pp4fpga CORDIC report

電子所碩一 R09943019 李唐

➢ System introduction

CORDIC (Coordinate Rotation Digital Computer) is an efficient method for performing a set of vector rotations on 2D plane by shifting and adding operations instead of complicated multiplication.

$$R_i(\theta) = \begin{bmatrix} \cos\theta_i & -\sin\theta_i \\ \sin\theta_i & \cos\theta_i \end{bmatrix} = \frac{1}{\sqrt{1+\tan^2\theta_i}} \begin{bmatrix} 1 & -\tan\theta_i \\ \tan\theta_i & 1 \end{bmatrix}$$

If we further restrict $\tan\theta_i$ to be a power a 2, the rotation becomes simple shift and add operations.

$$v_i = K_i \begin{bmatrix} 1 & -\sigma_i 2^{-i} \\ \sigma_i 2^{-i} & 1 \end{bmatrix} \begin{bmatrix} x_{i-1} \\ y_{i-1} \end{bmatrix}$$

where $K_i = \frac{1}{\sqrt{1+2^{-2i}}}$ is the scaling factor and σ_i is the rotation direction

i	2^{-i}	Rotating Angle	Scaling Factor	CORDIC Gain
0	1.0	45.000°	1.41421	1.41421
1	0.5	26.565°	1.11803	1.58114
2	0.25	14.036°	1.03078	1.62980
3	0.125	7.125°	1.00778	1.64248
4	0.0625	3.576°	1.00195	1.64569
5	0.03125	1.790°	1.00049	1.64649
6	0.015625	0.895°	1.00012	1.64669

✧ Sine/cosine calculation

Start with a vector on the positive x-axis (initial angle = 0), and perform CORDIC until the angle is approximately at the target angle. We simply read the x and y coordinates to get the values of $\cos\phi$ and $\sin\phi$.

✧ Cartesian to polar conversion

Given Cartesian coordinate (x, y), we rotate the vector to the positive x-axis using CORDIC. The amplitude is the x value of the rotated vector while the angle is the accumulated angle that have rotated.

If y>0 the initial vector is in I or II quadrant, we can first rotate it by -90 degree to put it into I or IV quadrant. If y<0 the initial vector is in III or IV quadrant, we can first rotate it by +90 degree to put it into I or IV quadrant.

✧ Number representation

Arbitrary precision types: ap_int<>, ap_uint<>, ap_fixed<>, ap_ufixed<>

Floating point representation provides large amount of precision but requires significant amount of computations.

➢ Screen dump and Observations

✧ **Solution1: Rotation using multiplier with 32-bit fixed-point representation and 32 iterations**

COS_SIN_TYPE cos_shift = current_cos * sigma * factor;

COS_SIN_TYPE sin_shift = current_sin * sigma * factor;

current_cos = current_cos - sin_shift;

current_sin = current_sin + cos_shift;

In kernel code, each iteration calculates the multiplication between previous cosine and sine value with a power of 2. Then the current cosine and sine values are calculated by addition and subtraction.

✧ Synthesis report

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	8.502 ns	1.25 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
97	97	0.970 us	0.970 us	97	97	none

Detail

Instance

Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- Loop 1	96	96	3	-	-	32	no

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	8	0	512	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	1	-	0	0	-
Multiplexer	-	-	-	72	-
Register	-	-	339	-	-
Total	1	8	339	584	0
Available	280	220	106400	53200	0
Utilization (%)	~0	3	~0	1	0

