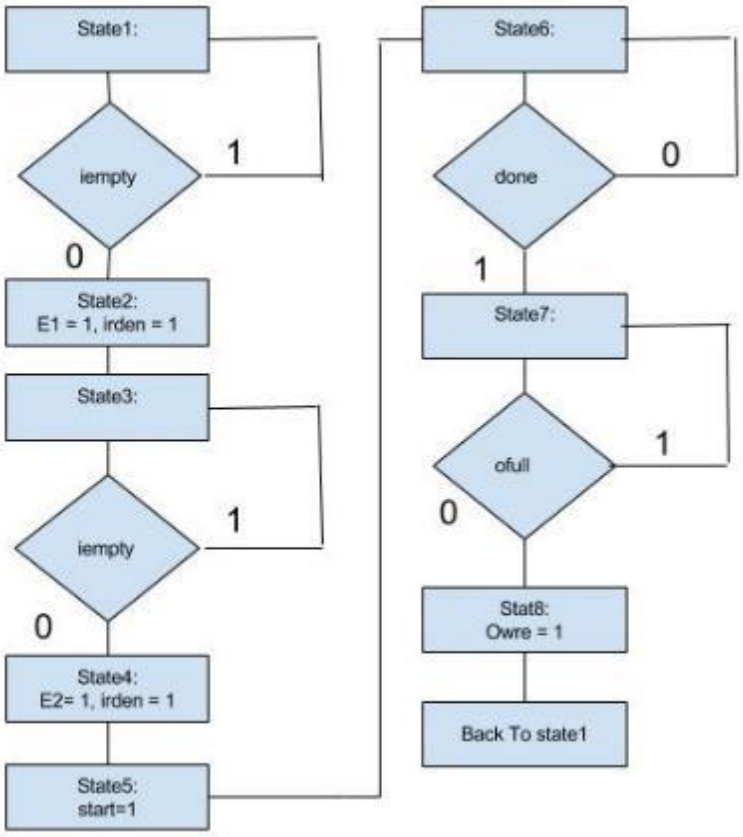
AXI4,FIFO and Power Interface

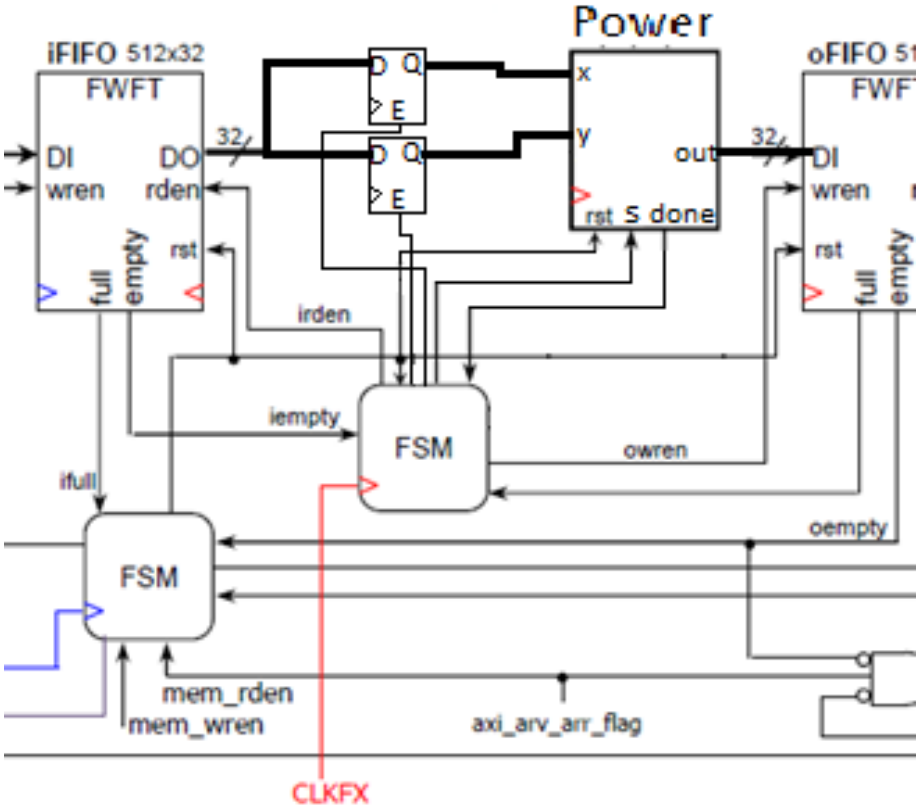
- We have two 32 bit words (x and y)
- FIFOs are used in the AXI interface
- After the FIFIOs we do have registers to capture the data once its available.



