LSI Design Contest 2015 - Level 2

Low-Resource Low-Latency Hybrid Adaptive CORDIC With Floating-Point Precision

---

Team Name: THUx3

- **Hong-Thu Nguyen**

  Research Student
  The University of Eletro-Communications, Tokyo, Japan
  Address: Room 105, Fujimi House 2, Fujimi-cho 2-10-67
  Chofu city, Tokyo, Japan, 182-0033.
  Tel: +81-70-2611-6852
  E-mail: `hongthu@vlsilab.ee.uec.jp`
  T-shirt size: S (Small)

- **Trong-Thuc Hoang**

  First Year Master Student
  The University of Science, Ho Chi Minh City, Vietnam
  Address: 255 Huynh Van Banh St., Ward 12, Phu Nhuan Dist.,
  Ho Chi Minh, Vietnam.
  Tel: +84 16-6866-6951
  E-mail: `htthuc@fetel.hcmus.edu.vn`
  T-shirt size: XXL (Extreme Large)

- **Xuan-Thuan Nguyen**

  First Year Doctor Student
  The University of Eletro-Communications, Tokyo, Japan
  Address: Room B517, International Village 1-29-1 Gakuen-nishi-machi
  Kodaira-shi, Tokyo, Japan, 187-0045.
  Tel: +81-80-2255-3829
  E-mail: `xuanthuan@vlsilab.ee.uec.jp`
  T-shirt size: M (Medium)

Table of Contents

1 General Block Diagram of Proposed Hybrid Adaptive CORDIC (HA-CORDIC)

The proposed design is composed of four main modules namely ANGLE_SELECTION (ASEL), FIFO, PRE_CALCULATION (PREC), and POST_CALCULATION (POSC), which is illustrated in Fig. 1a. The input is a 24-bit fixed-point (FIX) angle and the output are two 32-bit floating-point (FLP) trigonometric results. The FIX format is 1.8.15, i.e. 1-bit sign, 8-bit magnitude, and 15-LSBs consecutively fractional, which can literally support all the angles within $[-180^o, 180^o]$. The input range and the output precision, in our point of view, are suitable for most of advanced applications.

To begin with, HA-CORDIC receives an input angle at the assertion of *iData_valid*. The output of ASEL such as angle recovery information and selected rotation angles is subsequently sent to FIFO. Module PREC consecutively calculates each coordinate X/Y and factor K by reading data from FIFO. Following this, the product of factor and latest coordinates are corrected by the angle recovery information in POSC. Signal *oData_valid*, eventually, confirm that the current sine/cosine result is valid. With the aid of two acknowledge signals, HA-CORDIC is more likely to be integrated into other systems easily.

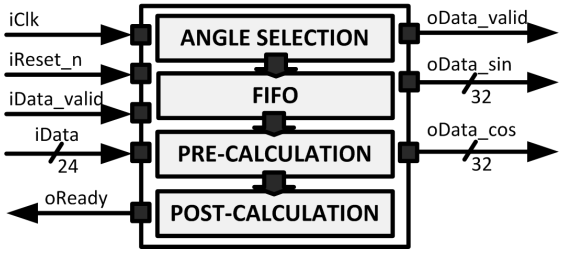
The processing of HA-CORDIC is separated into two parallel threads, which can be seen from Fig. 1b. Because of the difference in operating cycles a FIFO is inserted between ASEL and PREC to ease the latency. In fact, ASEL and PREC/POSC cost one and two clock cycles for FIX and FLP operation, respectively. Besides, pipeline processing is applied in all modules to increase the throughput. Depending on the number of iterations determined in ASEL, the execution latency of each angle is markedly different. Each module is described in more detail below.

2 Explanation of Proposed HA-CORDIC Algorithm and Hardware

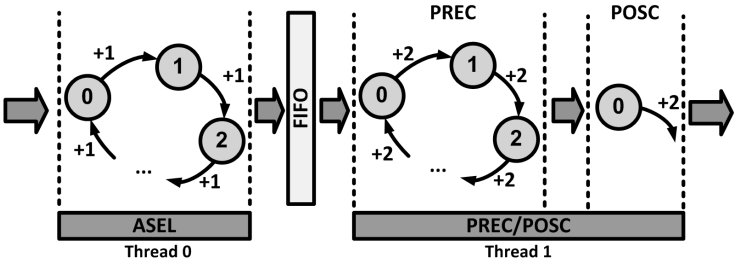
2.1 Explanation Of Proposed Algorithm

Overview - The HA-CORDIC algorithm is proposed to reduce the number of iterations and thereby reduces the calculation latency. In other words, only several angles in the set of N predetermined angle constant are utilized to form the closest result. Our optimized 4-step algorithm developed for sine/cosine calculation is shown in Fig. 1. To begin with, input angle is converted into predefined range. Secondly, an optimum set of micro-rotations whose sum approximates the input angle are selected. The coordination x, y, and K factor, then, are correspondingly updated. Finally, sine/cosine is achieved by the product of latest x_t/y_t, and K, together with simple adjustments.

Angle Normalization - The CORDIC algorithm will only converge across a limited range of input values. In rotation mode, convergence is guaranteed



(a) The general block diagram of proposed CORDIC.



(b) The mechanism of proposed CORDIC.

for the angles below the sum of entire N, i.e. between -99.88^o and 99.88^o. A trigonometric identity is used to translate any outside angle into that within this range. In addition, by employing several simple adjustments, we can shorten the range to $[0^o, 45^o]$, according to (1). Value Φ and Φ_t are initial and normalized angle, q_b, q_c, and q_t (2) are normalized parameters used to correct the final sine/cosine result.

$$\Phi_a = \begin{cases} \Phi + 360^o & \text{repeat until } \Phi > -180^o \\ \Phi - 360^o & \text{repeat until } \Phi < 180^o \end{cases} \tag{1a}$$

$$\Phi_b = \begin{cases} \Phi_a + 180^o & \text{if } \Phi_a < -90^o \\ \Phi_a - 180^o & \text{if } \Phi_a > 90^o \end{cases} \tag{1b}$$

$$\Phi_c = \begin{cases} \Phi_b + 90^o & \text{if } \Phi_b < -45^o \\ 90^o - \Phi_b & \text{if } \Phi_b > 45^o \end{cases} \tag{1c}$$

$$\Phi_t = \begin{cases} \Phi_c & \text{if } \Phi_c \in [0^o, 45^o] \\ |\Phi_c| & \text{if } \Phi_c \in [-45^o, 0^o] \end{cases} \tag{1d}$$

$$q_b = \begin{cases} 1 & \text{if } \Phi_a \in [-90^o, 90^o] \\ -1 & \text{if } \Phi_a \notin [-90^o, 90^o] \end{cases} \tag{2a}$$

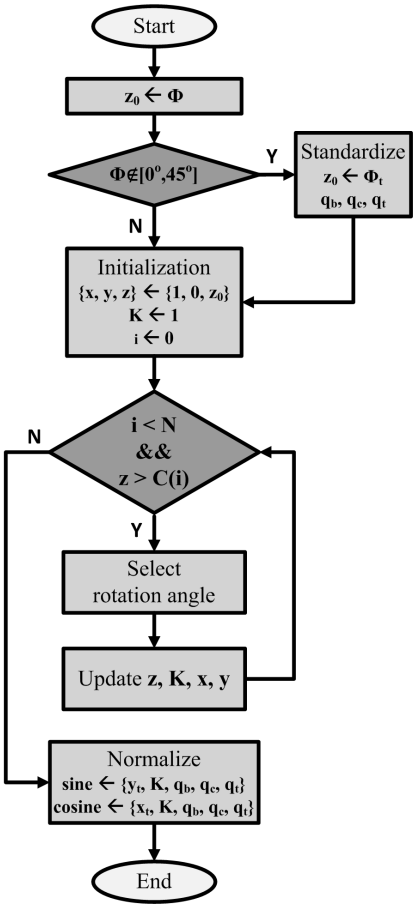


Fig. 1: The flow chart of HA-CORDIC algorithm.