Interface

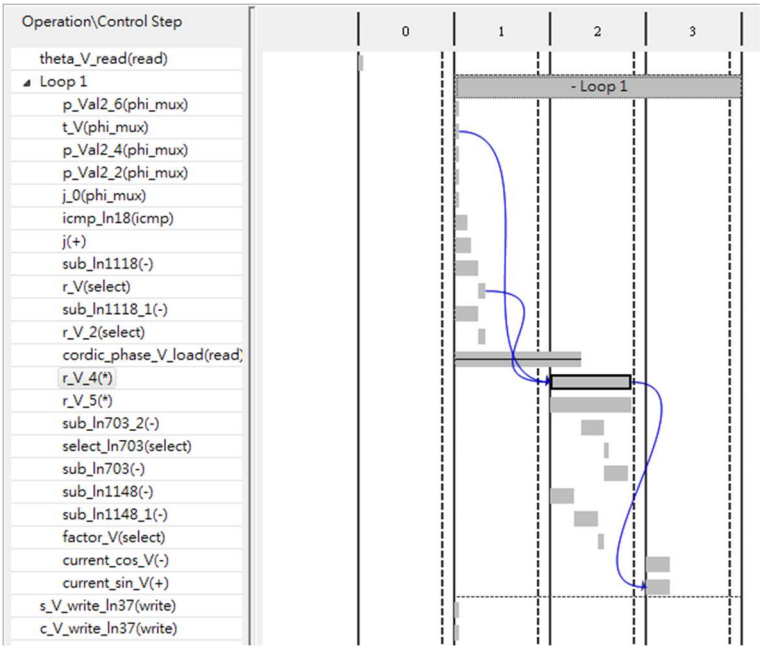
Summary

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_clk	in	1	ap_ctrl_hs	cordic	return value
ap_rst	in	1	ap_ctrl_hs	cordic	return value
ap_start	in	1	ap_ctrl_hs	cordic	return value
ap_done	out	1	ap_ctrl_hs	cordic	return value
ap_idle	out	1	ap_ctrl_hs	cordic	return value
ap_ready	out	1	ap_ctrl_hs	cordic	return value
theta_V	in	32	ap_none	theta_V	scalar
s_V	out	32	ap_vld	s_V	pointer
s_V_ap_vld	out	1	ap_vld	s_V	pointer
c_V	out	32	ap_vld	c_V	pointer
c_V_ap_vld	out	1	ap_vld	c_V	pointer

The latency in solution 1 requires 97 cycles in total with 3 cycles in each iteration. The utilization includes 8 DSP for two 32-bit multiplications.

✧ Analysis perspective

	Pipelined	Latency	Iteration Latency	Initiation Interval	Trip count
◢ cordic	-	97	-	98	-
Loop 1	no	96	3	-	32



The multiplication operation requires a whole cycle to complete, followed by an adder and a subtractor.

✧ In co-simulation, the average error in output sine and cosine values are both 0.0160(%).

✧ **Solution2: CORDIC with 32-bit fixed-point representation and 32 iterations**

```
COS_SIN_TYPE cos_shift = current_cos >> j;
COS_SIN_TYPE sin_shift = current_sin >> j;
```

In kernel code, the multiplication with a power of 2 can be replaced by simple right shifting operation, which would greatly reduce the hardware

resource.

✧ Synthesis report

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	7.670 ns	1.25 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
65	65	0.650 us	0.650 us	65	65	none

Detail

Instance

Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- Loop 1	64	64	2	-	-	32	no

Utilization Estimates

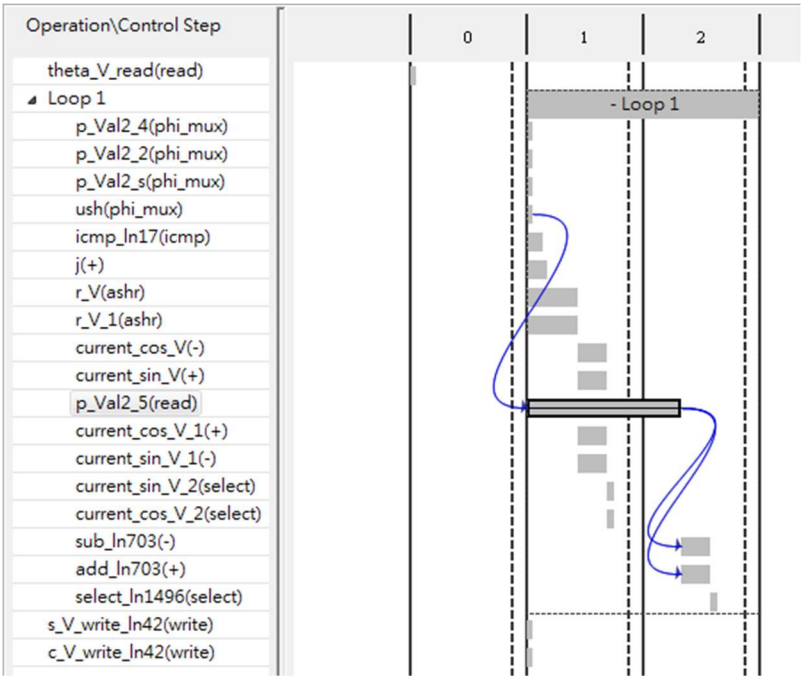
Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	558	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	1	-	0	0	-
Multiplexer	-	-	-	57	-
Register	-	-	176	-	-
Total	1	0	176	615	0
Available	280	220	106400	53200	0
Utilization (%)	~0	0	~0	1	0

The latency in solution 2 drops from 97 cycles to 65 cycles, which is due to the 1-cycle reduction in each iteration. The utilization is lowered by zero requirement in DSP resource and some reduction in FF resource, but the LUT resource is a bit higher due to shifting operation.

