Giaan Nguyen

Lab 2: CORDIC Circuit Design on FPGAs

ECEN 689-600: FPGA Information Processing Systems
Friday, February 12, 2021

CORDIC Polar Simulation

Figure 1 shows the results for simulating CORDIC calculations of $\cos\left(\frac{\pi}{3}\right)$ and $\sin\left(\frac{\pi}{3}\right)$ within the first 50 nanoseconds, followed by $\tan^{-1}(1)$ for the latter 50 nanoseconds. For the cosine/sine mode, we assume $x_0 = 1$ and $y_0 = 0$, i.e., taking our initial angle to be 0 radians; we input $z_0 = \frac{\pi}{3}$ as our target angle. Inversely for the arctan mode, we input initial angle $z_0 = 0$ radians; assuming the unit circle, we let $x_0 = y_0 = \frac{\sqrt{2}}{2}$.

Using 7 iterations starting at index $k = 0$, we see around the 40-ns mark are the computed values for the cosine/sine mode. Taking $G = 1.64676$, we should see the following results:

$$x = G(x_0 \cos z_0 - y_0 \sin z_0) = 1.64676\left(1 \times \frac{1}{2} - 0 \times \frac{\sqrt{3}}{2}\right) = 0.82338$$

$$y = G(x_0 \sin z_0 - y_0 \cos z_0) = 1.64676\left(1 \times \frac{\sqrt{3}}{2} - 0 \times \frac{1}{2}\right) = 1.42613$$

$$z = 0$$

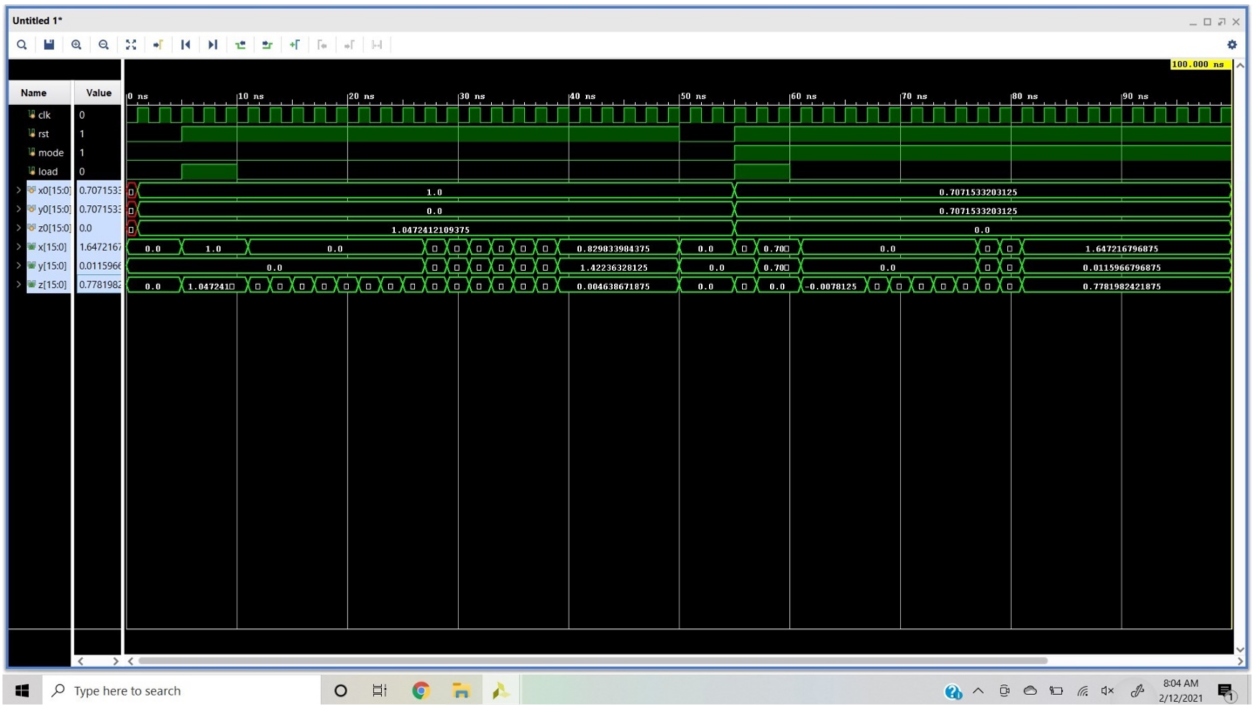


Figure 1. Simulation for CORDIC polar design. Cosine/sine mode is simulated within the first half, followed by arctan mode.

