SINE WAVE GENERATORS ON FPGA

Marabotto Miriana
Department of Physics
Università degli Studi di Torino
miriana.marabotto@edu.unito.it

Sachero Selene
Department of Physics
Università degli Studi di Torino
selene.sachero@edu.unito.it

***Abstract*—The aim of this paper is to describe two different methods to obtain a sine wave on FPGA, using Verilog language.**

In both cases the CORDIC algorithm has been used.
The first method involves the implementation of the algorithm itself, while the second one makes use of the IP offered by Xilinx Vivado.

The paper will show the main differences between the two and compare the results.

I. INTRODUCTION

Trigonometric functions, as well as logarithm, square root and other transcendental functions, can be computed digitally using CORDIC.

CORDIC (CO-ordinate Rotation DIgital Computer) is a useful and simple algorithm, known also as Volder's algorithm, that uses only simple operations such as additions, subtractions, lookup tables (LUT) and bitshifts to perform several computing tasks.
CORDIC is an iterative fixed-point technique that at every iteration achieves one more bit of accuracy.

The aim of this project is to design a CORDIC processor that is able to reproduce a sine wave by generating angles and to compute the respective sine values using simple iterative relationships. This method will be compared to the one that involves the use of the Xilinx CORDIC v6.0 LogiCORE IP in Sin and Cos functional configuration.

The code is written in Verilog language and the simulation is achieved using Xilinx Vivado Design Suite.

In section 2 the CORDIC algorithm is explained; in section 3 the first method is described and the results are presented in fig. 4, where Xilinx Vivado simulation window is shown. Section 4 explains how the CORDIC IP has been used, the MQN notation required for the inputs and the outputs and the results are shown in fig. 9.

II. CORDIC ALGORITHM

The main idea of CORDIC algorithm is to describe a digital equivalent of an analog vector, which is rotated by a given angle (θ) to obtain a new vector with the same magnitude.

The starting vector is $V_0 = [X_0, Y_0]$ and the new one is $V_1 = [X_1, Y_1]$.

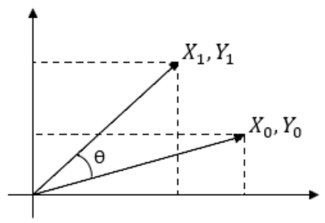

Fig. 1. Graphical representation of the rotation.

X_1 and Y_1 are calculated as shown in equation (1).

$$\begin{aligned} X_1 &= cos(\theta)[X_0 - Y_0 tan(\theta)] \\ Y_1 &= cos(\theta)[Y_0 + X_0 tan(\theta)] \end{aligned} \tag{1}$$

Since the fastest hardware is necessary, the aim is to compute the sine without using multiplications or trigonometric functions. For this reason only values of $tan(\theta) = \pm 2^{-i}$ are permitted.
In table I some values are shown.

i	$tan(\theta)$	θ
0	1	45
1	1/2	26.565
2	1/4	14.036
3	1/8	7.125
4	1/16	3.576
5	1/32	1.79
6	1/64	0.895

TABLE I

This assumption ensures that, after the i_{th} pseudo rotation, the new coordinates X_i and Y_i are calculated by the simple process of shifting and adding, as shown in equation (2).

$$\begin{bmatrix} X_{i+1} & Y_{i+1} \end{bmatrix} = K_i \begin{bmatrix} X_i - Y_i d_i 2^{-i} & Y_i + X_i d_i 2^{-i} \end{bmatrix} \tag{2}$$

Where K_i and d_i are defined as shown in equation (3). The variable d_i expresses the direction of the rotation at every step.

$$\begin{aligned} K_i &= \frac{1}{\sqrt{1 + (\pm 2^{-i})^2}} \\ d_i &= \pm 1 \end{aligned} \tag{3}$$

In this way it is possible to compute only the angles shown in table I, but the purpose is to calculate any possible angle therefore an angle accumulator Z is necessary.
It is modified as follows (4):

$$Z_{i+1} = Z_i - d_i tan^{-1}(2^{-i}) \tag{4}$$

Z is the desired rotation angle, each iteration of the CORDIC algorithm is applied so that the magnitude of Z reaches the zero value. Therefore, if $Z_i > 0$ ($d_i = +1$) the current iteration angle (θ_i) is subtracted from Z_i, otherwise it is added to Z_i. Also appropriate X and Y calculations are done.

