

# Implementing and Optimizing a Direct Digital Frequency Synthesizer (DDFS) on FPGA

Jung-Seob LEE

Xiangning Yang

May 10, 2006

# 1 Introduction

In the past, the conventional phase-locked loop (PLL) synthesizer is used as frequency synthesizer. The fundamental advantage of PLL's has been their ability to synthesize an output sine wave of high spectral purity which may be tuned over a wide bandwidth. However, modern communication systems are placing ever-increasing demands on the resolution, bandwidth, and switching speed of frequency synthesizers.

Due to its fast frequency switching capability, continuous phase, fine frequency resolution, and good spectral purity, the direct digital frequency synthesizer (DDFS) plays a crucial role in frequency spread-spectrum communications systems as well as phase shift-keying modulation and software radio system.

After survey and comparison with proposed DDFS algorithms ([7][6][3][11][10] [12][4][8] [9]), we chose the algorithm presented in [7] to be implemented in our project. The selected algorithm has some advantages such as the highest clock frequency and spurious free dynamic range (SFDR) while requires a modest implementation complexity.

We implemented this algorithm on FPGA using the algorithm transformation and mapping approach. Our project focus is to optimize the introduced hardware design with cut-set re-timing implementation while maintaining the original precision

## 2 Prior Arts

Most, if not all DDFS implementation techniques will follow the general architecture shown in Figure 1. The phase accumulator generate an angle phase periodically. Then the sin/cos generator output the digital sin/cos value of that angle. Lastly DAC and LPF convert the signal into a smooth analog waveform. The central design problem will be how to generate the sin/cos values in a fast, precise and efficient way. Generally, there are 2 methodologies.

First one is to pre-compute all the sin/cos values and then store them in the LUT, and runtime the DDFS just read the LUT and output the value periodically. This methodology will require less sophisticated design and can achieve high speed and waveform purity. The main

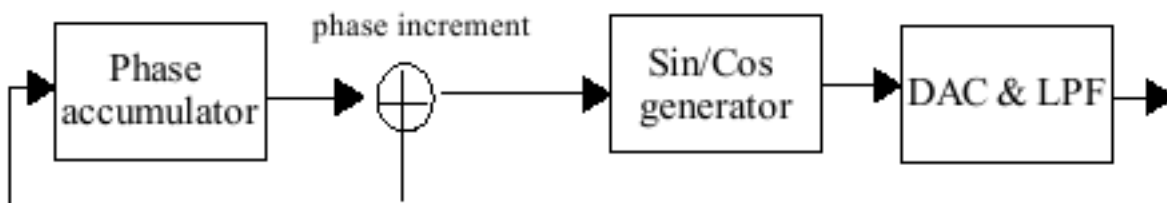


Figure 1: General DDS Architecture

disadvantage is to require a huge LUT (Look up Table), which size will grow exponentially as the accuracy of the phase value increases linearly. Various compression techniques are proposed to reduce the table size (such as [9]).

Another type of DDS is to directly generate the sin/cosine value based on angle rotation scheme. The CORDIC [1] algorithm is a widely used angle rotation techniques. In addition, lots of interpolative angle rotation techniques (such as [10]) are also proposed. Compared with the LUT table technique, the angle rotation method can greatly reduced the area requirement although it increases the design complexity. However, as it is pointed out in [5], generally a wide internal data-path is needed to obtain desired output precision. For example, to get 15-bit output precision, 19-bit internal data-path is needed.

Besides, many hybrid techniques have also proposed to try to give a good engineering design point. Generally, those algorithm will first try to get a coarse position of the given angle, then fine-tune the angle with angle rotation method (such as CORDIC) to get a close approximation. Those techniques aim to get a good trade off between internal data-path, latency and area by combining two major methodologies.

The algorithm used in our project is a hybrid one, as it is described in following section.

### 3 The Hybrid Algorithm

This section describes the DDS algorithm we used in our project. The algorithm is purposed in [7]. Figure 2 shows the basic concept of calculating the sin/cos value with angle rotation method. Given a angle  $\theta$ , if the radius  $r = 1$ , the X coordinate will be the cos value while the

Y coordinate will be the sin value.

