Computing the Sine Function

When implementing a sine calculator in digital hardware one has to remember the expense of the multiplication needed for many algebraical methods. Alternative techniques are based on polynomial approximation and/or table-lookup [4] as well as shift and add algorithms [5]. Among the various properties that are desireable, one can cite speed, accuracy or the reasonable amount of resource [1.1] [6]. The architecture of FPGAs specifies suitable techniques or might even change desireable properties. Because the number of sequential cells and amount of storage area, needed for table-lookup algorithms, are limited but combinational logic in terms of LUT in the FPGA's CLBs is sufficiently available, shift and add algorithms fit perfectly into an FPGA.

Subsections

Norbert Lindlbauer

2000-01-19

The CORDIC-Algorithm for Computing a Sine

In 1959 Jack E. Volder [7] described the COordinate Rotation DIgital Computer or CORDIC for the calculation of trigonometric functions, multiplication, division and conversion between binary and mixed radix number systems. The CORDIC-algorithm provides an iterative method of performing vector rotations by arbitrary angles using only shifts and adds. Volder's algorithm is derived from the general equations for vector rotation. If a vector V with components (x, y) is to be rotated through an angle ϕ a new vector V' with components (x', y') is formed by:

$$V' = \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \cdot \cos(\phi) - y \cdot \sin(\phi) \\ y \cdot \cos(\phi) + x \cdot \sin(\phi) \end{bmatrix} \tag{1.2}$$

Figure 1.2 illustrates the rotation of a vector $V=\left[\\ y\end{a by the angle ϕ.

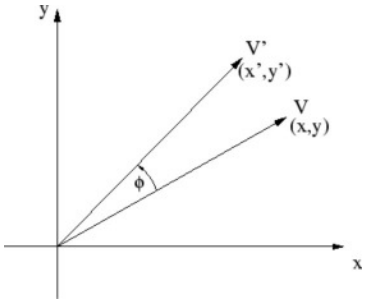

Figure 1.2: *Rotation of a vector V by the angle* ϕ

The individual equations for x' and y' can be rewritten as [8]:

$$x' = x \cdot \cos(\phi) - y \cdot \sin(\phi) \tag{1.3}$$

$$y' = y \cdot \cos(\phi) + x \cdot \sin(\phi) \tag{1.4}$$

and rearranged so that:

$$x' = \cos(\phi)\,[x - y \cdot \tan(\phi)] \tag{1.5}$$

$$y' = \cos(\phi)\,[y + x \cdot \tan(\phi)] \tag{1.6}$$

The multiplication by the tangent term can be avoided if the rotation angles and therefore $\tan(\phi)$ are restricted so that $\tan(\phi) = 2^{-i}$. In digital hardware this denotes a simple shift operation. Furthermore, if those rotations are performed iteratively and in both directions every value of $\tan(\phi)$ is representable. With $ \arbitrary constant \ the cosine term could also be simplified and since $\cos(\phi) = \cos(-\phi)$ it is a constant for a fixed number of iterations. This iterative rotation can now be expressed as:

$$x_{i+1} = K_i\,[x_i - y_i \cdot d_i \cdot 2^{-i}] \tag{1.7}$$

$$y_{i+1} = K_i\,[y_i + x_i \cdot d_i \cdot 2^{-i}] \tag{1.8}$$

where $K_i = \cos\left(\arctan\left(2^{-i}\right)\right)$ and $d_i = \pm 1$. The product of the K_i's represents the so-called K factor [9]:

$$K = \prod_{i=0}^{n-1} K_i\,. \tag{1.9}$$

This K factor can be calculated in advance and applied elsewhere in the system. A good way to implement the K factor is to initialize the iterative rotation with a vector of length $|K|$ which compensates the gain inherent in the CORDIC algorithm. The resulting vector V' is the unit vector as shown in Figure 1.3.

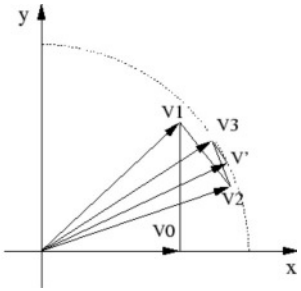

Figure 1.3: *Iterative vector rotation, initialized with V0*

Equations 1.7 and 1.8 can now be simplified to the basic CORDIC-equations:

$$x_{i+1} = \left[x_i - y_i \cdot d_i \cdot 2^{-i}\right] \tag{1.10}$$

$$y_{i+1} = \left[y_i + x_i \cdot d_i \cdot 2^{-i}\right] \tag{1.11}$$

The direction of each rotation is defined by d_i and the sequence of all d_i's determines the final vector. This yields to a third equation which acts like an angle accumulator and keeps track of the angle already rotated. Each vector V can be described by either the vector length and angle or by its coordinates x and y. Following this incident, the CORDIC algorithm knows two ways of determining the direction of rotation: the *rotation mode* and the *vectoring mode*. Both methods initialize the angle accumulator with the desired angle ϕ. The *rotation mode*, determines the right sequence as the angle accumulator approaches 0 while the *vectoring mode* minimizes the y component of the input vector[1,2].