$$q_c = \begin{cases} -1 & \text{if } \Phi_b < -45^o \\ 1 & \text{if } \Phi_b > 45^o \\ 0 & \text{Otherwise} \end{cases} \tag{2b}$$

$$q_t = \begin{cases} 1 & \text{if } \Phi_c \in [0^o, 45^o] \\ -1 & \text{if } \Phi_c \in [-45^o, 0^o] \end{cases} \tag{2c}$$

Angle Selection - Assume that 16 predetermined angles, θ, is employed in our CORDIC - Table 1. Conventional CORDIC exploits all θ_i angles and their direction d_i for trigonometric computations. For example, an 30^o input angle

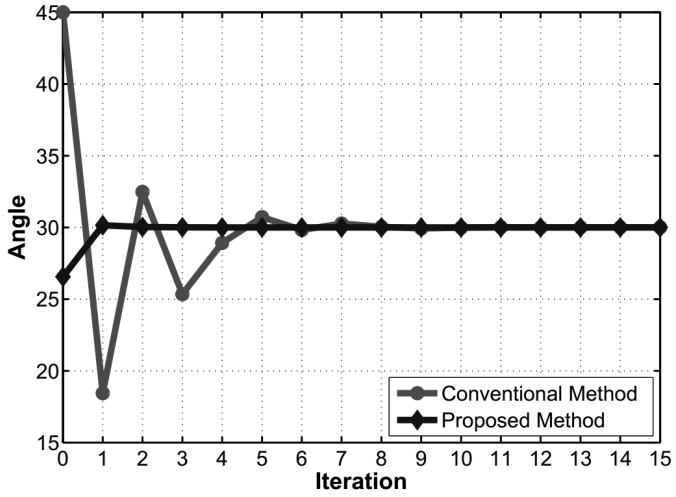


Fig. 2: An example of iterations between conventional and proposed CORDIC, assume that input angle is 30^o.

costs 16 iterations.

$$(\theta_0 - \theta_1 + \theta_2 - \theta_3 + \theta_4 + \theta_5 - \theta_6 + \theta_7 - \theta_8 \\ - \theta_9 + \theta_{10} + \theta_{11} - \theta_{12} + \theta_{13} - \theta_{14} - \theta_{15}) = 30.000834^o$$

The residual angle z of conventional CORDIC in a case of input angle 30^o is 0.000834. Our angle selection method, however, only requires five iterations to achieve smaller residual angle, 0.000245. The comparison in iterations between two methods is exemplified in Fig. 2.

$$(\theta_1 + \theta_4 - \theta_9 - \theta_{11} - \theta_{15}) = 29.999755^o$$

The key point of this method is that in each iteration i, rotation angle θ_i is chosen so that residual angle z_i closes to zero. The iteration stops upon z_i is smaller than a predefined *threshold*.

$$\begin{aligned} || z_i | - \theta_i | &= \min_{i \le j \le (N-1)} || z_i | - \theta_j | \\ z_{i+1} &= z_i - d_i \theta_i \\ d_i &= sign(z_i) \end{aligned} \tag{3}$$

In order to reduce the comparators in (3), we proposed a set of parameters, C, which expresses the range of residual angles around one angle constant, as defined in (4). The details of C is described in Table 1.

$$c_i = \begin{cases} \frac{\theta_i + \theta_{i+1}}{2} & \text{if } 0 \le i \le (N-2) \\ \frac{\theta_i}{2} & \text{otherwise} \end{cases} \tag{4}$$

On this basis, the angle recording pseudocode is summarized in Fig. 3.

Table 1: The values of θ, C, and K.

i	θ	C	K
0	45.000000000	35.782525588	0.707106781
1	26.565051177	20.300647322	0.894427191
2	14.036243468	10.580629908	0.970142500
3	7.125016349	5.350675362	0.992277877
4	3.576334375	2.683122491	0.998052578
5	1.789910608	1.342542159	0.999512076
6	0.895173710	0.671393940	0.999877952
7	0.447614171	0.335712335	0.999969484
8	0.223810500	0.167858088	0.999992371
9	0.111905677	0.083929284	0.999998093
10	0.055952892	0.041964672	0.999999523
11	0.027976453	0.020982340	0.999999881
12	0.013988227	0.010491170	0.999999970
13	0.006994114	0.005245585	0.999999993
14	0.003497057	0.002622792	0.999999998
15	0.001748528	0.000874264	0.999999999

1: $i = j = 0$
2: $threshold = c(15)$
3: **while** $z(j) > threshold \quad and \quad j < (N-1)$ **do**
4: **if** $z(i) > c(j)$ **then**
5: *select rotation angle* $\theta(j)$
6: *update residual* $z(i)$
7: $i = i + 1$;
8: **end if**
9: $j = j + 1$;
10: **end while**

Fig. 3: The pseudocode of angle selection function.

Pre-calculation: Coordinate, Residual, And Factor Adaption - The adaption of coordinate x and y, residual z, and factor K are described in (5), where k_j is listed in Table 1. Because only several angles are selected, factor 2^{-i} and θ_i are replaced by 2^{-j} and θ_j, respectively. Both j and θ_j are obtained in Fig. 3.

$$\begin{aligned} x_{i+1} &= x_i - d_i y_i 2^{-j} \\ y_{i+1} &= y_i + d_i x_i 2^{-j} \\ z_{i+1} &= z_i - d_i \theta_j \\ K &= K * k_j \end{aligned} \tag{5}$$

If the residual is smaller than the defined threshold, we achieve latest x_t, y_t, and K.

$$\begin{aligned} X_t &= x_t * K \\ Y_t &= y_t * K \end{aligned} \tag{6}$$

Post-calculation: Sine/cosine Recovery - Sine and cosine of input angle, then, are calculated according to Equation (7).

$$X_1 = X_t$$

$$Y_1 = \begin{cases} Y_t & \text{if } q_t = 1 \\ -Y_t & \text{if } q_t = -1 \end{cases} \tag{7a}$$

$$X_2 = \begin{cases} Y_1 & \text{if } q_c = 1 \\ X_1 & \text{if } q_c = 0 \\ Y_1 & \text{if } q_c = -1 \end{cases}$$

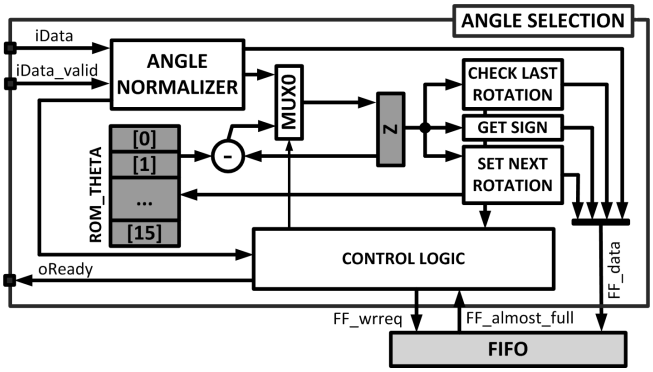
$$Y_2 = \begin{cases} X_1 & \text{if } q_c = 1 \\ Y_1 & \text{if } q_c = 0 \\ -X_1 & \text{if } q_c = -1 \end{cases} \tag{7b}$$

$$\begin{aligned} sin(\Phi) &= q_b * X_2 \\ cos(\Phi) &= q_b * Y_2 \end{aligned} \tag{7c}$$

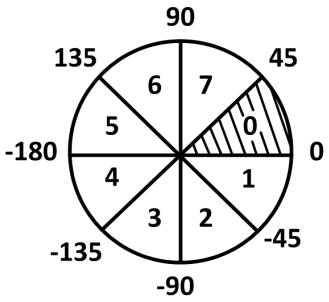
2.2 Explanation Of Proposed Hardware

Angle Selection (ASEL) - Module ASEL strives to obtain the precise result with least number of iterations by reducing divergent pseudo-rotations. This module is composed of three main componenents, ANGLE_NORMALIZER (ANOR), SET_NEXT_ROTATION (SNR), and CHECK_LAST_ROTATION (CLR), which is depicted in Fig. 4a.