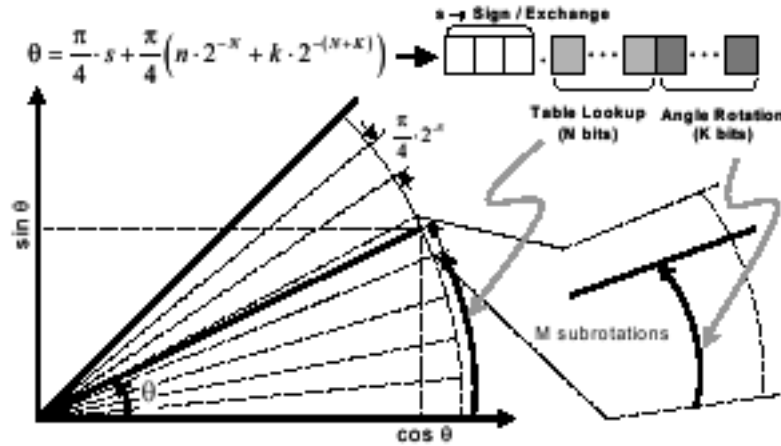


Figure 2: Basic Concept of The Hybrid Algorithm

An angle  $\theta$  can be decomposed into three parts as:

$$\theta = \frac{\pi}{4} \cdot s + \frac{\pi}{4} \left( n \cdot 2^{-N} + k \cdot 2^{-(N+K)} \right)$$

The first part  $s$  is of 3-bit wide. It represents the  $\pi/4$  symmetry of trigonometric functions (e.g.  $\cos(\pi/4 - \theta) = \sin(\theta)$ ,  $\sin(\pi - \theta) = \theta$ , etc.). Hence, only the  $X, Y$  values in  $[0, \pi/4)$  range are needed to be calculated, while for other parts, the  $\cos$  and  $\sin$  values can be obtained by exchanging  $X$  and  $Y$  values (in  $[0, \pi/4)$ ) and applying minus sign probably. The corresponding manipulation are shown in Table 1

The second  $N$ -part is the coarse phase.  $\cos$  and  $\sin$  values of the coarse phase are stored in two loop-up table (LUT). Thus each LUT will contain  $2^N$  entries and the value resolution is set to  $L$  bits. The coarse phase value are the start point of fine-grain angle rotation. The LUT size increases exponentially with  $N$  and linearly with  $L$ . Thus, one should control  $N$  to be small so that size of LUT will not be too large.

Finally, an angle rotation method will be applied to fine turn the position of the angle. Since the LUTs give  $\cos$  and  $\sin$  values of angle  $n \cdot 2^{-N}$ . The fine-grained rotation should further rotate the angle by  $\theta_k = \pi/4 \cdot k \cdot 2^{-(N+K)}$ , where  $k$  is an integer with  $K$ -bit wide. From the CORDIC algorithm, we know that  $\theta_k$  can be calculated by:

$$\theta_k = \sum_i \sigma_i \cdot \arctan(2^{-i})$$

Region	s[2]	s[1]	s[0]	Phase	Sine Output	Cosine Output
$[0, \pi/4)$	0	0	0	$\theta$	Y	X
$[\pi/4, \pi/2)$	0	0	1	$-\theta$	X	Y
$[\pi/2, 3\pi/4)$	0	1	0	$\theta$	X	-Y
$[3\pi/4, \pi/)$	0	1	1	$-\theta$	Y	-X
$[\pi, 5\pi/4)$	1	0	0	$\theta$	-Y	-X
$[5\pi/4, 3\pi/2)$	1	0	1	$-\theta$	-X	-Y
$[3\pi/2, 7\pi/4)$	1	1	0	$\theta$	-X	Y
$[7\pi/4, 2\pi)$	1	1	1	$-\theta$	-X	X

Table 1: Output Exchange and the Sign

Where  $\sigma_i \in \{-1, 1\}$ . Since  $\pi/4 < 1$ , it is easy to see that  $\theta_k = \pi/4 \cdot k \cdot 2^{-(N+K)}$  must contain at least  $N$  leading zeros (when it is presented in binary fix-point format). That is,  $\theta_k$  can be represented as:

$$\theta_k = 0.00 \cdots x_{N+1}x_{N+2}x_{N+3} \cdots$$

Further more, we have  $\arctan(2^{-i}) = 2^{-i} - 1^{-3i}/2 + \cdots$ . Thus, the  $\sigma_i$  prediction scheme purposed in [2] can be applied. That is,  $\sigma_i = 1$  for  $i = N+1$ , and  $\sigma_i = 2x_{i-1} - 1$  for  $i \in [N+2, 3N+4)$ . Thus, if we maintain  $K < 2N$ , we can directly use prediction scheme to get  $\sigma_i$ .