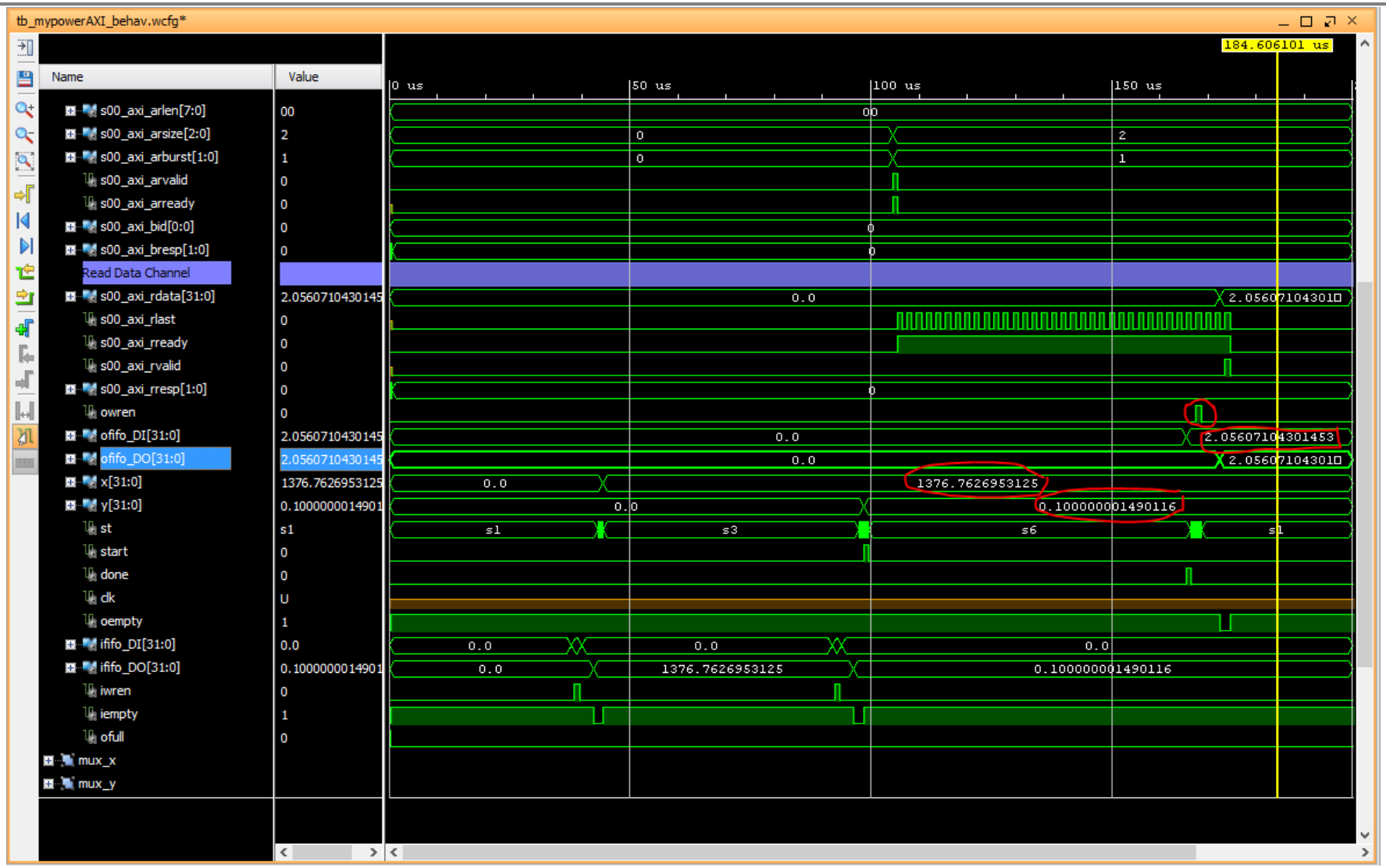
AXI4,FIFO and Power Interface

- Blue FSM stay the same
- Red FSM was modified as follows

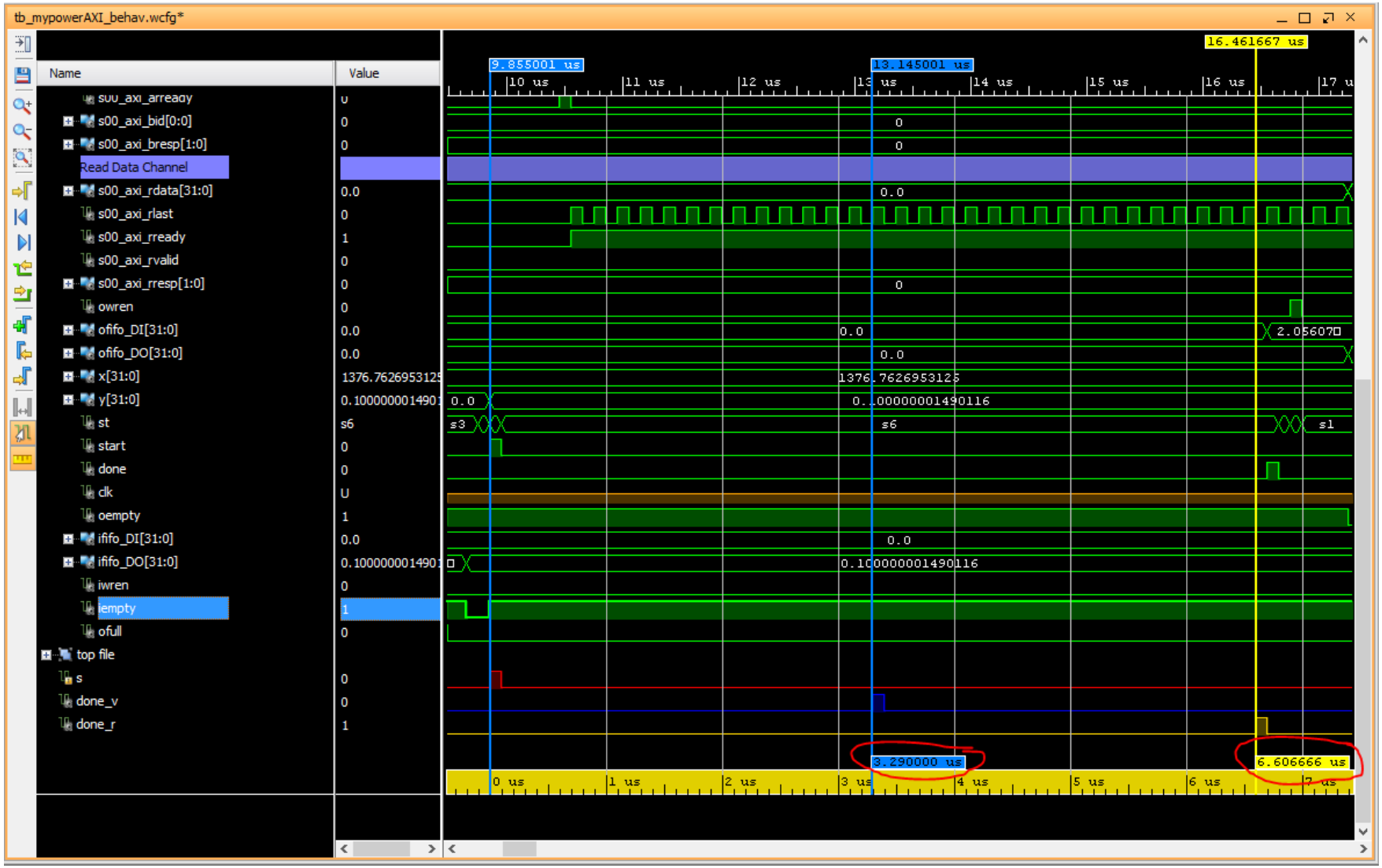