✧ Analysis perspective

	Pipelined	Latency	Iteration Latency	Initiation Interval	Trip count
◢ ● cordic	-	65	-	66	-
● Loop 1	no	64	2	-	32



The shifting operation can be executed without an extra cycle, so the cycle in each iteration is reduced from 3 cycles to 2 cycles.

- **Solution3: pipeline directive to the iteration loop**
- Synthesis report

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	7.670 ns	1.25 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
34	34	0.340 us	0.340 us	34	34	none

Detail

Instance

Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- ITERATION	32	32	2	1	1	32	yes

Utilization Estimates

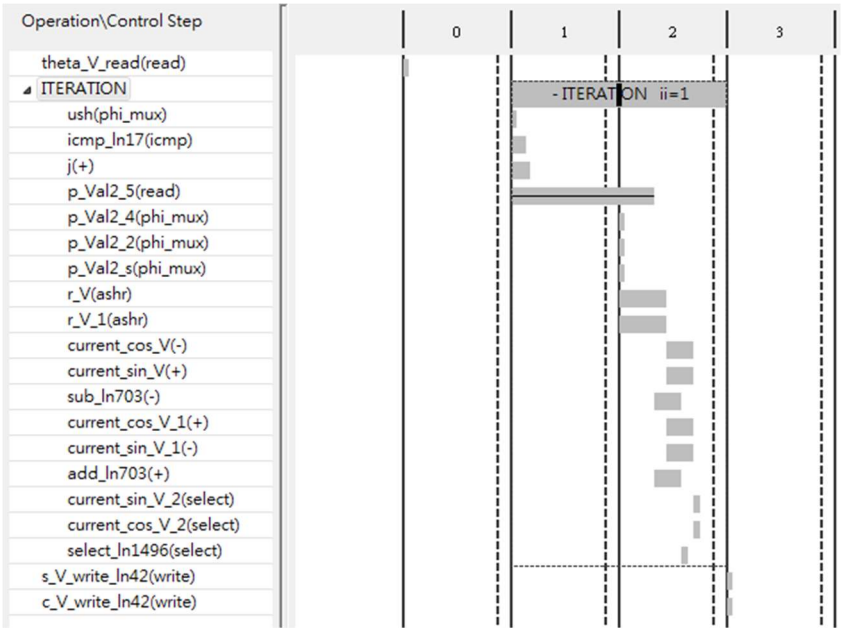
Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	562	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	1	-	0	0	-
Multiplexer	-	-	-	81	-
Register	-	-	114	-	-
Total	1	0	114	643	0
Available	280	220	106400	53200	0
Utilization (%)	~0	0	~0	1	0

The latency in solution 3 is further reduced to 34 cycles by applying pipelining with II = 1. We can also find that the FF resource is a bit reduced compared with solution 2. If we compare the register utilization and the scheduler timeline of solution 2 and 3, we will find that the current_cos and current_sin values are calculated in each cycle due to pipeline without having to be stored in FF to wait for the next cycle.