Now, suppose the LUTs' outputs are  $X_{n,N}$  and  $Y_{n,N}$  respectively, we have:

$$X_{n,i+1} = X_{n,i} - \sigma_{i+1}2^{-i-1}Y_{n,i}$$

$$Y_{n,i+1} = Y_{n,i} + \sigma_{i+1}2^{-i-1}X_{n,i}$$

Suppose, we want to rotate  $M$  times to get the position  $(X_{n,N+M}, Y_{n,N+M})$ , based on the approximation purposed on [8] and [7].

$$X_{n,N+M} \cong X_{n,N} - Y_{n,N} \sum_{i=N+1}^{N+M} \sigma_i 2^{-i} - 2^{-N-1} \cdot X_{n,N} \sum_{i=N+1}^{N+M} \sigma_i 2^{-i}$$

$$Y_{n,N+M} \cong Y_{n,N} + X_{n,N} \sum_{i=N+1}^{N+M} \sigma_i 2^{-i} - 2^{-N-1} \cdot Y_{n,N} \sum_{i=N+1}^{N+M} \sigma_i 2^{-i}$$

Now, replace  $\sigma_i$  with  $2x_{i-1} - 1$  as it is described above, we can make further approximation

as:

$$\begin{aligned}
\sum_{i=N+1}^{N+M} \sigma_i 2^{-i} &= 2^{-N-1} + \sum_{i=N+1}^{N+M} (2x_{i-1} - 1) 2^{-i} \\
&\cong \sum_{i=N+1}^{N+M-1} x_i 2^{-i} \\
&= \theta_k
\end{aligned}$$

This means that the formulas to obtain the final position (after  $M$  sub-rotations) can be reduced to:

$$\begin{aligned}
X &= X_{n,N+M} = X_{n,N} - (Y_{n,N} \theta_k + X_{n,N} \cdot 2^{-N-1} \theta_k) \\
Y &= Y_{n,N+M} = Y_{n,N} + (X_{n,N} \theta_k - Y_{n,N} \cdot 2^{-N-1} \theta_k)
\end{aligned}$$

$X_{n,N}$  and  $Y_{n,N}$  are the outputs from the LUTs. To get the final  $X$  and  $Y$  position, only two multiplication, two additions and two subtractions are needed after the accessing the LUTs.

As the simulation results shown in [7], when we choose  $N = 7$ ,  $K = 9$  (hence the angle phase is  $S + N + K = 19$  bits) and  $L = 16$  (hence the internal data part is 16-bit), the output precision can achieve 15 bits precision (in magnitude, with the sign bit, 16 bits totally).

## 4 Simulation Study

To study the characteristics of the algorithm, we first implement it in C language. The program uses fix-point arithmetic rather than floating point to fully simulate the algorithm. Figure 3 shows the plotting of the sine/cosine waveforms generated by the simulator. We can see that shape of the waveforms are closed to what is expected.

