CorALU Group

Web Report (summary)

New Wannasatit (nw@ualberta.ca)

Ruud Benjaminsen (benjamins@ee.ualberta.ca)

Rev 1.0 **December, 2001**

Declaration of original content

The design elements of this project and report are entirely the original work of the authors and have not been submitted for credit in any other course except as follows:

Summary of the CORDIC algorithm [1]

New Wannasatit ______________

Ruud Benjaminsen ______________

Date ______________

Abstract

The goal of this project is to implement (add) a customized arithmetic logical unit (CorALU) for an Altera based Nios-embedded processor to compute various functions from the set of trigonometric, linear and exponential functions.

The CorALU functionality is based on a collection of algorithms commonly known as CORDIC (Coordinate Rotation Digital Computer). The interesting feature of these algorithms is that the computation involved only uses shifts and adds in an iterative solution. Traditionally, the computation involving the same functions would have used some kind of power series solution, which in most cases uses regular multipliers and adders.

Many hardware implementations of the CORDIC algorithm exist. Examples of these can be found in the field of robotics, digital signal processing and scientific calculators.

Table of contents

Achievements	5
Description of operation (CORDIC)	6
Description of operation (bit-parallel design)	7
Characteristics	8
References	9

Achievements

A bit-parallel design using one of the six possible CORDIC operation modes has been implemented, simulated and shown to be working in an hardware implementation.

There was not enough time to simulate a bit-serial design or to add the bit-parallel design to the Nios CPU system.

The bit-parallel design operates in the rotation mode of the CORDIC algorithms using a circular system.

Description of operation (CORDIC) [1]

The CORDIC algorithm is an iterative solution to many functions (like trigonometric, linear and hyperbolic) using only shifters and adders. There are two possible modes of operation in the algorithms, the rotation mode, which tries to rotate a vector by a certain angle, and the vector mode, which tries to rotate a vector to the x-axis noting the angle needed for this rotation.

The rotation of a vector in a Cartesian plane by an angle is given by:

$$x' = x\cos f - y\sin f$$
$$y' = y\cos f + x\sin f$$

We can rearrange these two equations so that:

$$x' = \cos f \cdot [x - y\tan f]$$
$$y' = \cos f \cdot [y + x\tan f]$$

If we limit the angle of rotation so that $\tan f = \pm 2^{-i}$, the multiplication by the tangent term becomes just a shift operation (determined by i, an iteration counter). Successively smaller elementary rotations allow for computation of arbitrary rotation angles.
Making the decision is which direction to rotate rather than whether to rotate, the cosine term becomes a constant ($\cos(a) = \cos(-a)$). The iterative rotation is expressed as a shift-add algorithm for vector rotation, with a scale constant K (you could call this the gain of the system):

$$x_{i+1} = K_i[x_i - y_i \cdot d_i \cdot 2^{-i}] \qquad d_i = \pm 1$$
$$y_{i+1} = K_i[y_i + x_i \cdot d_i \cdot 2^{-i}] \qquad K_i = \cos(\tan^{-1} 2^{-i})$$

An additional adder-subtractor that accumulates the elementary rotation angle for each iteration adds a third equation:

$$z_{i+1} = z_i - d_i \cdot \tan^{-1} 2^{-i}$$

For rotation mode in a circular system these equation become:

$$x_{i+1} = x_i - y_i \cdot d_i \cdot 2^{-i}$$
$$y_{i+1} = y_i + x_i \cdot d_i \cdot 2^{-i}$$
$$z_{i+1} = z_i - d_i \cdot \tan^{-1} 2^{-i}$$

Where the decision function d_i depends on the sign of the current angle in the accumulator (d_i = -1 if $z_i < 0$, +1 otherwise).

The elementary angles (in case of the circular system: $\tan^{-1} 2^{-i}$) can be easily stored in a ROM. The three equations can be readily transformed into a hardware implementation.

Description of operation (bit-parallel design)

With reference to [1], a bit-parallel design has been implemented.

The width of the data path was 16 bits, with 1 sign bit, 7 integer bits and 8 fraction bits (actually, we could have done with less integer bits but that would not make structure more universal).

For the adder / subtractor, one of Altera’s megafunctions has been used as they map reasonably well into their FPGAs. Since we used this megafunction, no performance comparison between a CLA adder and RCA adder has been made.

The shifters have been implemented by considering the slice needed to be shifted at every iteration and then concatenating this with the number of MSB needed to be added to this slice.

The registers could also be implented as a megafunction. We decided to just build our own using a simple ‘flip flop’ process.

At first a lpm_rom was used to store the elementary rotation angles. But due to timing problems this gave us, we decided to simply use a with … select statement to retrieve the elementary angle we need for the z adder – subtractor. The elementary angles are given by $\tan^{-1} 2^{-i}$, where i is a counter that gives the value of the current iteration.

A state machine was used to increment that counter, and the amount of ‘shift’ needed in the x and y values at every iteration (the counter also gives the address needed for the rom)

Characteristics

Clock Speed

Measured clock speed:	9.30 MHz
Minimum clock period:	107.9 ns

Logic Cells

Total dedicated input pins used:	6/6	(100%)
Total I/O pins used:	174/183	(95%)
Total logic cells used:	517/1152	(44%)
Total embedded cells used:	0/48	(0%)
Total EABs used:	0/6	(0%)
Average fan-in:	3.38/4	(84%)
Total fan-in:	1750/4608	(37%)

References

[1] Andrake, R. J., "A survey of CORDIC algorithms for FPGA based computers", FPGA '98. Proceedings of the 1998 ACM/SIGDA sixth international symposium on Field programmable gate arrays, Feb. 22-24, 1998, Monterey, CA. pp191-200.