The angle accumulator is defined by:

$$z_{i+1} = z_i - d_i \cdot \arctan\left(2^{-i}\right) \tag{1.12}$$

where the sum of an infinit number of iterative rotation angles equals the input angle ϕ [10]:

$$\phi = \sum_{i=0}^{\infty} d_i \cdot \arctan\left(2^{-i}\right) \tag{1.13}$$

Those values of $ \arctan(1) $ can be stored in a small lookup table or hardwired depending on the way of implementation. Since the decision is which direction to rotate instead of whether to rotate or not, d_i is sensitive to the sign of z_i. Therefore d_i can be described as:

$$d_i = \begin{cases} -1, & \text{if } z_i < 0 \\ +1, & \text{if } z_i \geq 0 \end{cases} . \tag{1.14}$$

With equation 1.14 the CORDIC algorithm in *rotation mode* is described completely. Note, that the CORDIC method as described performs rotations only within $-\pi/2$ and $\pi/2$. This limitation comes from the use of 2^0 for the tangent in the first iteration. However, since a sine wave is symmetric from quadrant to quadrant, every sine value from 0 to 2π can be represented by reflecting and/or inverting the first quadrant appropriately.

Next Up Previous

Next: Implementation of various CORDIC **Up:** Computing the Sine Function **Previous:** Computing the Sine Function
Home

Norbert Lindlbauer
2000-01-19

Introduction

In 1906, Thaddeus Cahill demonstrated a new instrument - the Telharmonium, the first and largest sound synthesizer ever developed [1]. Powered by electricity, but without the benefit of electronic amplification, the smoothly rotating tone generators of the Telharmonium emitted synthetic tones purer than nature - sinusoidal waves in the precise integer ratios of just intonation. Moved by the spectacle of this demonstration, the elderly American author Mark Twain (1835-1910) wrote: ``Every time I see or hear a new wonder like this I have to postpone my death right off. I couldn't possibly leave this world until I have heard it again and again!'' [2]. Many efforts on developing and improving sound synthesis followed with a greater or lesser degree of success. The invention of the stored program electronic digital computer in the 1940s finally opened the way for the present era of sound synthesis and since the first experiments of Max V. Mathews in 1957, multiple techniques of sound synthesis have been invented.

One of the most successful methods of sound synthesis is based on the summing of time-varying sinusoids, also known as *additive synthesis*. This method is described using the following equation:

$$y(t) = \sum_{i=0}^{N} A_i(t) \cos\left(2\pi f_i(t)\, t + \theta_i(t)\right). \tag{1.1}$$

The structural reconstruction can be carried out either in the time domain (see Figure 1.1) or the frequency domain.

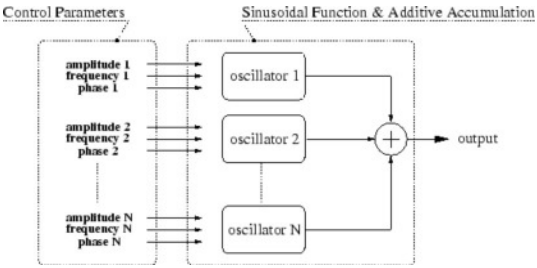

Figure 1.1: *Time-domain additive synthesis*

In either case the sinusoids are additively accumulated to create the output [3]. Theoretically any sound can be reproduced by proper control of the sinusoids over time. However, all the control parameters plus the sinusoidal function themselves must be calculated at the sampling rate - a quite demanding task even for todays CPUs. The load on the CPU can be reduced significantly by separating the control and

synthesis tasks, and assigning the task of computing the sinusoids to a co-processor, such as an FPGA. The following sections describe how to compute sinusoids and how to control the sinusoids attributes in such a device.

Next Up Previous

Next: Computing the Sine Function **Up:** FPGAs for Sound Synthesis **Previous:** FPGAs for Sound Synthesis
Home

Norbert Lindlbauer
2000-01-19

Implementation of various CORDIC Architectures

As intended by Jack E. Volder, the CORDIC algorithm only performs shift and add operations and is therefore easy to implement and resource-friendly. However, when implementing the CORDIC algorithm one can choose between various design methodologies and must balance circuit complexity with respect to performance. The most obvious methods of implementing a CORDIC, bit-serial, bit-parallel, unrolled and iterative, are described and compared in the following sections.

Subsections

Norbert Lindlbauer

2000-01-19

Center for New Music and Audio Technologies

Next Up Previous

Next: A Bit-Parallel Unrolled CORDIC **Up:** Implementation of various CORDIC **Previous:** Implementation of various CORDIC

A Bit-Parallel Iterative CORDIC

The CORDIC structure as discribed in equations 1.10, 1.11, 1.12 and 1.14 is represented by the schematics in Figure 1.4 when directly translated into hardware. Each branch consists of an adder-subtractor combination, a shift unit and a register for buffering the output. At the beginning of a calculation initial values are fed into the register by the multiplexer where the MSB of the stored value in the z-branch determines the operation mode for the adder-subtractor. Signals in the x and y branch pass the shift units and are then added to or subtracted from the unshifted signal in the opposite path.