Beforehand, ANOR converts input angle *iData* into normalization range of $[0^o,45^o]$. It can be seen in Fig. 4b, if we split a circle into eight pieces, any angle from first to seventh piece can be transformed into equivalent angle in zeroth piece according to (1). For example, an angle -120^o in third piece can be normalized to 30^o in zeroth piece with the transformation sequence of $\{-120^o, +60^o, +30^o\}$. In fact, $sin(-120^o) = -0.866$ and $cos(-120^o) = -0.5$ are correspondingly equal to $-cos(30^o)$ and $-sin(30^o)$. The 4-bit angle recovery information in this case is 1111. Figure 5 shows the architecture of ANOR. ROM_RANGE contains the predefined ranges such as $[-45^o,-90^o]$, $[-90^o,-135^o]$ etc. and ROM_RECOVERY stores correction information. Recovery information and normalized angle are sent out by *rec_info* and *norm_angle*, respectively.



(a) The hardware architecture of ASEL.



	Range	Rec. Info	Correction
0	[0, 45]	00 00	$\begin{cases}\sin\Phi = \sin\Phi_t \\ \cos\Phi = \cos\Phi_t\end{cases}$
1	[-45, 0]	01 00	$\begin{cases}\sin\Phi = -\sin\Phi_t \\ \cos\Phi = \cos\Phi_t\end{cases}$
2	[-90, -45]	11 10	$\begin{cases}\sin\Phi = -\cos\Phi_t \\ \cos\Phi = -\sin\Phi_t\end{cases}$
3	[-135, -90]	11 11	$\begin{cases}\sin\Phi = -\cos\Phi_t \\ \cos\Phi = -\sin\Phi_t\end{cases}$
4	[-180, -135]	01 01	$\begin{cases}\sin\Phi = -\sin\Phi_t \\ \cos\Phi = -\cos\Phi_t\end{cases}$
5	[135, 180]	00 01	$\begin{cases}\sin\Phi = \sin\Phi_t \\ \cos\Phi = -\cos\Phi_t\end{cases}$
6	[90, 135]	10 11	$\begin{cases}\sin\Phi = \cos\Phi_t \\ \cos\Phi = -\sin\Phi_t\end{cases}$
7	[45, 90]	10 10	$\begin{cases}\sin\Phi = \cos\Phi_t \\ \cos\Phi = \sin\Phi_t\end{cases}$

(b) A description of normalization technique.

Fig. 4: The hardware architecture and normalization technique of ASEL.

After receiving the normalized angle, SNR determines the next angle in ROM_θ by utilizing a pair of ROM_C and priority encoder, as illustrated in Fig. 6. Module ROM_θ includes 16 predetermined rotation angles θ while ROM_C stores the range of residual angles C around one angle constant, which is defined in (4). Both LUTs are graphically illustrated in Table 2. In order to eliminate the *while* loop in pseudocode from Fig. 3, a set of comparators are deployed in parallel together with a priority encoder to search for next suitable angle θ at the speed of one cycle. The iteration completes as soon as residual angle register Z is smaller than threshold ROM_C[15].

In order to reduce the last wasted cycle in each loop, i.e. the cycle in which Z is smaller than threshold and thereby no θ is selected, CLR circuit is deployed in Fig. 6. At each iteration i, CLR checks whether next process $(i+1)$ is the last or not. If the current process is final iteration, CLR signals ASEL to stop calculating

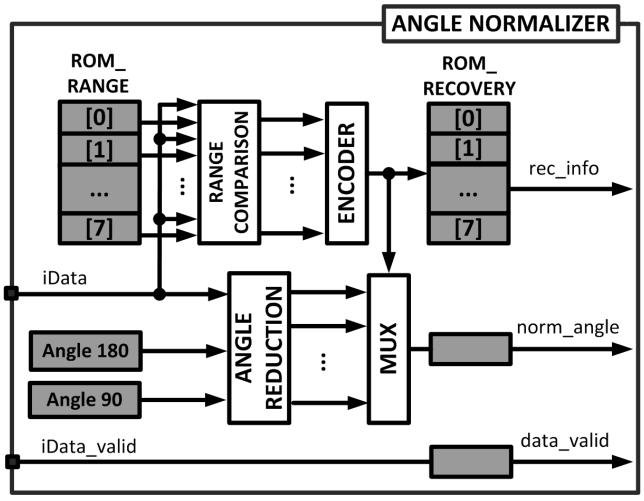


Fig. 5: The hardware architecture of ANOR.

and start the new input angle rotation in following cycle. The pseudocode of this circuit is described in Fig. 7. If input angle Φ is approximately zero, no rotation is executed (line 5 and 6). If z is within the determined range, updated z will become smaller than threshold in next rotation (line 7 to 10). Because the addition and subtraction affects the circuit frequency, two LUTs ROM_s ($\theta(i) - threshold$) and ROM_a ($\theta(i) + threshold$) are implemented instead. It can be seen that *last_rotation* and *phase_addr* are implemented in parallel.

A collection of 4-bit *norm_info*, 2-bit *last_rotation*, 1-bit sign, and 4-bit *phase_addr*, lastly, is brought together and put into FIFO. At the same time, CONTROL_LOGIC gets FIFO to accept data by asserting *FF_wrreq*. If FIFO is not available or current angle is still in progress, *oReady* will go to low level and thereby CORDIC cannot accept new input angle.

Pre-calculation (PREC) - PREC performs the FLP arithmetic of X and Y coordinate, and K factor in parallel, as mentioned in (5). This module contains four main components, FADD_SUB, FMUL_ki, FMUL_XYK, and CONTROL_LOGIC, as shown in Fig. 8. Beforehand, *signZ* and i which store *phase_sign* and *phase_addr* are get from FIFO by CONTROL_LOGIC. FADD_SUB and FMUL_ki, then, consecutively update X, Y, and K by asserting *start* signal. The format of X and Y include 1-bit sign and 24-bit fraction. Both X and Y are smaller than two, thereby their fractions have one MSB before dot and 23-bit LSB after dot, which equals to 1.23 FIX values. K also contains 24-bit fraction as same as that of X and Y data. However, because K is always larger than zero, its sign bit becomes unnecessary. Due to those features, both FADD_SUB and FMUL_ki can be optimized by resource sharing and pipelining techniques. The final results are completed by FMUL_XYK as soon as *last* goes high.

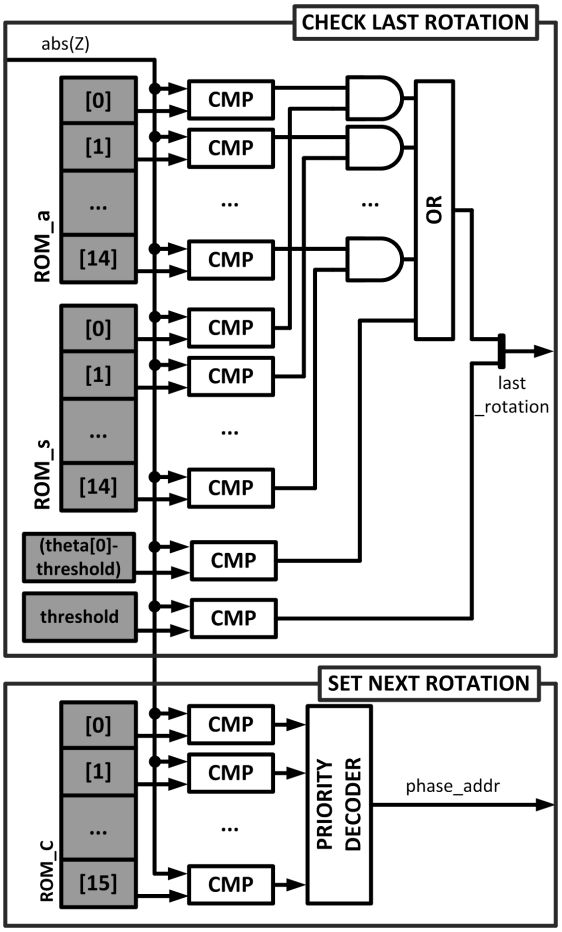


Fig. 6: The hardware architecture of CLR and SNR circuit.