Since every $[X_{i+1}\ Y_{i+1}]$ coordinates are obtained by a multiplication of the $[X_i\ Y_i]$ input coordinates with the rotation matrix (2), V_n is computed as shown in equation (5).

$$\begin{bmatrix} X_n & Y_n \end{bmatrix} = \prod K_i \begin{bmatrix} X_0 & Y_0 \end{bmatrix} \prod \begin{bmatrix} 1 & d_i 2^{-i} \\ -d_i 2^{-i} & 1 \end{bmatrix} \quad (5)$$

To reach the final equations, the scale constant K_i is removed.
Without this value the rotation algorithm will have a gain, A_n. The exact value of the gain depends on the number of iteration, as shown in equation (6).

$$A_n = \prod_{i=0}^{n-1} \sqrt{1 + 2^{-2i}} \quad (6)$$

So the final equations are (7):

$$\begin{aligned} Y_n = A_n[Y_0 cos(Z_0) + X_0 sen(Z_0)] \\ Z_n = 0 \\ Y_0 = 0 \\ X_0 = 1/A_n \end{aligned} \quad (7)$$

These CORDIC equations are limited to rotation angles (θ) in the range -$\frac{\pi}{2} < \theta < \frac{\pi}{2}$ due to the use of 2^0 for the tangent of the first iteration. For composite rotation angles, outside this range, an additional rotation is required before using CORDIC algorithm.

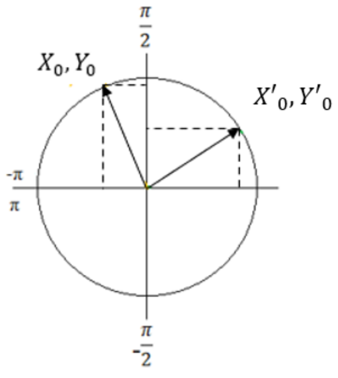

Fig. 2. Graphical representation of the additional rotation.

The following calculations must be done:

$$\begin{aligned} X_0^{'} &= -dY_0 \\ Y_0^{'} &= dX_0 \\ Z_0^{'} &= Z_0 - d\frac{\pi}{2} \end{aligned} \quad (8)$$

Where $d = \begin{cases} +1 & Y_0 > 0 \\ -1 & otherwise \end{cases}$

$X_0^{'}$ and $Y_0^{'}$ are the new coordinates.

III. CORDIC IMPLEMENTATION IN VERILOG

CORDIC implementation requires two modules: the core one and the testbench.

A. *Core module*

- *definition of useful variables:*
  The first step is to instantiate input and output ports as shown in fig. 3.

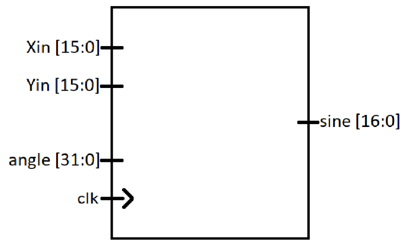

Fig. 3. Core module diagram

According to $Z_{i+1} = Z_i - d_i tan^{-1}(2^{-i})$ an arctangent should be computed, to avoid this calculation a lookup table is used, one entry for each iteration.
This set of constants is called $[0:31] atan\_table[0:30]$, each constant represents a specific angle in the range $[0, \frac{\pi}{4}]$ written using 32 bits.
Example:
atan_table[00] = 32'b00100000000000000000000000000000 stands for 45°.

The second step is to define X,Y and Z registers. It is useful to define a set of registers for each stage because in this way the iteration is pipelined, so at every clock cycle new calculations can be done.

- *pre rotation of the vector:*
  The first two bits of $angle$ represent the quadrant in which it lies.
  When the initial vector is in the first (2'b00) or in the fourth (2'b11) quadrant no pre rotation is needed because the angles are in the range -$\frac{\pi}{2} < \phi < \frac{\pi}{2}$.
  In the second quadrant (2'b01) $\frac{\pi}{2}$ has to be subtracted from angle to bring it back in the range of $0 < \phi < \frac{\pi}{2}$, while in the third (2'b10) it has to be added so to bring angle in the range of -$\frac{\pi}{2} < \phi < 0$.
  In each quadrant the initial values of the registers are set according to equations (8).
  For example, in quadrant 2'b01:
  - $Y_0 > 0$ so $d = +1$
  - $X[0]$ = -Y_{in}
  - $Y[0]$ = X_{in}

- *logic part of the code :*
  In this part a for loop is used to write the code to do the calculations shown in equations (2) and (4).
  As said, CORDIC is an iterative fixed-point technique that at every iteration achieves one more bit of accuracy, so two variables X_{shift} and Y_{shift} are defined to compute X_{i+1} and Y_{i+1}.