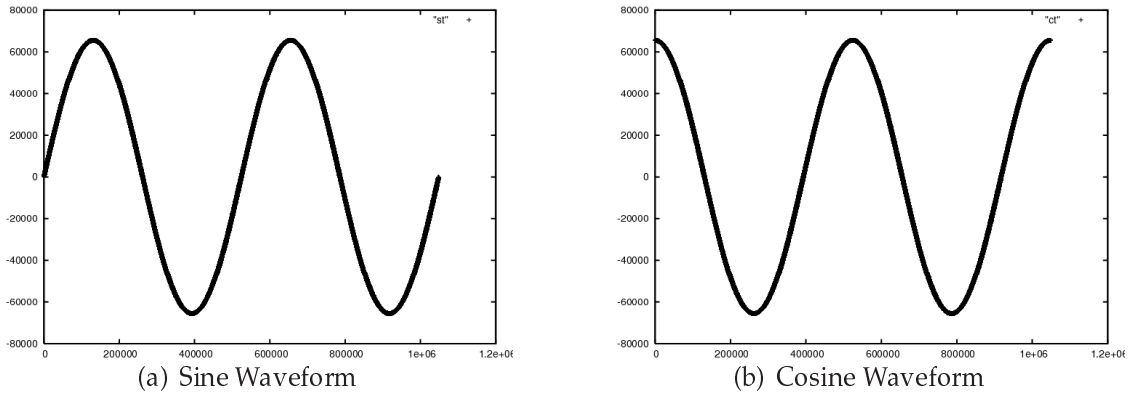


Figure 3: Output From the Simulator

## 5 FPGA Implementation

The algorithm is first mapped to the FPGA with un-pipelined style. Figure 4 shows the block diagram.

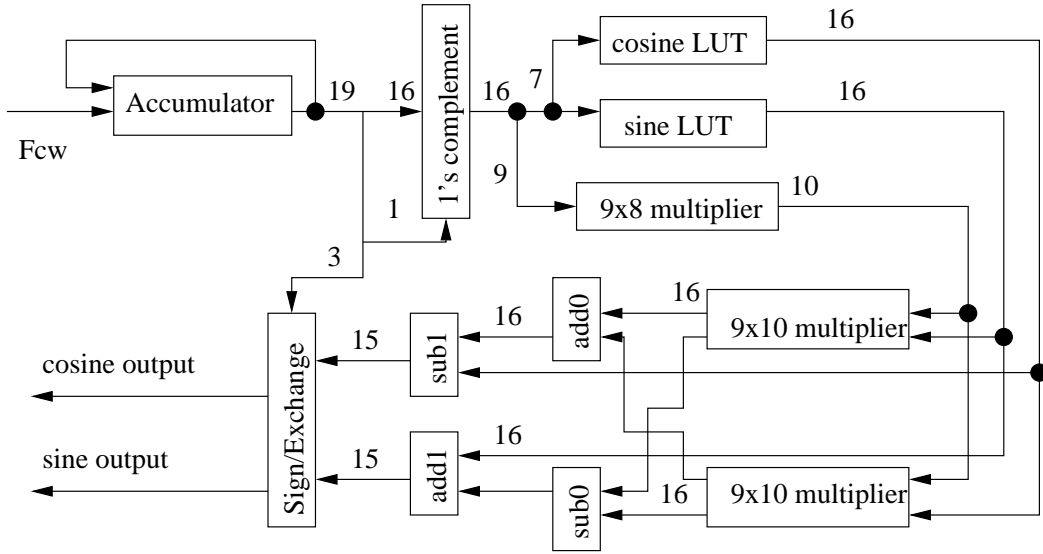


Figure 4: Block Diagram of Un-pipelined Implementation

The accumulator generates the desired angle phase to be calculated. Please be noted as we represent the angle as:

$$\theta = \frac{\pi}{4} \dots + \frac{\pi}{4} \left( n \cdot 2^{-N} + k \cdot 2^{-(N+K)} \right)$$

The accumulator generates the value of  $s$ ,  $n$  and  $k$  as a binary number. As the we know  $S = 3$ ,  $N = 7$  and  $K = 9$ , the topmost 3-bit of accumulator's output will be the  $s$ , the next 7 bits will be the  $n$  while the next 9 bits will be the  $n$  value. The  $Fcw$  input is the increment of the

accumulator. Increasing  $F_{cw}$  will increase the number of samples per sine/cosine period but decrease the sine/cosine frequency.

The  $s$  value from the accumulate is to control the *1's complement* and the *Sign/Exchange* blocks, as it is shown in Table 1. As it is described in previous section, the  $s$  is to exploit the symmetry of trigonometric functions so the whole  $[0, 2\pi)$  range can be covered by the values calculated in  $[0, \pi/4)$  (Hence, the amount of calculation is reduced).

The  $n$  value is used to index the cosine and sine LUT. As we select  $L = 16$ , while  $N = 7$ . The two LUTs are in the size of 128x16-bit, namely 256 bytes, respectively. The  $9x8$  multiplier is to convert the  $k$  value into  $\theta_k$ , namely, calculate  $k \times \pi/4$  while  $\pi/4$  is represented as  $0xc9$  in binary. Since one input to the multiplier is a constant, the multiplier can be simplified into several shift and add operations.