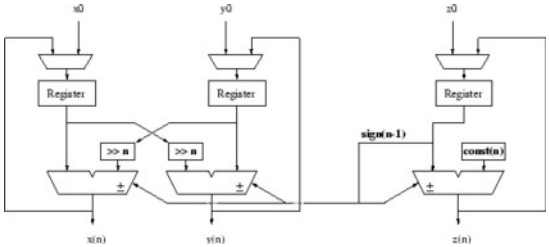

Figure 1.4: *Iterative CORDIC*

The z branch arithmetically combines the registers values with the values taken from a lookup table (LUT) whose address is changed accordingly to the number of iteration. For n iterations the output is mapped back to the registers before initial values are fed in again and the final sine value can be accessed at the output. A simple finite-state machine is needed to control the multiplexers, the shift distance and the addressing of the constant values.

When implemented in an FPGA the initial values for the vector coordinates as well as the constant values in the LUT can be hardwired in a word wide manner. The adder and the subtractor component are carried out separately and a multiplexer controlled by the sign of the angle accumulator distinguishes between addition and subtraction by routing the signals as required. The shift operations as implemented change the shift distance with the number of iterations but those require a high fan in and reduce the maximum speed for the application [11]. In addition the output rate is also limited by the fact that operations are performed iteratively and therefore the maximum output rate equals $1/n$ times the clock rate.

Next Up Previous

Next: A Bit-Parallel Unrolled CORDIC **Up:** Implementation of various CORDIC **Previous:** Implementation of various CORDIC
Home
Norbert Lindlbauer

2000-01-19

Center for New Music and Audio Technologies

Next Up Previous

Next: A Bit-Serial Iterative CORDIC **Up:** Implementation of various CORDIC **Previous:** A Bit-Parallel Iterative CORDIC

A Bit-Parallel Unrolled CORDIC

Instead of buffering the output of one iteration and using the same resources again, one could simply cascade the iterative CORDIC, which means rebuilding the basic CORDIC structure for each iteration. Consequently, the output of one stage is the input of the next one, as shown in Figure 1.5, and in the face of seperate stages two simplifications become possible. First, the shift operations for each step can be performed by wiring the connections between stages appropriately. Second, there is no need for changing constant values and those can therefore be hardwired as well.

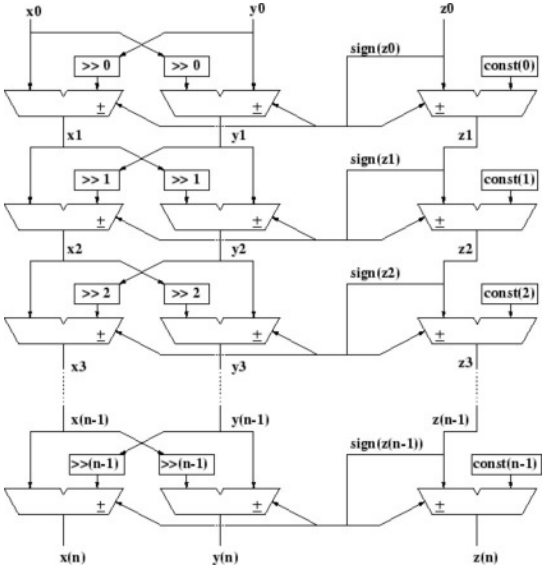

Figure 1.5: *Unrolled CORDIC*

The purely unrolled design only consists of combinatorial components and computes one sine value per clock cycle. Input values find their path through the architecture on their own and do not need to be controlled.

Obviously the resources in an FPGA are not very suitable for this kind of architecture. As we talk about a bit-parallel unrolled design with 16 bit wordlength, each stage contains 48 in- and outputs plus a great number of cross-connections between single stages. Those cross-connections from the x-path through the shift components to the y-path and vice versa make the design difficult to route in an FPGA and cause additional delay times. From table 1.1 it can be seen how performance and resource usage change with the number of iterations if implemented in an XILINX FPGA *XC4010E*. Naturally, the area and therefore the maximum path delay increase as stages are added to the design where the path delay is an equivalent to the speed which the application could run at.