X_{shift} and Y_{shift} are shifted right at every stage of the for loop, thus one more bit is achieved at every step.

B. *Test bench*

The test bench is a stimulus file necessary to drive the input signal of the core module.
Firstly, the inputs and outputs are defined and a local parameter is used to choose the sine's amplitude.

Then the initial block is instantiated, where the stimulus are defined using a for loop, which goes from the angle zero (i=0) to 360 degrees, incrementing of 1 degree in order to have a 1 degree resolution. In this for loop, at every positive edge of clock, the new variable *angle* is set in the following way: the angle (i) is multiplied by 2^{32} and divided by 360 degrees.

The DUT is instantiated and all the ports are connected.

C. *Results*

The results are shown in fig.4. The trend is as expected.

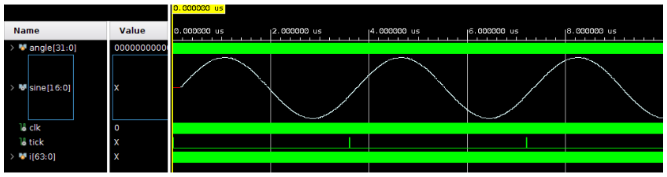

Fig. 4. The sine wave with an amplitude of 256.

The tick is used to restart the calculations. After 360 clock cycles the tick is high and the algorithm restarts, in this way more than one sine wave's period is computed.

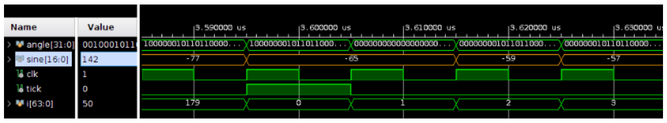

Fig. 5. The input tick is high

As an example, in fig. 6 the sine value of θ =30° is shown in orange:
$sin_{signeddecimal}(\theta) = 127 \quad \rightarrow \quad sin(\theta) = 0.4961.$
The result is consistent with the expected value.

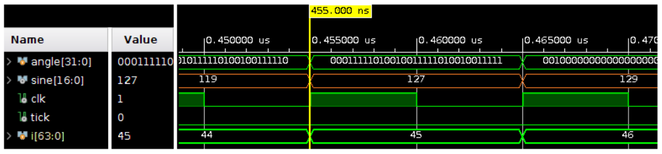

Fig. 6. The sine value of $\theta = \dfrac{\pi}{6}$

IV. CORDIC IP

Xilinx CORDIC v6.0 LogiCORE IP is a useful instrument to perform different tasks, such as calculation of trigonometric functions, square root and hyperbolic functions.
For this project Sin and Cos functional configuration has been used, with which sine and cosine values of a certain input angle can be obtained.

The core requires a specific notation: the input, PHASE_IN, and the outputs, X_OUT and Y_OUT, are represented as fixed-point twos complement numbers, as described below.

A. *Q numbers format*

An MQN format number is represented using 1+M+N bits: the first one is the sign bit, 0 is for positive numbers and 1 for negative ones; the next M bits are the integer ones and are followed by N fractional bits, as shown in fig. 7.

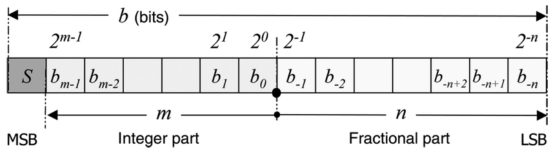

Fig. 7. Q numbers representation

The input port is dedicated to the phase θ, which is in the range $[-\pi; +\pi]$ and two integer bits are required to represent it, asking for the 2QN notation.
The output port gives $\{sin(\theta), cos(\theta)\}$ as a sequence of bits in the 1QN notation because the range of this values is $[-1; +1]$ and only one integer bit is needed.

The process to convert a decimal number y to the MQN format is described below.

- Positive numbers: scale y to an integer i

$$i = y \cdot 2^N \tag{9}$$

  then approximate i and convert it to binary.
- Negative numbers: take the absolute value of y and write it as

$$i = 2^{1+X+N} - (|y| \cdot 2^N) \tag{10}$$

  then approximate i and convert it to binary.

Example of 2Q7 format (used in the project):
$y = \pi/4 = 0.785 \qquad \rightarrow \qquad 0001100100$
$y = -\pi/4 = -0.785 \qquad \rightarrow \qquad 1110011011$

B. *Wrapper and testbench for the IP core*

The first module to be implemented is the wrapper for the IP core, where the ports are declared and connected to the IP (fig. 8).