Floating-point Adder Subtractor (FADD_SUB) calculates X and Y due to i and $signZ$ in each iteration, as illustrated in Fig. 9a. The initial values of X and Y are set as *one* and *zero*, respectively, immediately after FADD_SUB is reseted. Due to pipeline design, the most resources of Carry Look Ahead (CLA) adder, two?s complement (2C), and shifter are shared with. FADD_SUB is active by asserting *start* within two clock cycles while holding both $signZ$ and i. Simultaneously, *phase* becomes the control signal to multiplexing the data path during the operation. The 4-stage shifter performs a right shift operation with zeros fetched into empty MSB because of the fraction parts of X and Y. In order to compute 24-bit adder, six 4-bit CLA are connected together, as can be seen in Fig. 9b. The sign decision checks $signZ$, *phase*, and sign of previous X and Y data to decide the operation, addition or subtraction, in CLA. The 2C, then, will

Table 2: The values of ROM_θ, ROM_C, and ROM_K

Address	ROM_θ	ROM_C	ROM_K
0	0x168000	0x11E429	0x3F3504F3
1	0x0D4853	0x0A267B	0x3F64F92E
2	0x0704A3	0x054A52	0x3F785B42
3	0x039000	0x02ACE2	0x3F7E05EC
4	0x01C9C5	0x015770	0x3F7F805F
5	0x00E51B	0x00ABD8	0x3F7FE005
6	0x007295	0x0055F0	0x3F7FF800
7	0x00394B	0x002AF8	0x3F7FFE00
8	0x001CA5	0x00157C	0x3F7FFF80
9	0x000E52	0x000ABE	0x3F7FFFE0
10	0x000729	0x00055F	0x3F7FFFF8
11	0x000394	0x0002AF	0x3F7FFFFE
12	0x0001CA	0x000157	0x3F7FFFFF
13	0x0000E5	0x0000AB	0x3F7FFFFF
14	0x000072	0x000055	0x3F7FFFFF
15	0x000039	0x00002A	0x3F7FFFFF

correct the result in case it is a negative number. Finally, all of the information will produce the sign of the result to complete the process of the module.

Floating-point Multiplier ki (FMUL_ki) produces the length-factor K by multiplying each step-factor k_i in each iteration i, as shown in Fig. 9c. Upon resetting, *RegK* register is set as *one* and is sequentially updated by previous K and ROM_K that is depicted in Table 2. FMUL_ki also requires 2-clock-delay *start* for its pipeline computation.

Floating-point Multiplier XYK (FMUL_XYK) calculates the products of latest X/Y and K whose circuit is illustrated in Fig. 10a. The *last* signal is set within two clock cycles while remaining the X and Y to enable the FMUL_K.

$i = 1$
$threshold = ROM\_C(15)$
$last\_rotation = \{0, 0\}$
while $i < 16$ **do**
  if $|z| \leq threshold$ **then**
    $last\_rotation[0] = 1$
  else if $|z| \in [\theta(0) - threshold, \theta(0)]$ **then**
    $last\_rotation[1] = 1$
  else if $|z| \in [\theta(i) - threshold, \theta(i) + threshold]$ **then**
    $last\_rotation[1] = 1$
  end if
end while

Fig. 7: The pseudocode of angle selection function.

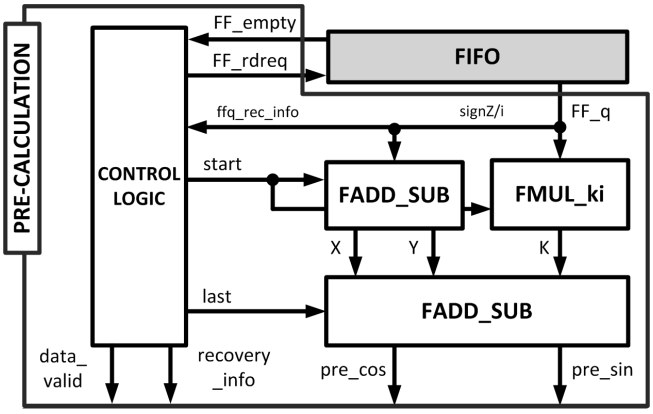


Fig. 8: The hardware architecture of PREC circuit.

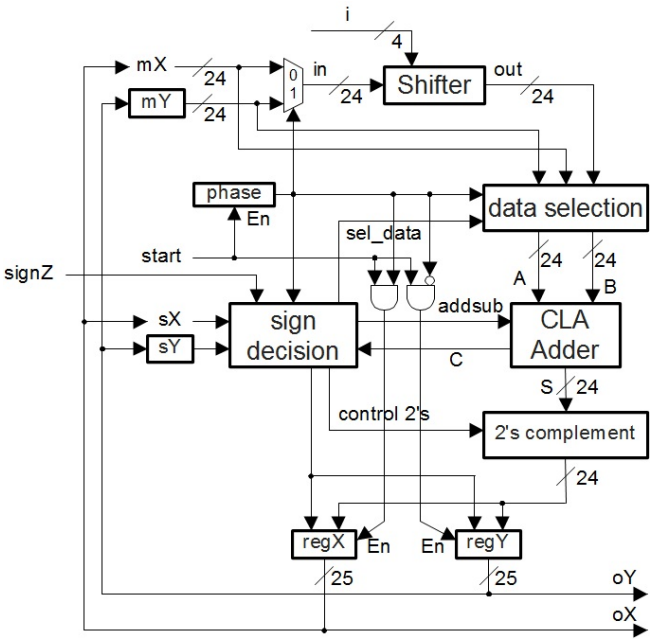
The raw cosine and sine results are ready in third and fourth clocks, respectively. The *Shift_count* and *Left_shift* modules subsequently produce the normalized value with exponent and fraction part from 48-bit product which is stored in Reg register. Two examples of *Shift_count* module are described in Fig. 10b. The 5-stage 5-bit *Left_shift* shares the similar design with the *Right_shift* of FADD_SUB. It can be seen that by employing parallel and pipeline processing and resource sharing techniques, latency, throughput, and hardware utilization are improved significantly.

Post-calculation (POSC) - The raw sine and cosine values are combined with *rec_info* to form the final trigonometric results. The recovery information *rec_info* given in Fig. 4b is utilized to select the suitable adjusted *pre_sin* or *pre_cos*.

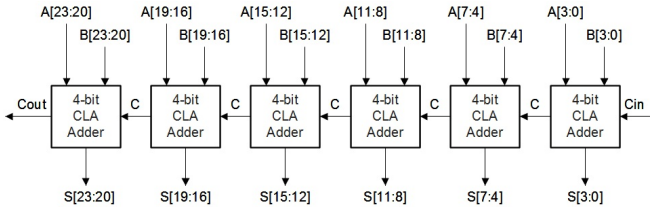
3 Appeal Points And Originality of HA-CORDIC

CORDIC [1]-[3] plays an essential role in most multimedia and wireless communication applications [4]-[5], where the evaluation of trigonometric functions are imperative. Fixed-point number representation is widely utilized in those systems due to its simple calculation and sufficient precision. However, some advanced applications such as Synthetic Aperture Radar (SAR) data processing [6], require not only highly precise results but also suitable format to manage wide dynamic range of numbers. In those systems, floating-point representation, instead of fixed-point, are literally deployed to retain real numbers resolution and accuracy effectively.