The two  $9x10$  multipliers are used to calculate  $X_{n,N} \times \theta_k$  and  $Y_{n,N} \times \theta_k$  while the final two adders and two subtractors give the magnitudes of the final sine and cosine output values.

The final output is represented in 1's complement format, namely, the topmost bit represents the sign (0 for positive and 1 for negative), while the lower 15-bit represents the magnitude of the cosine/sine value.

All the functional blocks described above are developed in RTL Verilog HDL code. This implementation is first simulated with ModelSim. Figure 5 shows the waveform of the cosine output.

We can see that the waveform do not look like a cosine wave. This is simply because that when the output value becomes negative, the sign bit (MSB bit) becomes 1. Thus the waveform "jumps" up.

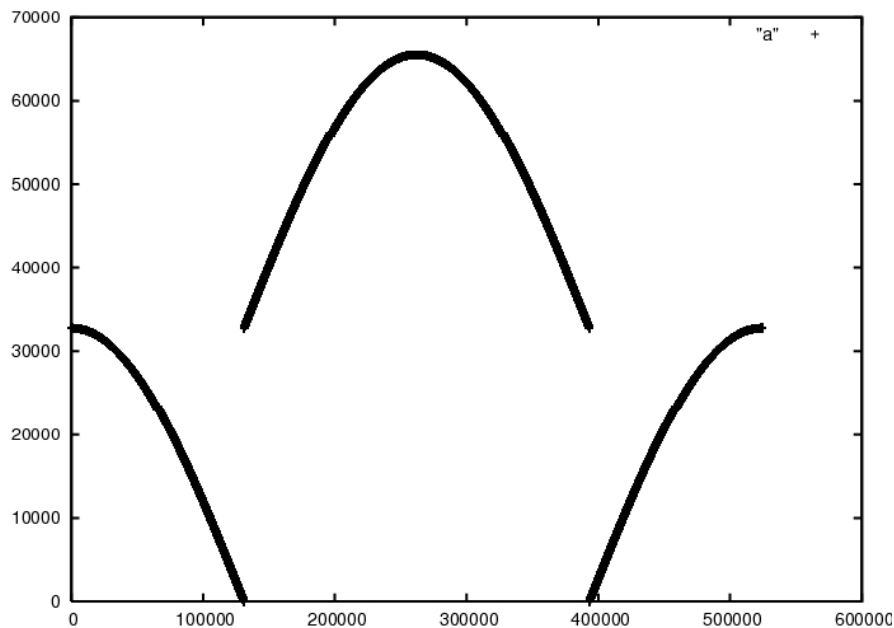


Figure 5: Cosine Output from Verilog Simulation

## 5.1 Synthesis Result

In this project, the *EP2C5 Cyclone II* low cost FPGA from *Altera* is used as the target device. The *Cyclone II* is a low-cost FPGA series while *EP2C5* is the cheapest one (contains least resource) in the series. The justification is that we want to limit available resource and try to obtain a low cost design. Another reason is that this FPGA provides internal RAM storage and multiplier structure. Those facilities are well-suited in this design.

The synthesis is carried out by the *Quatus II* EDA tool from *Altera*. Table 2 shows the synthesis result.

Here a logic element refers to a logic function implemented as a 2-bit input look up table. The registers used are for the accumulator and the final output registers. Since two 128x16-bit LUTs are used in our design, 4096 bits internal memory is consumed, as it is expected.

We can see that the maximal clock rate is 42.80 MHz. This is also the maximal output sample rates. In we define that at least  $M$  samples must be used to represent an whole period of sine/cosine waveform, the maximal sine/cosine frequency will be  $42.80/M$  MHz. For example, if  $M = 6$ , the maximal sine/cosine frequency will be around 7.13MHz.

The timing analysis also shows that the interconnect delay plays a significant part in the

<b>Resource Consumption</b>	
Logic Elements	145
Registers	64
Memory Bits	4096
Embedded Multiplier 9-bit Elements	5
<b>Timing Analysis</b>	
Max Clock Rate	42.80MHz
Clock Period	23.363ns
Total Cell Delay	15.045 (65.08%)
Total Interconnect Delay	8.073 ns(34.92%)

Table 2: Un-pipelined DFFS

total delay (34.92%). This does not agree with the assumption made in the course, where we only consider the computation delay.