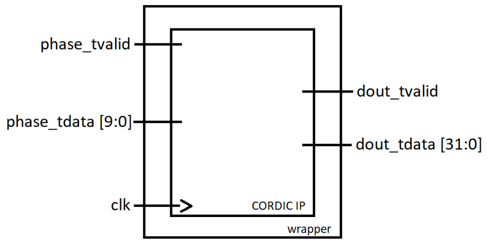

Fig. 8. Core module diagram

The testbench is implemented next by declaring the ports, instantiating the clock and the DUT.
The input phase values, in radians, are written in the 2Q7 format (10 bits) differentiating from each other by 5 degrees in order to have a 5 degrees resolution. At every positive edge of the clock a new input phase value is read from the ROM. The output is a concatenation of sine and cosine values written in the 1Q14 format as $\{sine[15:0], cosine[15:0]\}$.

C. Results

The result of the simulation is shown in fig. 9, where the sine and cosine waves are visible.

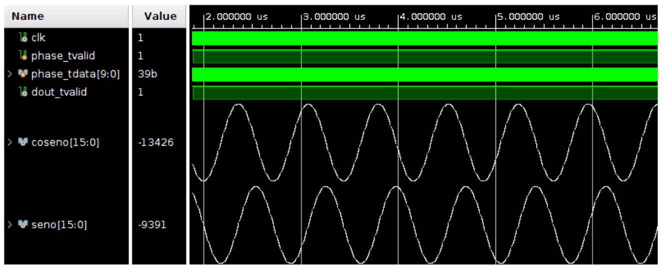

Fig. 9. Sine and Cosine obtained with the IP

As an example, in fig. 10 sine and cosine values of $\theta = 45°$ are shown in orange:
$sin_{1Q14}(\theta) = 0010110100010000 \rightarrow sin(\theta) = 0.704$
$cos_{1Q14}(\theta) = 0010110101110001 \rightarrow cos(\theta) = 0.710.$

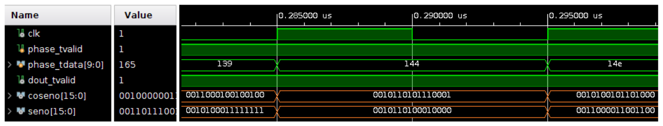

Fig. 10. Sine and Cosine values of $\pi/4$

In fig. 11 the results of $\theta = 30°$ are shown in orange after the yellow line which reports $255ns$:
$sin_{1Q14}(\theta) = 0001111110001110 \rightarrow sin(\theta) = 0.493$
$cos_{1Q14}(\theta) = 0011011110101110 \rightarrow cos(\theta) = 0.870.$

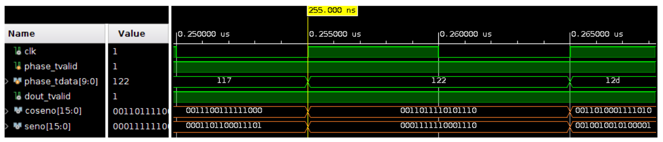

Fig. 11. Sine and Cosine of $\pi/6$

The results are consistent with the expected values, even though it is possible to notice a little discrepancy between these values and the actual ones. In particular, in both cases the sine is smaller and the cosine is bigger.

V. Conclusions

In conclusion, both methods are valid to compute a sine wave on FPGA.

The first makes use of the iterative relationships defined by CORDIC algorithm and requires some additional lines of code to rotate the needed angle in the right range of work and initialize the vectors.
The final result is as expected and the sine values are consistent with the actual ones, showing that the algorithm works fine.

The second is more intuitive and simpler to implement.
The inputs has been written and saved on a ROM. This stage requires a longer time with respect of the first method, which performs automatically the calculations.
The results show a little discrepancy of the values for sine and cosine, but are consistent regardless with the expected ones.

References

[1] Bibek Bhattarai, ***FPGA Implementation of CORDIC Processor***, researchgate.net, September 2013
[2] K. Masselos, V. Giannakopoulou, ***Hardware performance analysis of a parametric CORDIC IP***, researchgate.net, September 2012
[3] Sriram Madala, Anushuman Mishra, ***SINE AND COSINE GENERATOR USING CORDIC ALGORITHM IMPLEMENTED IN ASIC***, researchgate.net, June 2015
[4] Xilinx, LogiCORE IP product guide, ***CORDIC v6.0 (PG105)***, 2017
[5] https://miscircuitos.com/sinus-wave-generation-with-verilog-using-vivado-for-a-fpga/
[6] https://zipcpu.com/dsp/2017/08/30/cordic.html
[7] https://youtu.be/TJe4RUYiOIg
[8] https://kierdavis.com/cordic.html