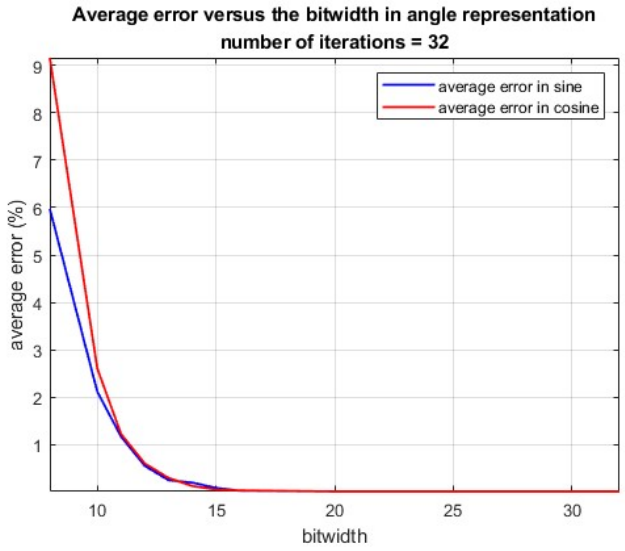
✧ Analysis perspective

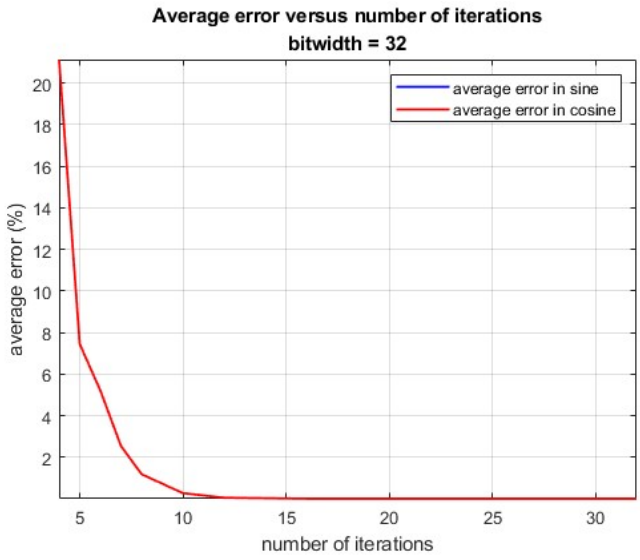
	Pipelined	Latency	Iteration Latency	Initiation Interval	Trip count
◢ cordic	-	34	-	35	-
ITERATION	yes	32	2	1	32



✧ **Solution4: 22-bit fixed-point representation and 22 iterations**

We try different bit-width representation for the angles and different number of iterations to observe the average error in output sine and cosine values.





From the above result, if we adopt 22-bit representation and 22 iterations, then the accuracy of the output would be acceptable. Also, it's interesting that the cosine values suffer more than the sine values when using low bit-width representation in range 0~90 degree.

✧ Synthesis report

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	7.106 ns	1.25 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
24	24	0.240 us	0.240 us	24	24	none

Detail

Instance

Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- ITERATION	22	22	2	1	1	22	yes

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	392	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	1	-	0	0	-
Multiplexer	-	-	-	81	-
Register	-	-	82	-	-
Total	1	0	82	473	0
Available	280	220	106400	53200	0
Utilization (%)	~0	0	~0	~0	0

Interface

Summary

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_clk	in	1	ap_ctrl_hs	cordic	return value
ap_rst	in	1	ap_ctrl_hs	cordic	return value
ap_start	in	1	ap_ctrl_hs	cordic	return value
ap_done	out	1	ap_ctrl_hs	cordic	return value
ap_idle	out	1	ap_ctrl_hs	cordic	return value
ap_ready	out	1	ap_ctrl_hs	cordic	return value
theta_V	in	22	ap_none	theta_V	scalar
s_V	out	22	ap_vld	s_V	pointer
s_V_ap_vld	out	1	ap_vld	s_V	pointer
c_V	out	22	ap_vld	c_V	pointer
c_V_ap_vld	out	1	ap_vld	c_V	pointer

The latency in solution 4 is further reduced to 24 cycles due to less iterations. Moreover, the utilization and timing are greatly improved by lower number of bits in representation.

✧ Analysis perspective

	Pipelined	Latency	Iteration Latency	Initiation Interval	Trip count
◢ ● cordic	-	24	-	25	-
● ITERATION	yes	22	2	1	22

✧ **Solution5: 32-bit floating point representation**

We replace the data types with float to make a comparison with fixed point representation. Because the shifting operations are not supported for float, we replace them with multiplication operation.