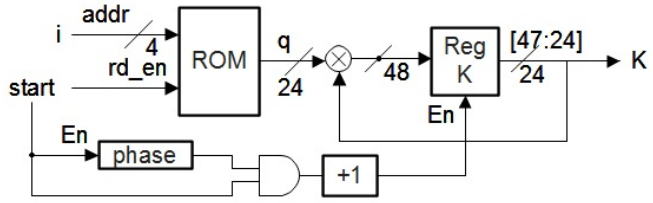
When it comes to the requirement for intensive computing and flexible configuration, Field-Programmable Gate Arrays (FPGAs) often have advantages over other solutions such as Digital Signal Processor (DSP) and Application-Specific



(a) The hardware architecture of FADD_SUB circuit.

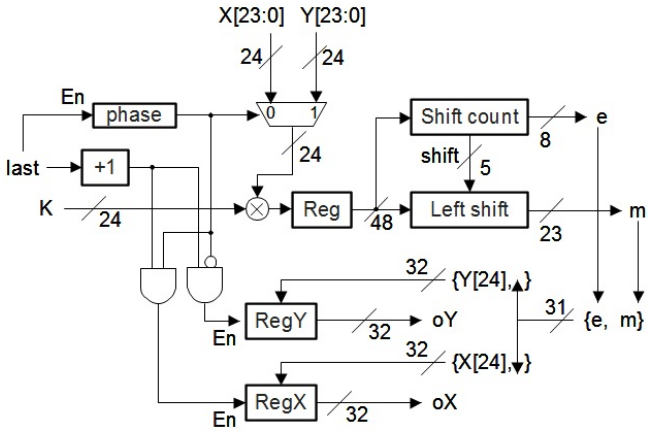


(b) An example 4-bit CLA adder.

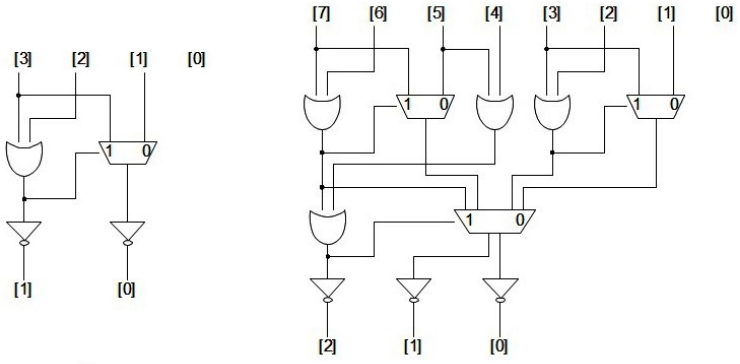


(c) The hardware architecture of FMUL_ki circuit.

Fig. 9: The hardware architecture of FADD_SUB and FMUL_ki circuit.



(a) The hardware architecture of FMUL_XYK circuit.



(b) An example of 4-bit input (left) and 8-bit input (right) Shift_count module.

Fig. 10: The hardware architecture of FMUL_XYK and shifter circuit.

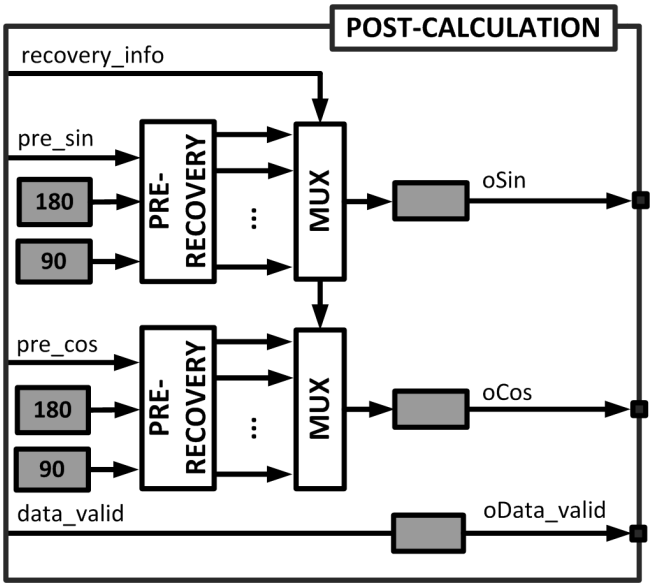


Fig. 11: The hardware architecture of POSC circuit.

Integrated Circuit (ASIC). Nevertheless, the implementation of floating-point arithmetics on FPGAs [7] still severely impact on the whole system latency and resource. Many approaches to highly efficient floating-point CORDIC, therefore, have been proposed recently. D. M. Munoz et al. [8] described a two-operation-mode floating-point CORDIC architecture that can compute the sine, cosine, or arctangent function. The design achieved an operating frequency of 86.1 MHz with elapsed time around 90 clock cycles at single-precision configuration. P. Surapong et al. [9] proposed an 8- and 16-stage pipelined floating-point CORDIC for phase and magnitude detector. 23% slice register and 38% slice lookup table (LUT) of Virtex-5 are required for this 16-stage system, whereas the maximum frequency is about 133.8 MHz. Nikhil Dhume et al.[10] and Jie Zhou [11] et al. presented a hybrid approach that firstly convert floating-point input into fixed-point format. A fixed-point CORDIC, then, computes the trigonometric functions and those results, lastly, are transformed into IEEE 754 floating-point format.

All improvements above, however, exhibits high and constant latency since the CORDIC algorithm always operates at a fixed number of iterations. In order to reduce error in results, more iterations must be performed. Moreover, the latency of an iterative algorithm is determined by the product of number of iterations and number of clock cycles in each iteration. Many approaches to enhance the precision without sacrificing the latency, therefore, have been increasingly attractive. Y. H. Hu et al. [12] proposed an Angle Recoding method which can reduce the total number of required elementary rotation angles by at least 50% without affecting the computational accuracy. Parallel angle recoding presented

by K. R. Terence et al. [13] improved the previous mechanism for hardware implementation. In this method, each angle associated with several contiguous range. However, this method add some overhead cycles before CORDIC iteration. Fayez Gebali et al. [14] recommends the method to skip some iterations in CORDIC algorithm but this method needs another CORDIC operation to scale the data.

In this paper, a low-resource low-latency hybrid adaptive CORDIC with floating-point precision is proposed. The originality of this research is described as follows.

- **Hybrid architecture** contains 24-bit fixed-point input angle in degree and 32-bit IEEE 754 floating-point sine/cosine outputs. This architecture aim to balance the accuracy of calculation with resource utilization.
- **Adaptive technique** not only decreases the number of rotation but also increases the precision in comparison with original CORDIC. Low-latency design obviously is one of crucial criteria in floating-point system.
- **Resource sharing technique** is implemented in floating-point arithmetic to reduce the logic utilization by around 70% in comparision with Altera library of the same function.
- **Parallel processing** is applied between fixed- and floating-point components and **pipeline processing** is deployed in each component to improve the throughput as well as latency.

4 Experimental Results of HA-CORDIC

The performance of our CORDIC is evaluated in two aspects: algorithm and hardware design. In the first assessment, we prove that our algorithm requires fewer iterations but achieves higher precision than the original CORDIC. In second assessment, we compare our design with the others in terms of latency and resource utilization. Moreover, the result is normalized by a 50-input EXOR circuit as LSI Design Contest [17] requirement.

4.1 Evaluation of Algorithm

In order to assess the algorithm, 9,001 angles, from $[-45^o, 45^o]$ with step width of 0.01, are generated. The total number of iterations are observed at three different predetermined angle constants - N. As can be seen in Fig. 12, our HA-CORDIC only requires maximum 4, 6, and 8 micro-rotations in case $N = 8, 12, 16$, respectively. In other words, in worst case, the HA-CORDIC is still $2.7X$, $2.4X$, and $2.3X$ faster than the original one. Moreover, on average, the latency improvements are 3.8X, 3.5X, and 3.4X at $N = 8, 12, 16$, respectively.