Similarly, for the arctan mode, around the 85-ns mark, we should see the following results:

$$x = G\sqrt{x_0^2 + y_0^2} = 1.64676\sqrt{\left(\frac{\sqrt{2}}{2}\right)^2 + \left(\frac{\sqrt{2}}{2}\right)^2} = 1.64676$$

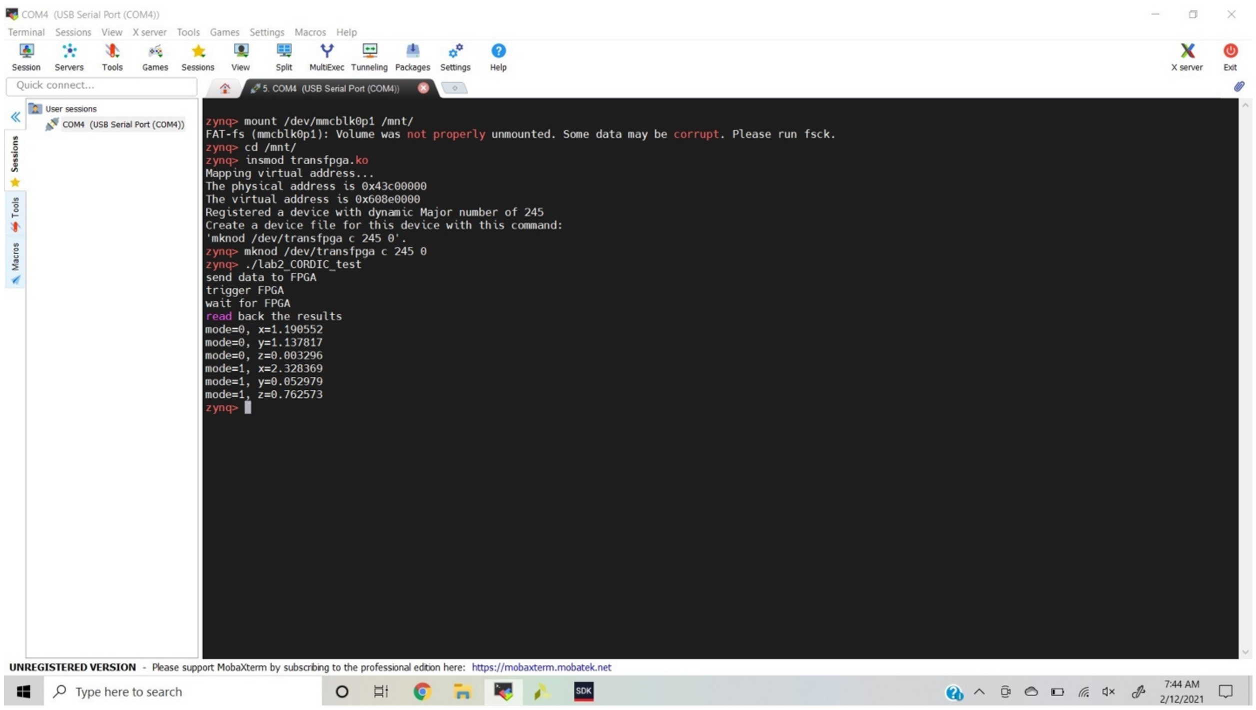
$$y = 0$$

$$z = z_0 + \tan^{-1}\left(\frac{y_0}{x_0}\right) = 0 + \tan^{-1}(1) = 0.78539$$

For both modes, we see that the simulated outputs are within 0.05 error from the equations' numerical solutions, verifying that the design should work.

CORDIC Polar FPGA Outputs

Figure 2 shows the results when implementing the CORDIC polar design on the FPGA, as seen in the terminal. Interestingly, cosine/sine mode outputs largely different results than the simulation, yet the arctan mode is able to generate similar values to the simulation, assuming the top module uses initial values of $x_0 = y_0 = 1$ for the arctan instead of coordinates along the unit circle.