Table 1.1: *Performance and CLB usage in an XC4010E*

No. of Iterations	8	9	10	11	12	13
complexity [CLB]	184	208	232	256	280	304
max path delay[ns]	163.75	177.17	206.9	225.72	263.86	256.87

As described earlier, the area in FPGAs can be measured in CLBs, each of which consist of two lookup tables as well as storage cells with additional control components [12]. For the purely combinatorial design the CLB's function generators perform the add and shift operations and no storage cells are used. This means registers could be inserted easily without significantly increasing the area. Pipelining adds some latency, of course, but the application needs to output values at 48kHz and the latency for 14 iterations equals 312.5 μs which is known to be inperceptible. However, inserting registers between stages would also reduce the maximum path delays and correspondingly a higher maximum speed can be achieved. Table 1.2 shows how the area versus speed trade off is affected by different pipelining methods.

Table 1.2: *Performance and CLB usage for various methods of pipelining in an XC4010E*

No. of Iterations between Registers	1	2	3	4	8	13
Complexity [CLB]	313	308	304	304	304	304
max. Frequency [MHz]	24.4	18.3	14.2	9.7	6.2	3.7

The values are taken from report files generated by the XILINX Foundation Series software when implementing the unrolled designs. It can be seen, that the number of CLBs stays almost the same while the maximum frequency increases as registers are inserted. The reason for that is the decreasing amount of combinatorial logic between sequentiell cells. Obviously, the gain of speed when inserting registers exceeds the cost of area and makes therefore the fully pipelined CORDIC a suitable solution for generating a sinewave in FPGAs. Especially if a sufficient number of CLBs is at one's disposal, as is the case in high density devices like XILINX's Virtex or ALTERA's FLEX families, this type of architecture becomes more and more attractive.

Next Up Previous

Next: A Bit-Serial Iterative CORDIC **Up:** Implementation of various CORDIC **Previous:** A Bit-Parallel Iterative CORDIC

Home

Norbert Lindlbauer
2000-01-19

Center for New Music and Audio Technologies

Next Up Previous

Next: Comparison of the Various **Up:** Implementation of various CORDIC **Previous:** A Bit-Parallel Unrolled CORDIC

A Bit-Serial Iterative CORDIC

Both, the unrolled and the iterative bit-parallel designs, show disadvantages in terms of complexity and path delays going along with the large number of cross connections between single stages. To reduce this complexity one could change the design into a completely bit-serial iterative architecture. Bit-serial means only one bit is processed at a time and hence the cross connections become one bit-wide data paths. Clearly, the throughput becomes a function of

$$\frac{clock\ rate}{number\ of\ iterations \times word\ width}.$$

In spite of this the output rate can be almost as high as achieved with the unrolled design. The reason is the stuctural simplicity of a bit-serial design and the correspondingly high clock rate achievable. Figure 1.6 shows the basic architecture of the bitserial CORDIC processor as implemented in a XILINX Spartan.

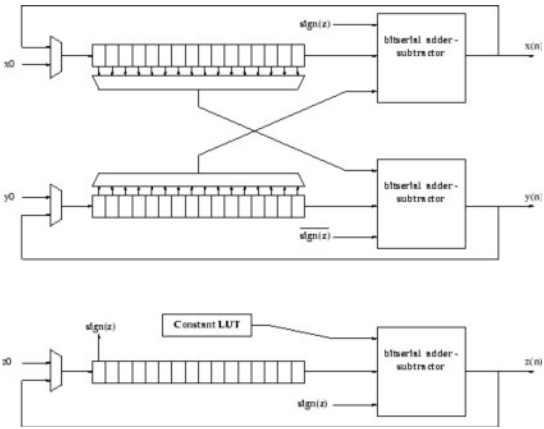

Figure 1.6: *Bit-serial CORDIC*

In this architecture the bit-serial adder-subtractor component is implemented as a fulladder where the subtraction is performed by adding the 2's complement of the actual subtrahent [13]. The subtraction is again indicated by the sign bit of the angle accumulator as described in section 1.2.1. A single bit of state is stored at the adder to realize the carry chain [14] which at the same time requires the LSB to be fed in

first. The shift-by-i operation can be realized by reading the bit i-1 from it's right end in the serial shift registers. A multiplexer can be used to change position according to the current iteration. The initial values $x0$, $y0$ and $z0$ are fed into the array at the left end of the serial-in - serial-out register and as the data enters the adder component the multiplexer at the input switch and map back the results of the bit-serial adder into the registers. The constant LUT for this design is implemented as a multiplexer with hardwired choices. Finally, when all iterations are passed the input multiplexers switch again and initial values enter the bit-serial CORDIC processor as the computed sine values exit.

The design as implemented runs at a much higher speed than the bit-parallel architectures described earlier and fits easily in a XILINX SPARTAN device. The reason is the high ratio of sequential components to combinatorial components. The performance is constrained by the use of multiplexers for the shift operation and even more for the constant LUT. The latter could be replaced by a RAM or serial ROM where values are read by simply incrementing the memory's address. This would clearly accelerate the performance but since optimization for one particular FPGA device falls outside the slope of this paper, we will not consider it further.