The precision between two algorithms is measured by mean square error (MSE) of the residual angles. In fact, the more the angle approximates to zero, the more the precision of sine/cosine can be attained. It can be seen in Fig 13, with the variation of N from eight to 16, the precision of both methods increase. However, our CORDIC always delivers smaller MSE value, 2.51e-7, than that from conventional CORDIC, 1.02e-6.

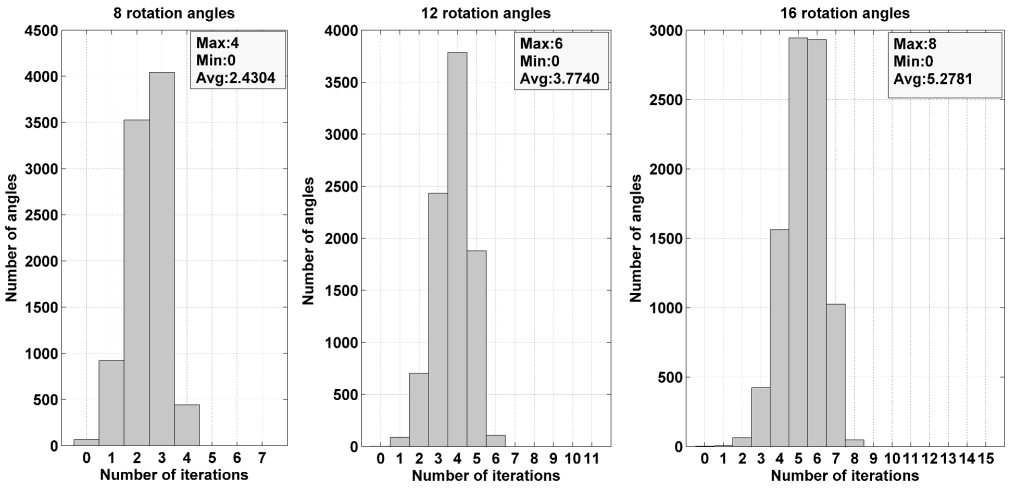


Fig. 12: The comparison in number of iterations in case of 8, 12, and 16 predetermined angle constants.

Table 3: The comparison between HA-CORDIC with the others

	[8]	[9]	[15]	[10]	[11]	[16]	Ours
Device Family	Xilinx Virtex 5	Xilinx Virtex 5	Xilinx Virtex 6	Xilinx Virtex 7	Altera Stratix II	Altera Stratix IV	Altera Stratix IV
Latency (clocks)	93	–	–	130	–	36	12/20/26
Frequency (MHz)	86	133	253	281	195	258	175.7
Lookup Table	3,152	13,723	13,744	6,514	6,469	5,612	1,139
Registers	–	8,346	–	4,725	5,372	4,231	498
Memory Bits	2,832	–	–	4,894	–	3,575	11
DSP Block	0	2	96	9	–	32	8

4.2 Evaluation of Hardware

The proposed HA-CORDIC is synthesized by Altera Quartus 14.0 with Stratix IV FPGA target. The resource utilization and operating frequency among HA-CORDIC and the others are illustrated in Table 3. Unlike original CORDIC methods, HA-CORDIC latency is varies due to the dynamic rotation. At $N = 16$, it costs 12, 20, and 26 clock cycles in best (zero or one rotation), typical (five rotations), and worst case (eight rotations). Besides, HA-CORDIC logic utilization is much a lot less than the others as regards the operating frequency of 175.7 MHz.

Table 4 shows our design in comparison with a 50-input EXOR circuit which is compulsory in the LSI Design Contest. According to Altera TimeQuest Timing

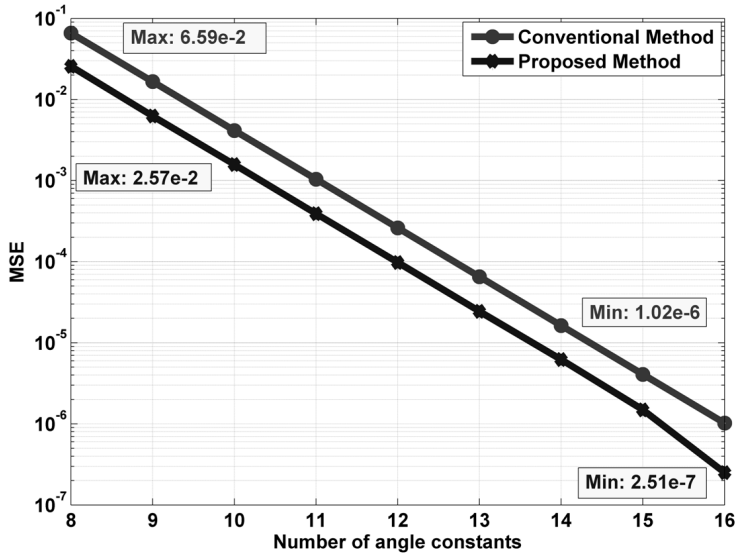


Fig. 13: The comparison in mean square error of residual angles.

Table 4: The comparison between HA-CORDIC with the others

	Combinational logic	**Propagation delay (ns)**
HA-CORDIC	1,139	5.69
50-input EXOR	11	11.52
Ratio (%)	103.5	0.49

Analyzer, EXOR circuit propagation delay is 11.52 ns, whereas HA-CORDIC suffers 5.69 ns (1/175.7 MHz).

The sine/cosine results of 17 angles recommended by the LSI Design Contest are shown in Table 5. Our HA-CORDIC takes four iterations in most cases. Moreover, the errors between our sine/cosine results with corresponding true values are almost negligible.

5 Verilog HDL code

The Verilog HDL source code of proposed HA-CORDIC is in **HDL_CODE** folder. The summary of this code is illustrated as follows.

Folder ROM_INIT: Initial values for ROMs

cordic_fixedpoint_anglenormalize_cmp_rom_value.txt
cordic_fixedpoint_anglenormalize_rom_value.txt
cordic_fixedpoint_check_last_rotation_add_threshold_rom.txt
cordic_fixedpoint_check_last_rotation_sub_threshold_rom.txt