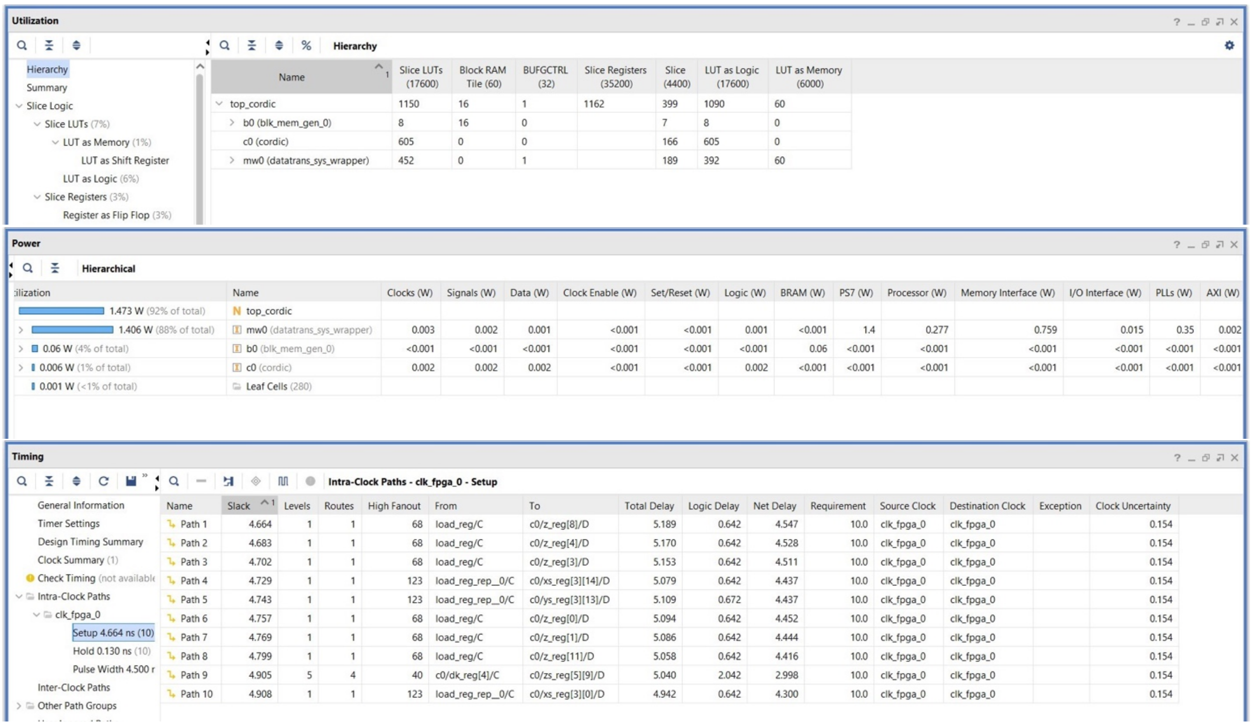
```
zynq> mount /dev/mmcblk0p1 /mnt/
FAT-fs (mmcblk0p1): Volume was not properly unmounted. Some data may be corrupt. Please run fsck.
zynq> cd /mnt/
zynq> insmod transfpga.ko
Mapping virtual address...
The physical address is 0x43c00000
The virtual address is 0x608e0000
Registered a device with dynamic Major number of 245
Create a device file for this device with this command:
'mknod /dev/transfpga c 245 0'.
zynq> mknod /dev/transfpga c 245 0
zynq> ./lab2_CORDIC_test
send data to FPGA
trigger FPGA
wait for FPGA
read back the results
mode=0, x=1.190552
mode=0, y=1.137817
mode=0, z=0.003296
mode=1, x=2.328369
mode=1, y=0.052979
mode=1, z=0.762573
zynq>
```

Figure 2. FPGA output for CORDIC polar rotation. Mode 0 represents the cosine/sine mode, while mode 1 represents arctan.

Given that $\tan^{-1}\left(\frac{1.137817}{1.190552}\right) = 0.76275 \approx \frac{\pi}{4}$, it is possible that the test uses $z_0 = \frac{\pi}{4}$ for the cosine/sine mode, though it is unlikely. Another likely reason to the discrepancy for mode 0 may be that unforeseen overflow that was not detected during simulation may have occurred during synthesis and implementation. This may explain our results later for the hyperbolic case, where we delve more into the reasoning.

CORDIC Polar Implementation Summary

Figure 3 shows the utilization, power, and timing summaries for the implementation of the CORDIC design for polar.