✧ Synthesis report

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	7.954 ns	1.25 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
290	290	2.900 us	2.900 us	290	290	none

Detail

Instance

Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- ITERATION	288	288	9	9	1	32	yes

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	155	-
FIFO	-	-	-	-	-
Instance	-	10	762	1661	-
Memory	1	-	0	0	-
Multiplexer	-	-	-	188	-
Register	-	-	312	-	-
Total	1	10	1074	2004	0
Available	280	220	106400	53200	0
Utilization (%)	~0	4	1	3	0

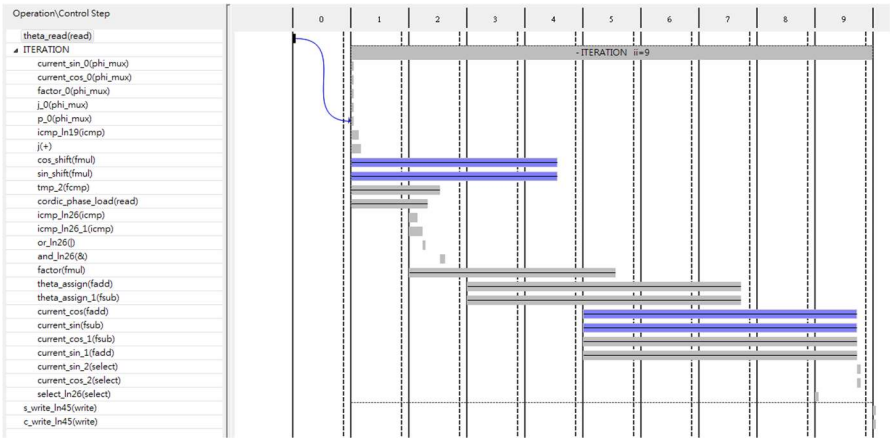
Instance

Instance	Module	BRAM_18K	DSP48E	FF	LUT	URAM
cordic_faddfsub_3bkb_U1	cordic_faddfsub_3bkb	0	2	205	390	0
cordic_faddfsub_3bkb_U2	cordic_faddfsub_3bkb	0	2	205	390	0
cordic_fcmp_32ns_dEe_U5	cordic_fcmp_32ns_dEe	0	0	66	239	0
cordic_fmul_32ns_cud_U3	cordic_fmul_32ns_cud	0	3	143	321	0
cordic_fmul_32ns_cud_U4	cordic_fmul_32ns_cud	0	3	143	321	0
Total	5	0	10	762	1661	0

It can be observed that floating point operations require significant amount of resource. The latency is also much longer than solution 1.

- ✧ Analysis perspective

	Pipelined	Latency	Iteration Latency	Initiation Interval	Trip count
cordic	-	290	-	291	-
ITERATION	yes	288	9	9	32



- ✧ In co-simulation, the average error in output sine and cosine values are both 0.0160(%).

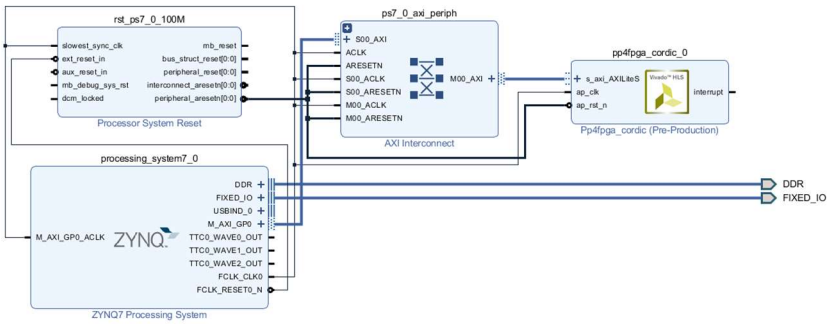
- ➢ ZYNQ implementation
  - ✧ We choose solution 4 for further implementation on ZYNQ. In addition, the port interface is modified to AXI-Lite protocol.