Table 5: The example results of HA-CORDIC

Input angle	**Output results**					**Difference in error with**	
	Iterations	**oData_sin**		**oData_cos**		**true cos [17]**	**true sin [17]**
		HEX	**DEC**	**HEX**	**DEC**		
$\frac{\pi}{2}$ (90^o)	0	0x3f800000	1.000000000	0x00000000	0.000000000	6.12323E-17	0.000000000
$\frac{\pi}{2} - \frac{\pi}{256}$ (89.296875^o)	4	0x3f7ffb10	0.9999247	0x3c48f951	0.012266473	5.06529E-06	4.184E-08
$\frac{\pi}{2} - \frac{\pi}{128}$ (88.59375^o)	4	0x3f7fec3c	0.9996984	0x3cc925c2	0.024554137	1.29085E-05	4.187E-07
$\frac{3}{8}\pi$ (67.5^o)	5	0x3f6c83bc	0.9238851	0x3ec3ed42	0.38266948	1.39524E-05	5.56749E-06
$\frac{\pi}{4}$ (45^o)	1	0x3f3504f3	0.70710677	0x3f3504f3	0.70710677	1.119E-08	1.119E-08
$\frac{\pi}{8}$ (22.5^o)	5	0x3ec3ed42	0.38266948	0x3f6c83bc	0.9238851	5.56749E-06	1.39524E-05
$\frac{\pi}{128}$ (1.40625^o)	4	0x3cc925c2	0.024554137	0x3f7fec3c	0.9996984	4.187E-07	1.29085E-05
$\frac{\pi}{256}$ (0.703125^o)	4	0x3c48f951	0.012266473	0x3f7ffb10	0.9999247	4.184E-08	5.06529E-06
0 (0^o)	0	0x00000000	0.000000000	0x3f800000	1.0000000	0.00000000	0.00000000
$-\frac{\pi}{256}$ (-0.703125^o)	4	0xbc48f951	-0.012266473	0x3f7ffb10	0.9999247	4.184E-08	5.03229E-06
$-\frac{\pi}{128}$ (-1.40625^o)	4	0xbcc925c2	-0.024554137	0x3f7fec3c	0.9996984	4.187E-07	1.29085E-05
$-\frac{\pi}{8}$ (-22.5^o)	5	0xbec3ed42	-0.38266948	0x3f6c83bc	0.9238851	5.56749E-06	1.39524E-05
$-\frac{\pi}{4}$ (-45^o)	1	0xbf3504f3	-0.70710677	0x3f3504f3	0.70710677	1.119E-08	1.119E-08
$-\frac{3}{8}\pi$ (-67.5^o)	5	0xbfc83bc	-0.9238851	0x3ec3ed42	0.3826695	1.39524E-05	5.56749E-06
$-\frac{\pi}{2} - \frac{\pi}{128}$ (-88.59375^o)	4	0xbf7fec3c	-0.9996984	0x3cc925c2	0.024554137	1.29085E-05	4.187E-07
$-\frac{\pi}{2} - \frac{\pi}{256}$ (-89.296875^o)	4	0xbf7ffb10	-0.9999247	0x3c48f951	0.012266473	5.03229E-06	4.184E-08
$-\frac{\pi}{2}$ (-90^o)	0	0xbf800000	-1.000000000	0x00000000	0.000000000	6.12323E-17	0.000000000

cordic_fixedpoint_get_phase_addr_rom_C.txt
cordic_fixedpoint_rom_C.txt
cordic_fixedpoint_updatephase_rom_theta.txt
cordic_floatingpoint_mul_ki_rom.txt
file_input_angle.txt

Folder SRC_CODE: Top-level file
cordic_system.v

Folder SRC_CODE/ASEL: Angle normalization and selection
cordic_fixedpoint.v
cordic_fixedpoint_anglenormalize.v
cordic_fixedpoint_anglenormalize_cmp.v
cordic_fixedpoint_anglenormalize_encoder.v
cordic_fixedpoint_anglenormalize_rom.v
cordic_fixedpoint_check_last_rotation.v
cordic_fixedpoint_control_logic.v
cordic_fixedpoint_get_phase_addr.v
cordic_fixedpoint_get_phase_addr_priority_4_to_2.v
cordic_fixedpoint_get_phase_addr_priority_16to4.v
cordic_fixedpoint_get_phase_addr_rom_C_cmp.v
cordic_fixedpoint_updatephase.v
cordic_fixedpoint_updatephase_rom_theta.v

Folder SRC_CODE/FIFO: Fifo
cordic_fifo.v

Folder SRC_CODE/PREC: Pre-calculation
cordic_floatingpoint.v
cordic_floatingpoint_addsub.v
cordic_floatingpoint_addsub_CLA_4bits.v
cordic_floatingpoint_addsub_CLA_adder.v
cordic_floatingpoint_addsub_Complement_2s.v
cordic_floatingpoint_addsub_Right_shifter.v
cordic_floatingpoint_control_logic.v
cordic_floatingpoint_mul_K.v
cordic_floatingpoint_mul_K_Left_shifter.v
cordic_floatingpoint_mul_K_Shift_count.v
cordic_floatingpoint_mul_ki.v
cordic_floatingpoint_mul_ki_rom.v

Folder SRC_CODE/POSC: Post-calculation
cordic_recovery_sin_cos.v

Folder TESTBENCH: Testbench
cordic_system_tb.v

cordic_fixedpoint_input_phase_rom.v

6 The Display Of Simulation Waveforms

The waveform of HA-CORDIC hardware in case angle of 30^o is shown in Fig. 14.

- **(1)**: a 24-bit fixed-point angle input *Data* is put into HA-CORDIC upon *iData_valid*=1.
- **(2)**: ANOR normalizes the angle into conventional range $[0^o, 45^o]$. *oPhase_norm* and *phase_normalize_info* exhibit the normalized angle and recovery information, respectively. Then, ASEL searchs for proper angles and reckons the sign of residual angles. *phase_addr* indicates the memory addresses of selected angles, $\{1, 4, 9, 11, 15\}$ while *phase_sign* shows the residual sign, $\{1, 1, -1, -1, -1\}$.
- **(3)**: X, Y, and K are updated in sequence by *ffq_phase_addr* (or i) and *ffq_phase_sign* (or *signZ*) obtained from FIFO. The addition and subtraction of X, Y and multiplication of K are processed in parallel by *start* and *last* signals. *oPre_cos* and *oPre_sin* show the cosine and sine results, respectively.
- **(4)**: after recovery, *oCos* and *oSin* are sent out with *oData_valid*. Sine value is one-clock delay by comparison with Cosine. It can be seen that the computation costs five rotations or 20 clock cycles in total.

In addition, three examples at zero (input angle 0^o), one (input angle 45^o), and eight (input angle 19.74^o) iterations are illustrated in Fig. 15.

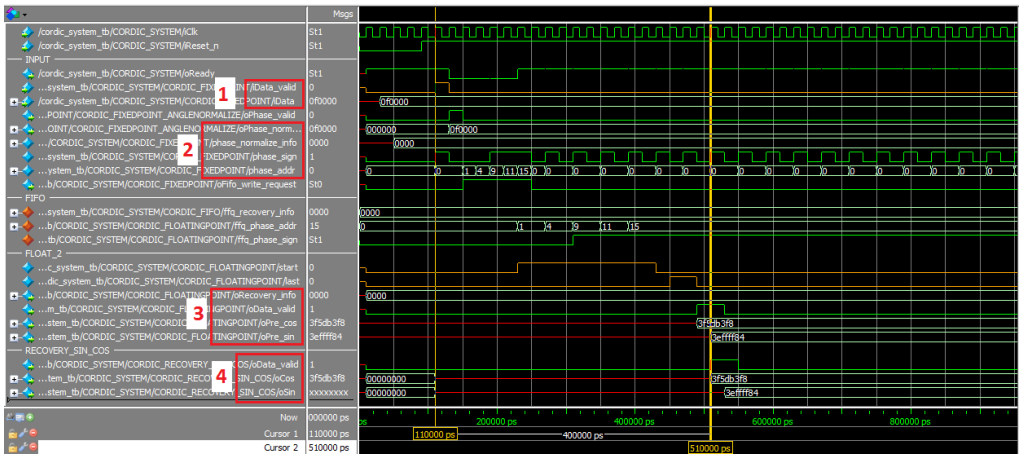
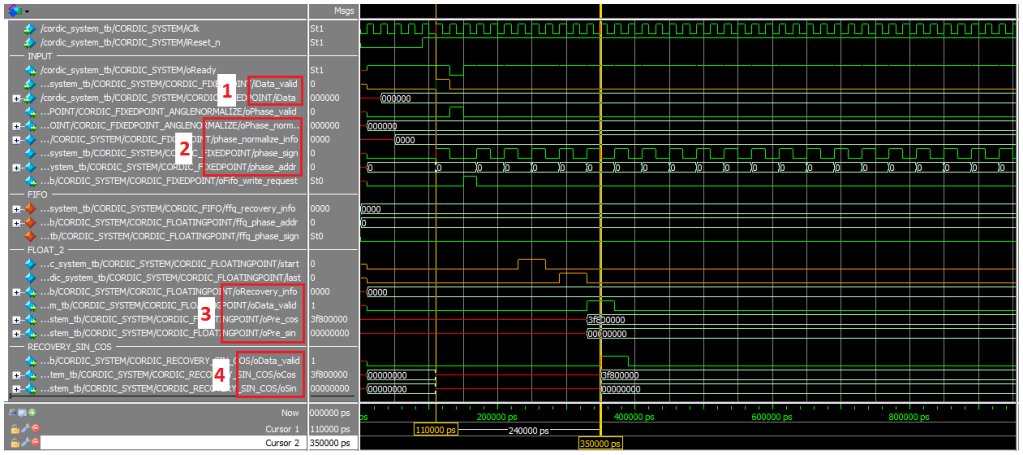
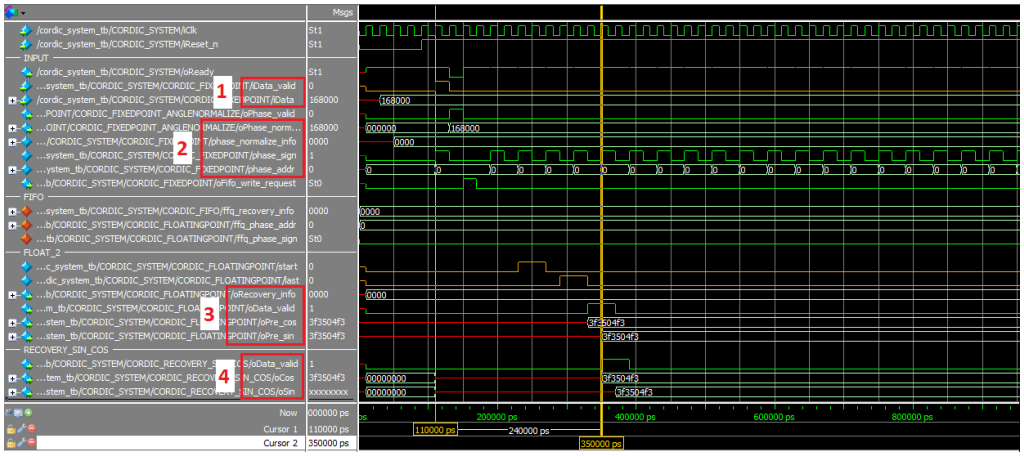