Utilization

Hierarchy

Hierarchy
Summary
Slice Logic
Slice LUTs (7%)
LUT as Memory (1%)
LUT as Shift Register
LUT as Logic (6%)
Slice Registers (3%)
Register as Flip Flop (3%)

Name	Slice LUTs (17600)	Block RAM Tile (60)	BUFGCTRL (32)	Slice Registers (35200)	Slice (4400)	LUT as Logic (17600)	LUT as Memory (6000)
top_cordic	1150	16	1	1162	399	1090	60
b0 (blk_mem_gen_0)	8	16	0		7	8	0
c0 (cordic)	605	0	0		166	605	0
mw0 (datatrans_sys_wrapper)	452	0	1		189	392	60

Power

Hierarchical

ilization	Name	Clocks (W)	Signals (W)	Data (W)	Clock Enable (W)	Set/Reset (W)	Logic (W)	BRAM (W)	PS7 (W)	Processor (W)	Memory Interface (W)	I/O Interface (W)	PLLs (W)	AXI (W)
1.473 W (92% of total)	top_cordic													
1.406 W (88% of total)	mw0 (datatrans_sys_wrapper)	0.003	0.002	0.001	<0.001	<0.001	0.001	<0.001	1.4	0.277	0.759	0.015	0.35	0.002
0.06 W (4% of total)	b0 (blk_mem_gen_0)	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	0.06	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
0.006 W (1% of total)	c0 (cordic)	0.002	0.002	0.002	<0.001	<0.001	0.002	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
0.001 W (<1% of total)	Leaf Cells (280)													

Timing

Intra-Clock Paths - clk_fpga_0 - Setup

General Information
Timer Settings
Design Timing Summary
Clock Summary (1)
Check Timing (not available
Intra-Clock Paths
clk_fpga_0
Setup 4.664 ns (10)
Hold 0.130 ns (10)
Pulse Width 4.500 r
Inter-Clock Paths
Other Path Groups

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception	Clock Uncertainty
Path 1	4.664	1	1	68	load_reg/C	c0/z_reg[8]/D	5.189	0.642	4.547	10.0	clk_fpga_0	clk_fpga_0		0.154
Path 2	4.683	1	1	68	load_reg/C	c0/z_reg[4]/D	5.170	0.642	4.528	10.0	clk_fpga_0	clk_fpga_0		0.154
Path 3	4.702	1	1	68	load_reg/C	c0/z_reg[3]/D	5.153	0.642	4.511	10.0	clk_fpga_0	clk_fpga_0		0.154
Path 4	4.729	1	1	123	load_reg_rep__0/C	c0/xs_reg[3][14]/D	5.079	0.642	4.437	10.0	clk_fpga_0	clk_fpga_0		0.154
Path 5	4.743	1	1	123	load_reg_rep__0/C	c0/ys_reg[3][13]/D	5.109	0.672	4.437	10.0	clk_fpga_0	clk_fpga_0		0.154
Path 6	4.757	1	1	68	load_reg/C	c0/z_reg[0]/D	5.094	0.642	4.452	10.0	clk_fpga_0	clk_fpga_0		0.154
Path 7	4.769	1	1	68	load_reg/C	c0/z_reg[1]/D	5.086	0.642	4.444	10.0	clk_fpga_0	clk_fpga_0		0.154
Path 8	4.799	1	1	68	load_reg/C	c0/z_reg[11]/D	5.058	0.642	4.416	10.0	clk_fpga_0	clk_fpga_0		0.154
Path 9	4.905	5	4	40	c0/dk_reg[4]/C	c0/zs_reg[5][9]/D	5.040	2.042	2.998	10.0	clk_fpga_0	clk_fpga_0		0.154
Path 10	4.908	1	1	123	load_reg_rep__0/C	c0/xs_reg[3][0]/D	4.942	0.642	4.300	10.0	clk_fpga_0	clk_fpga_0		0.154

Figure 3. Implementation summary for CORDIC polar design: utilization, power, timing (from top to bottom).

The requirement column in the timing summary details the target clock period, and the slack column details the worst negative slack (WNS). The clock period can be defined as the target clock period minus the WNS. Since all entries in the requirement column are the same, we can define the maximum clock frequency as:

$$f_{max} = \frac{1}{T_{min}} = \frac{1}{(10.0 - \min\{WNS\})\ [ns]} = \frac{1}{(10.0 - 4.664)\ [ns]} = 0.187\ [GHz] = 187\ [MHz]$$

CORDIC Hyperbolic Simulation

Figure 4 shows the results for simulating CORDIC calculations of $\cosh(1)$ and $\sinh(1)$ within the first 50 nanoseconds, followed by confirmation of $\tanh^{-1}\left(\frac{\sinh(1)}{\cosh(1)}\right) = 1$ for the latter 50 nanoseconds. For the cosh/sinh mode, we assume $x_0 = 1$ and $y_0 = 0$, i.e., taking our initial "hyperbolic angle" to be 0 radians; we input $z_0 = 1$ as our target. Inversely for the arctanh mode, we input initial input $z_0 = 0$ radians; assuming the unit hyperbola, we let $x_0 = \cosh(1)$ and $y_0 = \sinh(1)$.