Next Up Previous

Next: Comparison of the Various **Up:** Implementation of various CORDIC **Previous:** A Bit-Parallel Unrolled CORDIC
Home

Norbert Lindlbauer
2000-01-19

Center for New Music and Audio Technologies

Next Up Previous

Next: Controlling the Oscillator **Up:** Implementation of various CORDIC **Previous:** A Bit-Serial Iterative CORDIC

Comparison of the Various CORDIC Architectures

In the previous sections, we described various methods of implementing the CORDIC algorithm using an FPGA. The resulting structures show differences in the way of using resources available in the target FPGA device. Table 1.3 illustrates how the architectures for the iterative bit-serial and iterative bit-parallel designs for 16 bit resolution vary in terms of speed and area. The bit-serial design stands out due to it's low area usage and high achievable speed. Whereas the latency and hence the maximum throughput rate is much lower compared to the bit-parallel designs. The bit-parallel unrolled and fully pipelined design (see Table 1.2) uses the resources extensively but shows the best latency per sample and maximum throughput rate. The prototyping environment limited the implementation of the unrolled design to 13 iterations. The iterative bit-parallel design provides a balance between unrolled and bit-serial design and shows an optimum usage of the resources in a XILINX *XC4010E*.
In actual fact it would be more accurate to look at the resources available in the specific target devices rather than the specific needs in order to determine what architecture to use. The bit-serial structure is definitely the best choice for relatively small devices, but for FPGAs where sufficient CLBs are available one might choose the bit-parallel and fully pipelined architecture since latency is minimal and no control unit is needed.

Table 1.3: *Performance and CLB usage for the bit-parallel and bit-serial iterative designs.*

	CLB	LUT	FF	Speed	Latency	max. Throughput
	[1]	[1]	[1]	[MHz]	[μs]	[Mio. Samples $\cdot s^{-1}$]
bit-serial	111	153	108	48	5.33	0.1875
bit-parallel	138	252	52	36	0.44	2.25

Next Up Previous

Next: Controlling the Oscillator **Up:** Implementation of various CORDIC **Previous:** A Bit-Serial Iterative CORDIC
Home

Norbert Lindlbauer
2000-01-19

Controlling the Oscillator

An important requirement of digital additive synthesis is the accurate and independent mapping of the control parameters, *amplitude, frequency*, and *phase* into sinusoids as described earlier in equation 1.1:

$$y(t) = \sum_{i=0}^{N} A_i(t) \cos\left(2\pi f_i(t)\, t + \theta_i(t)\right).$$

The *amplitude* A_i represents the maximum displacement of the varying quantity from its average value. The *frequency* f_i gives the mumber of peaks per second in a sine wave whereas the reciprocal of f_i, the *period* T_i, is the time between amplitude peaks. Finally, the *phase* θ_i, describes the displacement in time from its origin [15]. The required resolutions for *frequency* f_i and *phase* ϕ_i can be estimated from the *just noticeable difference (JND)* curve derived for the human ear [16]. The curve shows that the human ear is capable for noticing pitch differences of □ percent at 50Hz which corresponds to a resolution of 0.5 Hz. With a sampling rate of 48kHz the resolution needs to be 16 bit for *frequency* f_i and *phase* ϕ_i.

Amplitude resolution is dictated largely by what is considered acceptible in audio terms, and the resources that are available. With this in mind, 16 bits was chosen for the amplitude parameter.

Subsections

Norbert Lindlbauer
2000-01-19

Frequency

The CORDIC algorithm returns sine values for a given input angle. In order to generate a sine wave a sequence of input angles corresponding the rotation of a unary vector within the angle range of the CORDIC algorithm, e.g., $-\pi/2$ to $\pi/2$ is required. Two such sequences of input angles are depicted in Figure 1.7.a.

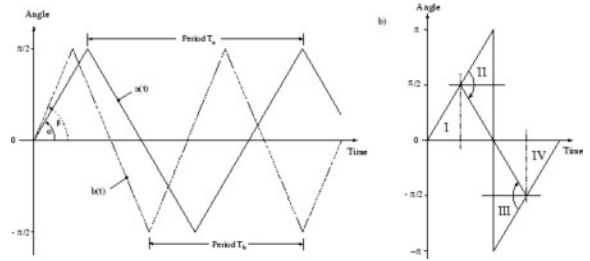

Figure 1.7: *a) Sequence of input angles. b) Triangular wave derived from a sawtooth.*

The gradient of the side of the triangle determines the period or frequency of the triangular wave and therefore of the resulting sinusoid.
A triangle wave can be derived from a sawtooth (see Figure 1.7.b), which is easily implementd using an accumulator. However, the direct translation of an interval $-\pi/2$ to $\pi/2$ into digital structure is an involved undertaking. This is because π is not a multiple power of 2 - essential for the use of an accumulator. In this place a feature inherent in the CORDIC algorithm presents a convenient solution. As stated in equation 1.13, the infinite sum of signed values in the CORDIC lookup table equals the input angle. Consequently, if the range for the input angles is converted into a more suitable interval, such as -2^{n-1} to $2^{n-1}-1$, a scaling factor ξ can be described:

$$\xi = \frac{2^n - 1}{\pi} = \frac{\phi'}{\phi} \qquad \text{for all } n \in \mathrm{N} \tag{1.15}$$

with $\phi \in [-\pi/2; \pi/2]$ and $\phi' \in \left[-2^{n-1}; 2^{n-1}-1\right]$. This factor applied to equation 1.13 gives:

$$\phi' = \xi \cdot \sum_{i=0}^{\infty} d_i \cdot \arctan\left(2^{-i}\right) \tag{1.16}$$