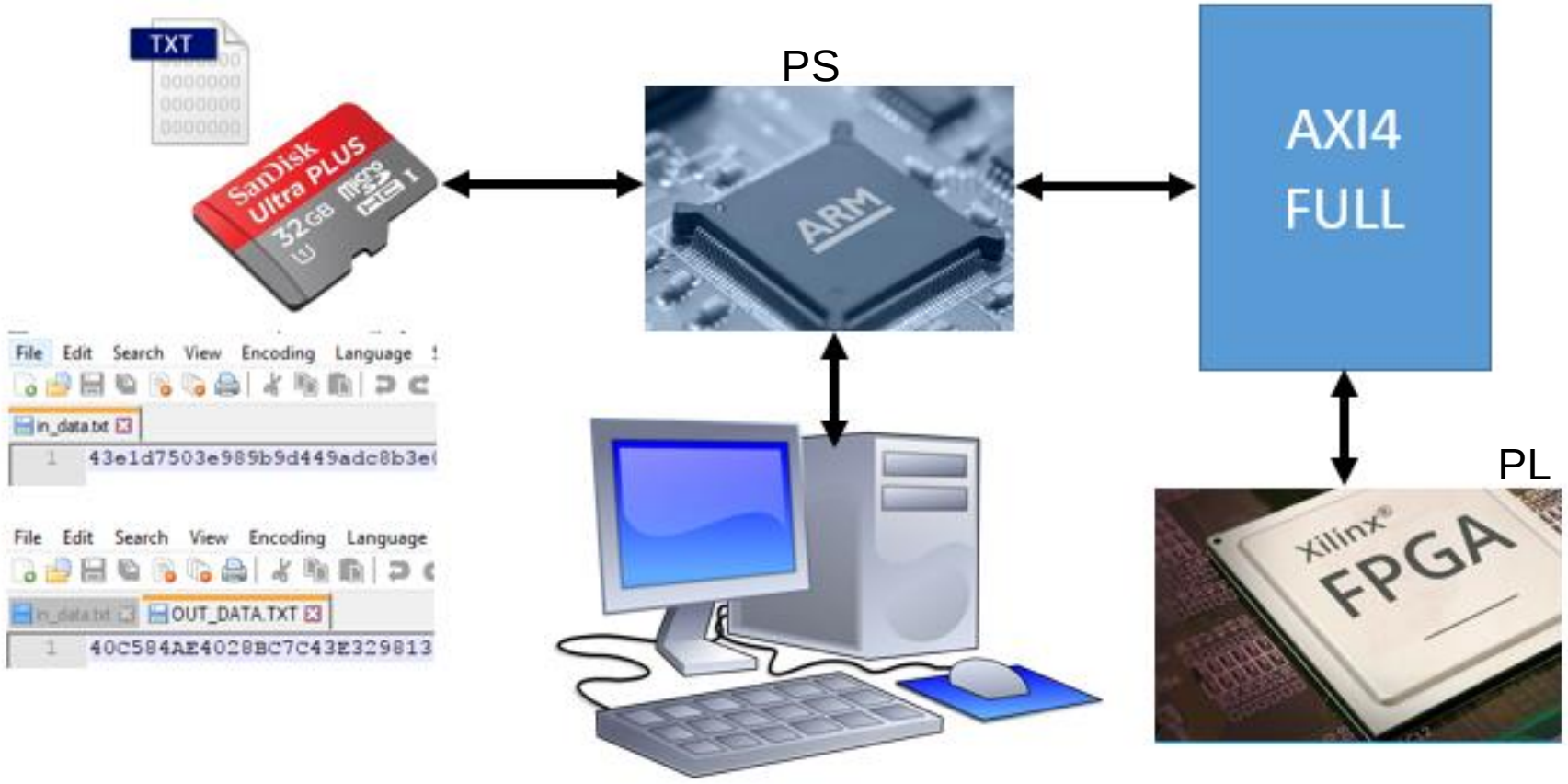
Simulation with AXI



Simulation

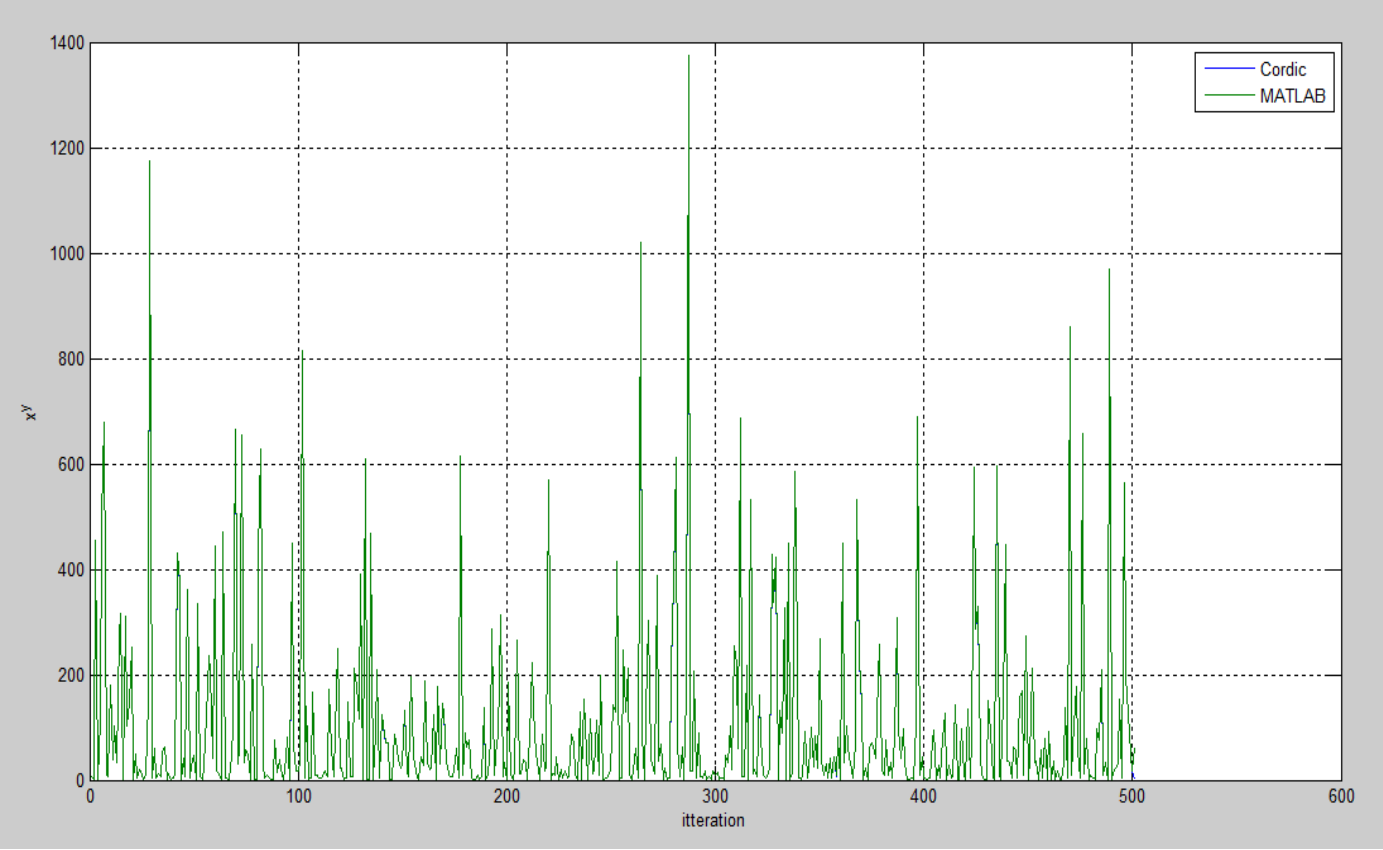

tb_mypowerAXI_behav.wcfg*
Name
Value
s00_axi_arready
s00_axi_bid[0:0]
s00_axi_bresp[1:0]
Read Data Channel
s00_axi_rdata[31:0]
s00_axi_rlast
s00_axi_rready
s00_axi_rvalid
s00_axi_rresp[1:0]
owren
ofifo_DI[31:0]
ofifo_DO[31:0]
x[31:0]
y[31:0]
st
start
done
clk
oempty
ififo_DI[31:0]
ififo_DO[31:0]
iwren
iempty
ofull
top file
s
done_v
done_r
9.855001 us
13.145001 us
16.461667 us
3.290000 us
6.606666 us