Using 7 iterations starting at index $k = 1$, we see around the 40-ns mark are the computed values for the cosh/sinh mode. Taking $G = 0.82816$, we should see the following results:

$$x = G(x_0 \cosh z_0 - y_0 \sinh z_0) = 0.82816(1 \times \cosh(1) - 0 \times \sinh(1)) = 1.27792$$
$$y = G(x_0 \sinh z_0 + y_0 \cosh z_0) = 0.82816(1 \times \sinh(1) + 0 \times \cosh(1)) = 0.97325$$
$$z = 0$$

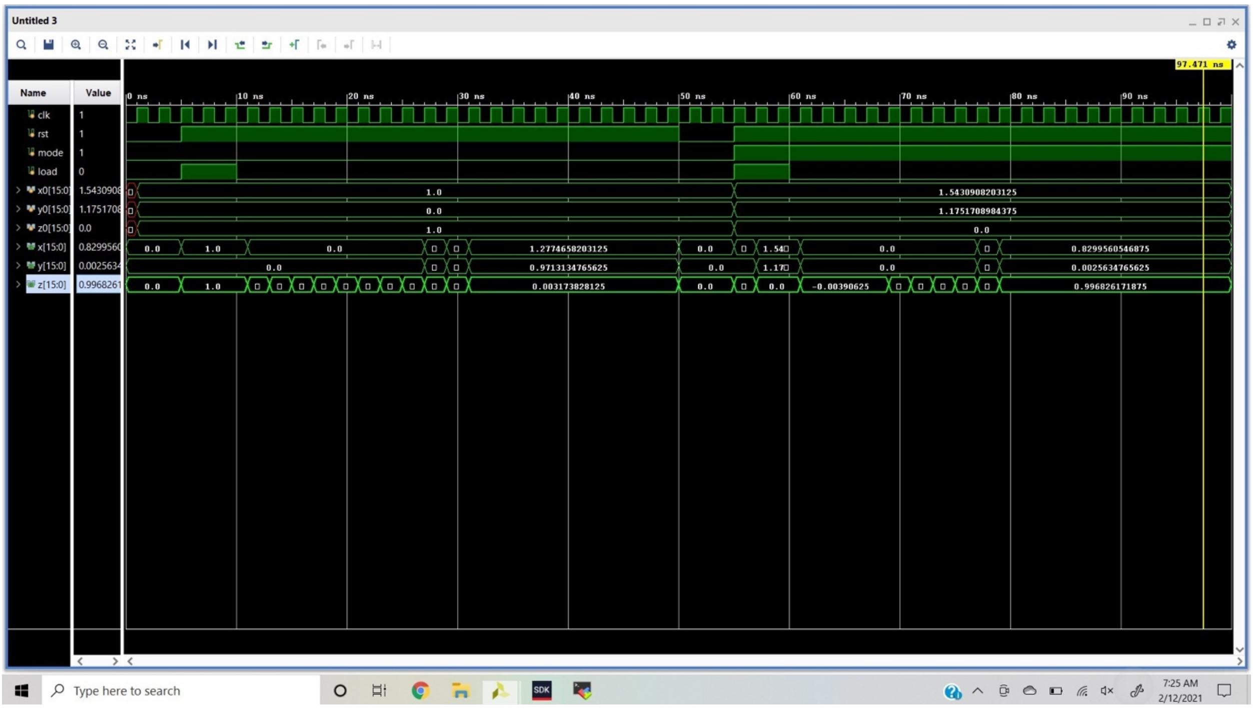


Figure 4. Simulation for CORDIC hyperbolic design. Cosh/sinh mode is simulated within the first half, followed by arctanh mode.