We then gather the rescaled values for use in the CORDIC lookup table. This facilitates the input angle to be of an interval -2^{n-1} to $2^{n-1}-1$, and hence using an accumulator for generating the sequence of input angles.

Subsections

Next Up Previous

Next: Implementation **Up:** Controlling the Oscillator **Previous:** Controlling the Oscillator
Home

Norbert Lindlbauer

2000-01-19

Center for New Music and Audio Technologies

Next Up Previous

Next: Phase **Up:** Frequency **Previous:** Frequency

Implementation

A triangle wave can be generated by mirroring parts of a sawtooth waveform. As mentioned earlier, a sawtooth is easily generated using an accumulator, consisting of an adder and a register. The register is incremented by the input value in synchrony with the sample clock. As a result the input determines the rate at which the angle increases, and hence the overall oscillator frequency. Periodically the register will overflow, signifying the end of a complete cycle. In order to derive a triangle wave from the sawtooth we divide the sawtooth curve into four parts (see Figure 1.7.b) representing the four quadrants of a rotary system. It can be seen that the upper half of the sawtooth must be inverted. This is within quadrant II and III. The two MSBs of the accumulator can be used to determin the present position within the four quadrants, and therefore to control the inversion.

```
10 process(sawtooth)
20 begin
30     case sawtooth(16 downto 15) is
40         when "00"—"11" => triangle <= sawtooth(15 downto 0);
50         when "01"—"10" => triangle <= not sawtooth(15 downto 0);
60         when others => null;
70     end case;
80 end process;
```

Figure 1.8: *VHDL code for inverting a sawtooth into a triangular wave.*

The VHDL code in Figure 1.8 illustrates the inversion of the sawtooth. The mapping into hardware represents a multiplexer between the original and the inverted signal depending on the present position within the quadrants. The resulting circuit is shown in Figure 1.9 as part of the complete design.

Next Up Previous

Next: Phase **Up:** Frequency **Previous:** Frequency
Home

Norbert Lindlbauer
2000-01-19

Next Up Previous
Next: Amplitude **Up:** Controlling the Oscillator **Previous:** Implementation

Phase

The *phase* provides a means to displace the positions of two waveforms in time relative to each other [17]. This equals an offset added to the sawtooth, easily implemented by inserting an adder at the output of the accumulator (see Figure 1.9).

Home
Norbert Lindlbauer
2000-01-19

Amplitude

Each sinusoid needs to be weighted appropriately to its magnitude within the spectrum. The normal way to do this is to use a multiplier controlled by the parameter *amplitude*. At the beginning of this chapter we mentioned the expense of multiplication in digital hardware and found a way around by using CORDIC for computing the sine function. At this point a simple replacement of multiplication by the CORDIC algorithm would be possible but is not necessarily a gain of resources. However, as described by equation 1.2.1 the CORDIC algorithm contains a gain due to the K-factor. We compensated this gain by initializing the rotation with a vector of length $|K|$, ($) respectively, so that the final vector was the unary vector. Hence, initializing with different $|K|$'s would result in a vector with length which is not one. This leads to the idea of initializing the rotation with the control parameter for *amplitude* and performing multiplication and computation of sine using the same resources. Thus, it is to be proved that the gain inherent in the CORDIC algorithm is linear and indepent of varying conditions, e.g., input angle, sequence of directions, initial values.

Subsections

Norbert Lindlbauer
2000-01-19

Center for New Music and Audio Technologies

Next Up Previous

Next: Implementation **Up:** Amplitude **Previous:** Amplitude

Proof

The sequence of *directions* $\vec{d}$ is completely defined in Equation 1.12

$$z_{i+1} = z_i - d_i \cdot \arctan\left(2^{-i}\right)$$

and Equation 1.14:

$$d_i = \begin{cases} -1, & \text{if } z_i < 0 \\ +1, & \text{if } z_i \geq 0 \end{cases}.$$

$$(x_n, y_n) = \left(K\left(x_0 \cdot P(d) + y_0 \cdot Q(d)\right), K\left(y_0 \cdot P(d) - x_0 \cdot Q(d)\right)\right) \tag{1.17}$$

be the general term of the iterative system determined by Equation 1.10 and 1.11, where $P(d)$ and $Q(d)$ are polynomials in the variables d.
Setting the initial vector V_0 to $(x_0, 0)$ we get:

$$(x_n, y_n) = (K \cdot x_0 \cdot P(d), -K \cdot x_0 \cdot Q(d)) \tag{1.18}$$

$$= K x_0 \left(P(d), -Q(d)\right) \tag{1.19}$$

Now define a new initial Vector $V_0' = (\lambda x_0, 0)$:

$$(x_n', y_n') = K \lambda x_0 \left(P(d), -Q(d)\right) \tag{1.20}$$

$$= \lambda (x_n, y_n) \tag{1.21}$$

From Equation 1.21 it follows that the gain inherent in the CORDIC algorithm is a constant for a constant number of iterations and independent of the initial values if coordinate $y_0 = 0$. Consequently,

multiplication can be performed by initializing the rotation with the control parameter *amplitude*. This eliminates the need for an additional multiplier and reduces the design complexity enormously.

Next Up Previous

Next: Implementation **Up:** Amplitude **Previous:** Amplitude
Home

Norbert Lindlbauer
2000-01-19

Center for New Music and Audio Technologies

Next Up Previous
Next: Results **Up:** Amplitude **Previous:** Proof

Implementation

As demonstrated the amplitude control can be carried out within the CORDIC structure. Instead of hard-wiring the initial values as proposed in section 1.2.2, the values are now fed into the CORDIC structure through a separate input. Figure 1.9 illustrates the resulting structure of the complete oscillator.

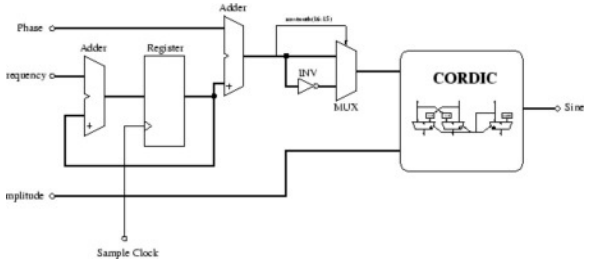

Figure 1.9: *A CORDIC-based Oscillator*

Home
Norbert Lindlbauer
2000-01-19

Results

The oscillator described in the previous sections has been implemented and tested in a XILINX *XC4010E*. The architecture of this device provides specific resources in terms of CLBs, LUTs, storage cells or maximum speed. The results obtained when implementing the oscillator are presented in the following sections.

Subsections

Norbert Lindlbauer

2000-01-19

Area Usage and Performance

The various architectures of the implemented CORDIC algorithm use resources in the target FPGA device differently. We chose the bit-parallel iterative CORDIC architecture to implement a complete oscillator using an FPGA since this structure shows a trade off between area usage and maximum achievable speed. Table 1.4 illustrates the exact results obtained using a XILINX *XC4010E* as the target device. The values are taken from report files generated by the FPGA implementation software and provide detailed information about area usage and timing issues.

Table 1.4: *Performance and CLB usage for the bit-parallel designs.*

	CLB	LUT	FF	Speed	Latency	max. Throughput
	[1]	[1]	[1]	[MHz]	[μs]	[Mio. Samples $\cdot s^{-1}$]
Control structure	26	49	17	61.2	0.03	30.6
Cordic	138	252	52	36	0.44	2.25
Oscillator	153	267	69	28	0.64	1.55

Norbert Lindlbauer
2000-01-19

Error

The use of CORDIC for computing sine introduces precision errors because of the limited number of iterations and the limited resolution of values in the CORDIC lookup table. The error results in distortion of the generated sine wave. Figure 1.10 illustrates the magnitude error of a CORDIC-based sine wave with a frequency of 375 Hz when compared to a reference sine wave. Both are normalized and sampled at 48 kHz. The error results in high frequency added to the 375 Hz sine wave, as represented by the spectrum in Figure 1.11.

The values are taken from data produced by the logic simulation within the XILINX Foundation software and are analyzed using a FFT[1,3] with Hamming window in MATLAB.

The energy of the error obtained can be also expressed by the equation:

$$\rho_{RMS} = \sqrt{\frac{1}{N} \cdot \sum_{n=0}^{N} \rho^2} = 5.8578 \cdot 10^{-5} \tag{1.22}$$

where ρ is the magnitude error. With S_{RMS} is the root-mean-square amplitude of the reference signal we get for the signal-to-noise ratio:

$$\mathrm{SNR} = 20 \log_{10} \left(\frac{S_{RMS}}{\rho_{RMS}} \right) = 81.6\mathrm{dB} \tag{1.23}$$

In the context of additive synthesis, where multiple sinusoids are added together, the SNR obtained here is an acceptable result.