Experimental Setup



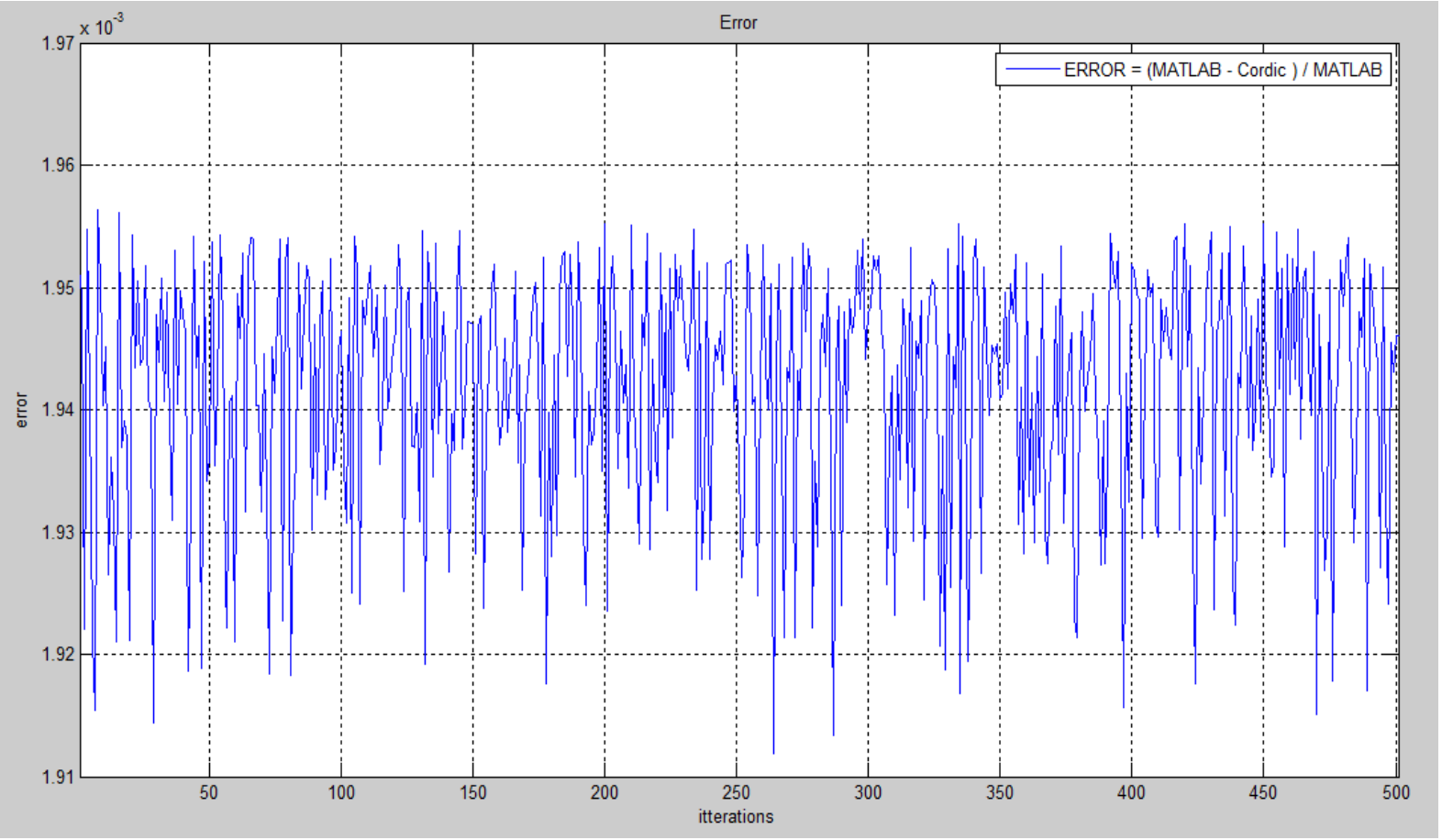
Results

- Random variables were chosen for x and y (using rand function)
- 0<= x <=1500 and 0<= y <= 1
- 500 data set were generated



Results

- The error relative error = (MATLAB - Cordic) / MATLAB



Demo

Thank You

- Any Questions??