Similarly, for the arctanh mode, we should see the following results around the 80-ns mark:

$$x = G\sqrt{x_0^2 - y_0^2} = 0.82816\sqrt{\cosh^2(1) - \sinh^2(1)} = 0.82816$$
$$y = 0$$
$$z = z_0 + \tanh^{-1}\left(\frac{y_0}{x_0}\right) = 0 + \tanh^{-1}\left(\frac{\sinh(1)}{\cosh(1)}\right) = 1$$

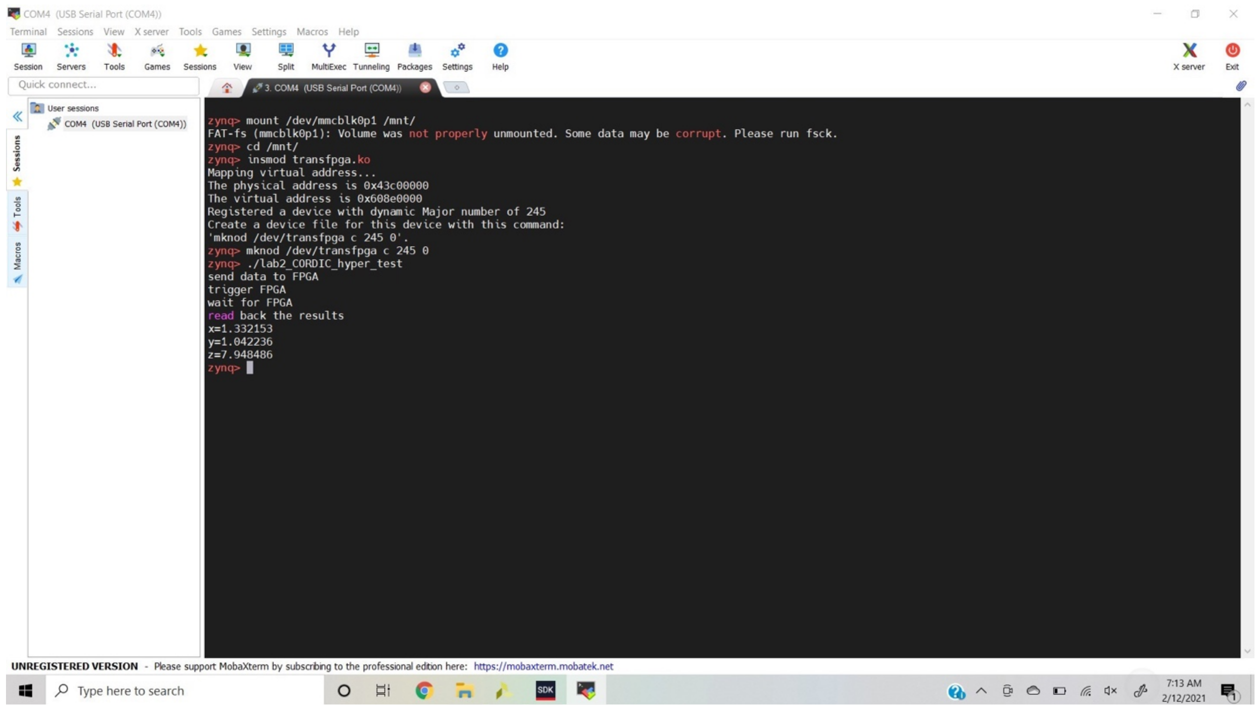
For both modes, we see that the simulated outputs are within 0.05 error from the equations' numerical solutions, verifying that the design should work.

CORDIC Hyperbolic FPGA Outputs

Figure 5 shows the results when implementing the CORDIC hyperbolic design on the FPGA, as seen in the terminal. The values for cosh/sinh (i.e., x and y) are within reasonable range from the simulation. (We can argue that 0.06 or 0.07 error is reasonable.) However, the final z value does not come out to be 0.

Similar to the polar FPGA outputs, it is likely that this is due to overflow not being detected during simulation but carried out during synthesis and implementation. Recall that the numbering format is $Q(3,13)$, which means 3 bits are allocated to the integer part of a floating point. Incrementing $111_2 = 7_{10}$ would result in a neglected carry, thus $000_2 = 0_{10}$ is the next integer. We can make the same inference in the opposite direction; decrementing $000_2 = 0_{10}$ would result in a neglected sign bit that would give $111_2 = 7_{10}$. It is also possible that the test is set such that signed numbers are read as unsigned.

Simply put, because of the closeness of 7.948486 to $8 \ mod \ 8 \equiv 0$, we can say that the terminal output for z is reasonable as well.