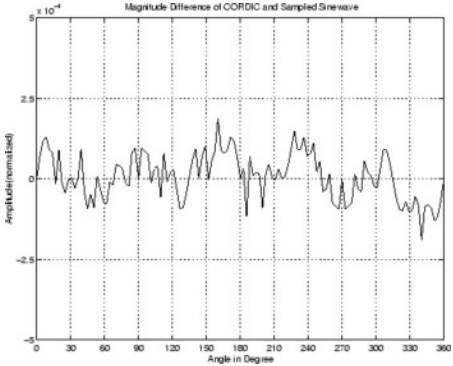

Figure 1.10: *Magnitude error of a sine wave generated by a CORDIC-based oscillator with 16 bit resolution.*

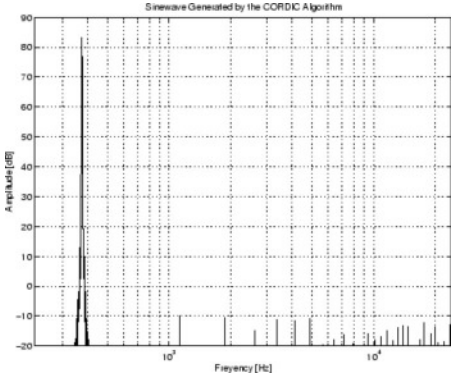

Figure 1.11: *Spectrum of a sine wave generated by a CORDIC-based oscillator with 16 bit resolution.*

Next Up Previous

Next: Conclusion **Up:** Results **Previous:** Area Usage and Performance
Home

Norbert Lindlbauer
2000-01-19

Conclusion

FPGAs provide an ideal environment for designing and emulating digital functions. The design *CORDIC-based Oscillator* had been successfully implemented and tested under real, performance like conditions.

The CORDIC algorithm is an iterative method to perform rotation, and therefore to compute sine values. This makes the CORDIC not the fastest method for calculating a sine function. However, the possibility of combining multiplication and sine computation in CORDIC makes the CORDIC-based oscillator an attractive solution since it saves the resources needed by a multiplier.

Clearly, many oscillators are necessary to perform additive synthesis. For 32-note polyphony (or timbre), with 64 partial components per note, a engine will need to generate 2,048 sine waves in total. This clearly exceeds the capacity of the prototyping environment used but high density devices provide sufficient resources for an additive synthesizer. Steadily increasing frequency rates for modern chip technologies, which make computing a desired value faster than accessing a lookup table, add another interesting aspect to the decision of what oscillator architecture to use.

Home
Norbert Lindlbauer
2000-01-19

Bibliography

1

Roads C.
The Computer Music Tutorial.
MIT Press, 2nd edition, 1995.

2

Rhea T.
The Evoluton of electronic musical instruments in the United States.
PhD thesis, Nashville: George Peabody College for Teachers, 1972.

3

Michal Goodwin.
Frequency-domain analysis-synthesis of musical sounds.
Master's thesis, CNMAT and Department of Electrical Engineering and Computer Science, UCB, 1994.

4

Muller J. M.
Elementary Functions - Algorithms and Implementation, chapter 3, page 19.
Birkhäuser, 1997.

5

Muller J. M.
Elementary Functions - Algorithms and Implementation, chapter 5, page 69.
Birkhäuser, 1997.

6

Muller J. M.
Elementary Functions - Algorithms and Implementation, pages 1,2.
Birkhäuser, 1997.

7

Volder J. E.
The CORDIC trigonometric computing technique.
In *IRE Trans. Electronic Computing*, volume EC-8, pages 330 - 334, 1959.

8

Considine V.
CORDIC trigonometric function generator for DSP.
In *IEEE-89, International Conference on Acoustics, Speech and Signal Processing*, pages 2381 - 2384, Glasgow, Scottland, May 1989.

9

Walther J.S.
A unified algorithm for elementary functions.

In *Spring Joint Computer Conference*, pages 379 - 385, 1971.

10

Muller J. M.
Elementary Functions - Algorithms and Implementation, chapter 6, page 102.
Birkhäuser, 1997.

11

Andraka R.
A survey of cordic algorithms for fpga based computers.
In *Proceedings of the 1998 ACM/SIGDA sixth international symposium on Field Programmable Gate Arrays*, pages 191-200, Monterey, CA, Feb.22-24 1998.

12

XILINX, Inc.
XC *4000E* and XC *4000X* series programmable gate arrays - product specification.
www.xilinx.com/partinfo/#4000.pdf, January 1999.

13

Andraka R.
Building a high performance bit serial processor in an fpga.
In *On-Chip System Design Conference*, Jan. 1996.

14

Warzynek J.
A bit serial cordic architecture.
California Institute of Technology, 1982.

15

Pierce R. J.
The Science of Musical Sound, chapter III, pages 38-39.
Freeman, 1983.

16

Evans E. F.
Basic Physics and Psychophysics of Sound, Functional Anatomy of the Auditory System.
In H. B. Barlow and J. D. Mollon - The Senses. Cambridge University Press, Cambridge, UK, 1982.

17

Jerse T. A. Dodge C.
Computer Music - Synthesis, Composition, and Performance, chapter 4. Synthesis Fundamentals, pages 75-76.
Schirmer Books, 1997.

Home
Norbert Lindlbauer
2000-01-19