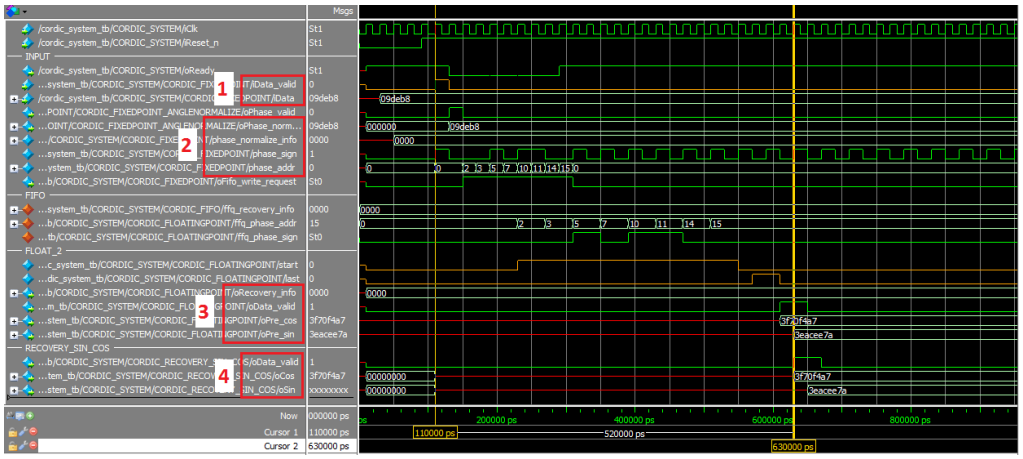
Fig. 14: An example of angle 30^o.



(a) The waveform of HA-CORDIC at 0^o.



(b) The waveform of HA-CORDIC at 45^o.



(c) The waveform of HA-CORDIC at 45^o.

Fig. 15: The example of angle 0^o, 45^o, and 19.74^o.

References

1. J.E. Volder, *The CORDIC Trigonometric Computing Technique*, IRE Trans. Electronic Computers, Vol. EC-8, pp. 330 - 334, 1959.
2. J.S. Walther, *A unified algorithm for elementary functions*, Proc. Spring Joint Computer Conf., pp. 379 - 385, 1971.
3. P.K. Meher, J. Valls, Tso-Bing Juang, K. Sridharan, Koushik Maharatna, *50 Years of CORDIC: Algorithms, Architectures, and Applications*, IEEE Trans. Cirt. Syst. I: Regular Papers, Vol. 56, Iss. 9, pp. 1893 - 1907.
4. A. M. Despain, *Fourier Transform Computers Using CORDIC Iterations*, IEEE Trans. Computers, Vol. C-23, Iss. 10, pp. 993 - 1001, 1974.
5. E. H. Wold, A. M. Despain, *Pipeline and Parallel-Pipeline FFT Processors for VLSI Implementations*, IEEE Trans. Computers, Vol. C-33, Iss. 5, pp. 414 - 426, 1984.
6. R. Touzi, R. K. Hawkins, S. Cote, *High-Precision Assessment and Calibration of Polarimetric RADARSAT-2 SAR Using Transponder Measurements*, IEEE Transac. Geoscience and Remote Sensing, Vol. 51, Iss. 1, pp. 487 - 503, 2013.
7. G. Govindu, Ling Zhuo, Seonil Choi, V. Prasanna, *Analysis of high-performance floating-point arithmetic on FPGAs*, Proc. 18th Int. Parallel and Distributed Processing Symp., 2004.
8. D. M. Munoz, D. F. Sanchez, C. H. Llanos, M. Ayala-Rincn *FPGA Based Floating-Point Library for Cordic Algorithms*, VI Southern Programmable Logic Conf. (SPL), pp. 55 - 60, Brazil, 2010.
9. Pongyupinpanich Surapong, and Manfred Glesner, *Pipelined Floating-Point Architecture for a Phase and Magnitude Detector Based on CORDIC*, Int. Conf. Field Programmable Logic and Applications (FPL), pp. 382 - 384, 2011.
10. Nikhil Dhume and Ramakrishnan Srinivasakannan, *Xilinx Application Note: Parameterizable CORDIC-Based Floating-Point Library Operations*, 2012.
11. Jie Zhou, Yazhuo Dong, Yong Dou, Yuanwu Lei, *Dynamic Configurable Floating-Point FFT Pipelines and Hybrid-Mode CORDIC on FPGA*, Int. Conf. Embedded Software and Systems (ICESS), pp. 616 - 620, China, 2008.
12. Y. H. Hu, S. Naganathan, *Angle recording method for efficient implementation of the CORDIC algorithm*, IEEE Int. Symp. Cirt. Syst., Vol. 1, pp. 175 - 178, 1989.
13. Terence K. Rodrigues and Earl E. Swartzlander, *Adaptive CORDIC: Using Parallel Angle Recoding to Accelerate Rotations*, IEEE Trans. Comp., Vol. 59, No. 4, pp. 522 - 531, 2010.
14. Fayez Gebali, *Floating-point adaptive cordic (FPA-CORDIC) algorithm for elementary function calculation*, Proc. 2011 IEEE National Aerospace and Elec. Conf. (NAECON), pp. 46 - 50, 2011, USA.
15. Yuanwu Lei, Yong Dou, Song Guo, Jie Zhou, *FPGA Implementation of Variable-Precision Floating-Point Arithmetic*, LNCS Advanced Parallel Processing Technologies, Vol. 6965, pp. 127 - 141, 2011.
16. Altera, *Floating-Point Megafunctions User Guide*, 2013.
17. LSI Design Contest 2015, *Input and Output Specification*, `http://www.lsi-contest.com/spec3/graph6-1e.png`.