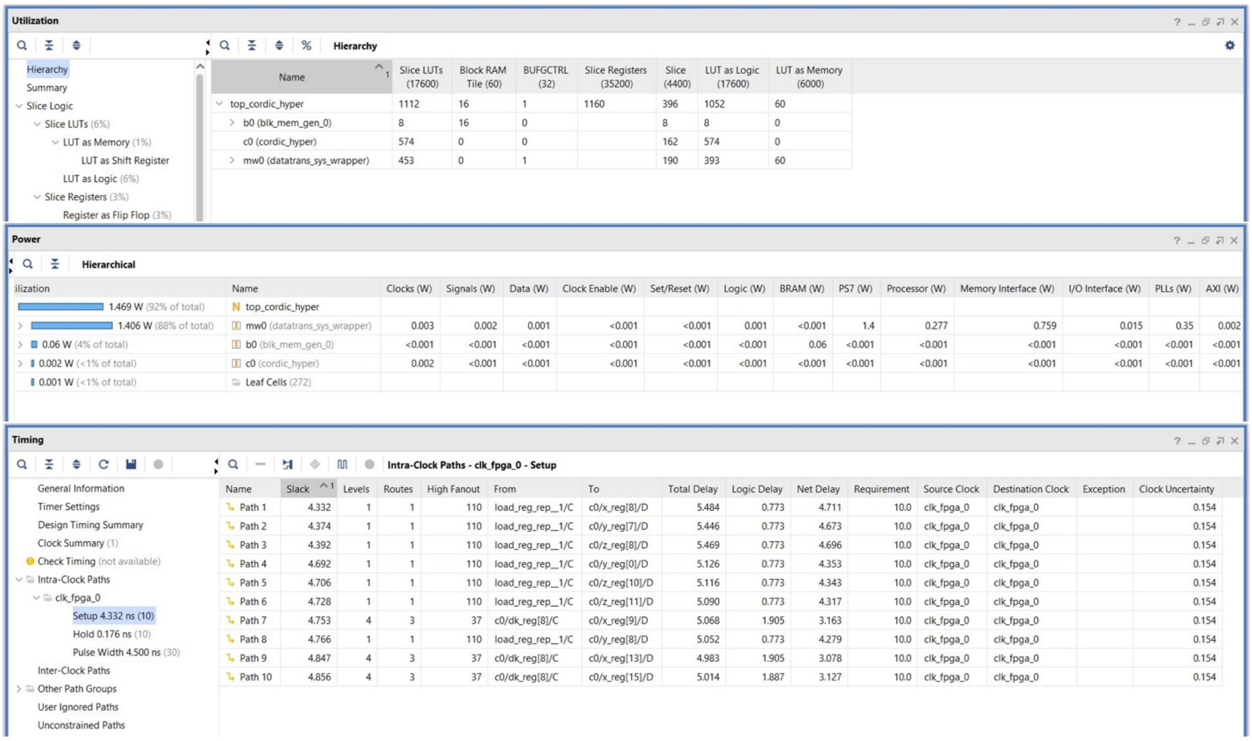

COM4 (USB Serial Port (COM4))
Terminal Sessions View X server Tools Games Settings Macros Help
Session Servers Tools Games Sessions View Split MultiExec Tunneling Packages Settings Help
X server Exit
Quick connect...
3. COM4 (USB Serial Port (COM4))
User sessions
COM4 (USB Serial Port (COM4))
UNREGISTERED VERSION - Please support MobaXterm by subscribing to the professional edition here: https://mobaxterm.mobatek.net
Type here to search
7:13 AM
2/12/2021

```
zynq> mount /dev/mmcblk0p1 /mnt/
FAT-fs (mmcblk0p1): Volume was not properly unmounted. Some data may be corrupt. Please run fsck.
zynq> cd /mnt/
zynq> insmod transfpga.ko
Mapping virtual address...
The physical address is 0x43c00000
The virtual address is 0x608e0000
Registered a device with dynamic Major number of 245
Create a device file for this device with this command:
'mknod /dev/transfpga c 245 0'.
zynq> mknod /dev/transfpga c 245 0
zynq> ./lab2_CORDIC_hyper_test
send data to FPGA
trigger FPGA
wait for FPGA
read back the results
x=1.332153
y=1.042236
z=7.948486
zynq>
```

Figure 5. FPGA output for CORDIC hyperbolic rotation. Only the cosh/sinh mode is tested.

CORDIC Hyperbolic Implementation Summary

Figure 6 shows the utilization, power, and timing summaries for the implementation of the CORDIC design for hyperbolic.