Interface

Summary

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
s_axi_AXILiteS_AWVALID	in	1	s_axi	AXILiteS	pointer
s_axi_AXILiteS_AWREADY	out	1	s_axi	AXILiteS	pointer
s_axi_AXILiteS_AWADDR	in	6	s_axi	AXILiteS	pointer
s_axi_AXILiteS_WVALID	in	1	s_axi	AXILiteS	pointer
s_axi_AXILiteS_WREADY	out	1	s_axi	AXILiteS	pointer
s_axi_AXILiteS_WDATA	in	32	s_axi	AXILiteS	pointer
s_axi_AXILiteS_WSTRB	in	4	s_axi	AXILiteS	pointer
s_axi_AXILiteS_ARVALID	in	1	s_axi	AXILiteS	pointer
s_axi_AXILiteS_ARREADY	out	1	s_axi	AXILiteS	pointer
s_axi_AXILiteS_ARADDR	in	6	s_axi	AXILiteS	pointer
s_axi_AXILiteS_RVALID	out	1	s_axi	AXILiteS	pointer
s_axi_AXILiteS_RREADY	in	1	s_axi	AXILiteS	pointer
s_axi_AXILiteS_RDATA	out	32	s_axi	AXILiteS	pointer
s_axi_AXILiteS_RRESP	out	2	s_axi	AXILiteS	pointer
s_axi_AXILiteS_BVALID	out	1	s_axi	AXILiteS	pointer
s_axi_AXILiteS_BREADY	in	1	s_axi	AXILiteS	pointer
s_axi_AXILiteS_BRESP	out	2	s_axi	AXILiteS	pointer
ap_clk	in	1	ap_ctrl_hs	pp4fpga_cordic	return value
ap_rst_n	in	1	ap_ctrl_hs	pp4fpga_cordic	return value
interrupt	out	1	ap_ctrl_hs	pp4fpga_cordic	return value

✧ The block diagram is shown as follows. The clock frequency in the design is 100 (MHz).

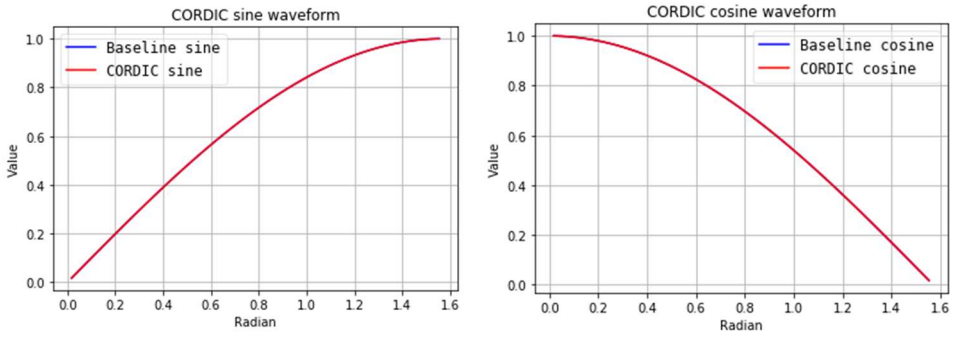


✧ The python simulation result shows that the kernel execution time over the 89 testing angles in radian is about 0.0167(s). Also, the baseline sine and cosine values calculated by python library are almost the same as those by kernel function for each radian. Finally, the average error in output sine and cosine values are 0.0158(%) and 0.0156(%) respectively.

```
Kernel execution time: 0.016658306121826172 s
Radian          Baseline_sin    Baseline_cos    CORDIC_sin      CORDIC_cos      error_sin(%)    error_cos(%)
0.017453        0.017452        0.999848        0.017456        1.000005        0.020904        0.015710
0.034907        0.034899        0.999391        0.034904        0.999550        0.014279        0.015914
0.052360        0.052336        0.998630        0.052340        0.998790        0.006874        0.016047
0.069813        0.069756        0.997564        0.069768        0.997724        0.016455        0.015992
0.087266        0.087156        0.996195        0.087167        0.996353        0.012671        0.015906
0.104720        0.104528        0.994522        0.104544        0.994681        0.014563        0.016034
0.122173        0.121869        0.992546        0.121887        0.992704        0.014658        0.015943
0.139626        0.139173        0.990268        0.139192        0.990427        0.013312        0.016051
0.157080        0.156434        0.987688        0.156458        0.987845        0.014981        0.015904
0.174533        0.173648        0.984808        0.173674        0.984963        0.014657        0.015807
0.191986        0.190809        0.981627        0.190840        0.981784        0.016127        0.015962
0.209440        0.207912        0.978148        0.207945        0.978301        0.015958        0.015688
0.226893        0.224951        0.974370        0.224989        0.974525        0.016841        0.015947
Total_Error_Sin=1.402338, Total_error_Cos=1.384689
```

✧ From the sine and cosine waveform, we can see that the baseline curves are almost the same as those generated from CORDIC, verifying the

effectiveness of the whole design.



- GitHub submission
  https://github.com/nodetibylno/MSoC_self_paced_learning
- Reference

1. R. Kastner, J. Matai, and S. Neuendorffer , Parallel Programming for FPGAs, arXiv , 2018.