Utilization

Hierarchy
Summary
Slice Logic
Slice LUTs (6%)
LUT as Memory (1%)
LUT as Shift Register
LUT as Logic (6%)
Slice Registers (3%)
Register as Flip Flop (3%)

Hierarchy

Name	Slice LUTs (17600)	Block RAM Tile (60)	BUFGCTRL (32)	Slice Registers (35200)	Slice (4400)	LUT as Logic (17600)	LUT as Memory (6000)
top_cordic_hyper	1112	16	1	1160	396	1052	60
b0 (blk_mem_gen_0)	8	16	0		8	8	0
c0 (cordic_hyper)	574	0	0		162	574	0
mw0 (datatrans_sys_wrapper)	453	0	1		190	393	60

Power

Hierarchical

ilization	Name	Clocks (W)	Signals (W)	Data (W)	Clock Enable (W)	Set/Reset (W)	Logic (W)	BRAM (W)	PS7 (W)	Processor (W)	Memory Interface (W)	I/O Interface (W)	PLLs (W)	AXI (W)
1.469 W (92% of total)	top_cordic_hyper													
1.406 W (88% of total)	mw0 (datatrans_sys_wrapper)	0.003	0.002	0.001	<0.001	<0.001	0.001	<0.001	1.4	0.277	0.759	0.015	0.35	0.002
0.06 W (4% of total)	b0 (blk_mem_gen_0)	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	0.06	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
0.002 W (<1% of total)	c0 (cordic_hyper)	0.002	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
0.001 W (<1% of total)	Leaf Cells (272)													

Timing

General Information
Timer Settings
Design Timing Summary
Clock Summary (1)
Check Timing (not available)
Intra-Clock Paths
clk_fpga_0
Setup 4.332 ns (10)
Hold 0.176 ns (10)
Pulse Width 4.500 ns (30)
Inter-Clock Paths
Other Path Groups
User Ignored Paths
Unconstrained Paths

Intra-Clock Paths - clk_fpga_0 - Setup

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception	Clock Uncertainty
Path 1	4.332	1	1	110	load_reg_rep__1/C	c0/x_reg[8]/D	5.484	0.773	4.711	10.0	clk_fpga_0	clk_fpga_0		0.154
Path 2	4.374	1	1	110	load_reg_rep__1/C	c0/y_reg[7]/D	5.446	0.773	4.673	10.0	clk_fpga_0	clk_fpga_0		0.154
Path 3	4.392	1	1	110	load_reg_rep__1/C	c0/z_reg[8]/D	5.469	0.773	4.696	10.0	clk_fpga_0	clk_fpga_0		0.154
Path 4	4.692	1	1	110	load_reg_rep__1/C	c0/y_reg[0]/D	5.126	0.773	4.353	10.0	clk_fpga_0	clk_fpga_0		0.154
Path 5	4.706	1	1	110	load_reg_rep__1/C	c0/z_reg[10]/D	5.116	0.773	4.343	10.0	clk_fpga_0	clk_fpga_0		0.154
Path 6	4.728	1	1	110	load_reg_rep__1/C	c0/z_reg[11]/D	5.090	0.773	4.317	10.0	clk_fpga_0	clk_fpga_0		0.154
Path 7	4.753	4	3	37	c0/dk_reg[8]/C	c0/x_reg[9]/D	5.068	1.905	3.163	10.0	clk_fpga_0	clk_fpga_0		0.154
Path 8	4.766	1	1	110	load_reg_rep__1/C	c0/y_reg[8]/D	5.052	0.773	4.279	10.0	clk_fpga_0	clk_fpga_0		0.154
Path 9	4.847	4	3	37	c0/dk_reg[8]/C	c0/x_reg[13]/D	4.983	1.905	3.078	10.0	clk_fpga_0	clk_fpga_0		0.154
Path 10	4.856	4	3	37	c0/dk_reg[8]/C	c0/x_reg[15]/D	5.014	1.887	3.127	10.0	clk_fpga_0	clk_fpga_0		0.154

Figure 6. Implementation summary for CORDIC hyperbolic design: utilization, power, timing (from top to bottom).

As with the polar case, we can define the maximum clock frequency as such:

$$f_{max} = \frac{1}{T_{min}} = \frac{1}{(10.0 - \min\{WNS\})\ [ns]} = \frac{1}{(10.0 - 4.332)\ [ns]} = 176\ [MHz]$$

Questions

1. **Please indicate one way to reduce the clock cycles needed for getting the converged results.**

One way to reduce the clock cycles is to implement a pipelined design (which I may have accidentally done for this lab). Instead of having an iterative design where the output registers update itself, the pipelined design would allow parallel computations by designating each iteration to a circuit stage. Seven iterations mean seven cascaded stages; the final stage would feed into the output registers. While there may be some initial latency, pipelining allows for increased throughput.