## 6 Pipeline Implementation

To order to increase the output sample rate, the design is further optimized by using the cut-set re-timing technique. Basically pipeline registers are added in all the interconnects between functional units. Figure 6 shows the where the pipeline stage are added.

Thus, the DFFS is turned into a six-stage pipeline implementation. The synthesis result is shown in Table 3 while Figure 7 shows the change of resource consumption and delay relative to the original pipeline design.

From the table as well as the figure, we can see that the number of logic elements and registers increases to 150 % and 300% respectively, compared with the original un-pipeline design. While the consumption of memory bits and internal multiplier elements is not changed, as it is expected.

However, in theory a six pipelined implementation can bring a 6x speed-up in clock rate, the result shows that only 2.6x speed-up is resulted. The insignificant improvement can be

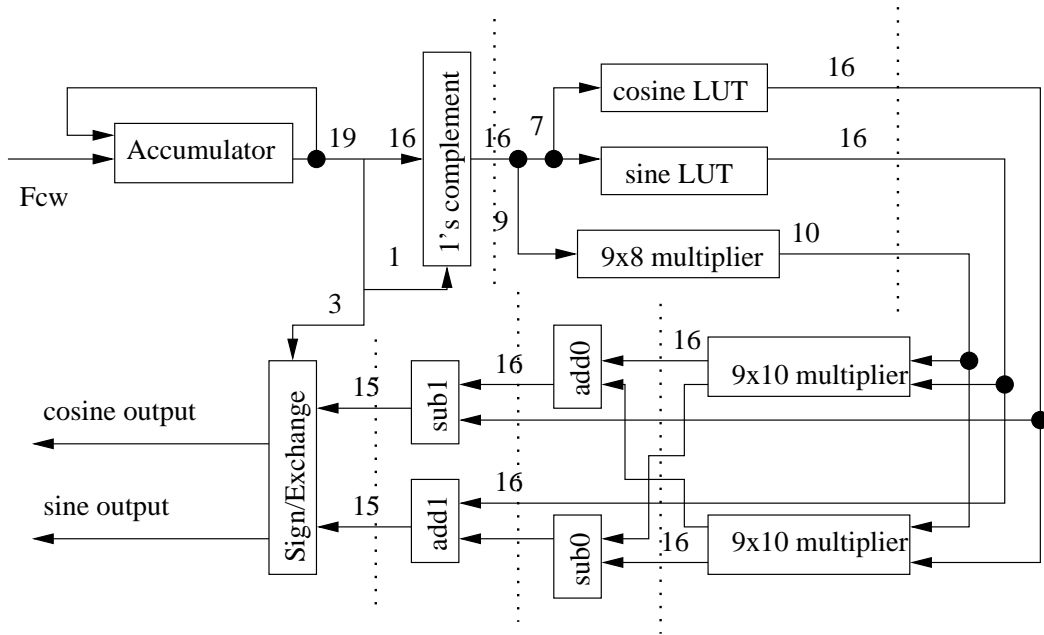


Figure 6: Pipeline Version

due to following reasons:

- The delay of each pipeline stage is not the same. Thus the maximal clock rate is determined by the longest pipeline stage. Unfortunately, the synthesis does not provide the delay of each functional units. Furthermore, some functional units which are susceptible to have long latency, such as the LUT and multipliers, are unable to be implemented in a pipeline version. This is because they are implemented internally with dedicated resource which by design is unable to further pipelined.
- The pipeline registers bring in additional overhead.
- The interconnect delay is unmanageable. As we can see that the interconnect delay plays a significant part of the total delay. However since the amount of interconnect delay is related to how the functional blocks are mapped onto the FPGA resources, which is solely controlled by the synthesis software rather than the designer (unlike in ASIC or custom IC design). On the other hand, it is good to see that in the pipeline implementation, the interconnect delay accounts for about 26.5% of the total delay, less than 35% as in the un-pipeline version. This shows that pipelining can result in a more compact design, which reduce the length of interconnect.

<b>Resource Consumption</b>	
Logic Elements	222
Registers	191
Memory Bits	4096
Embedded Multiplier 9-bit Elements	5
<b>Timing Analysis</b>	
Max Clock Rate	116.67MHz
Clock Period	8.503ns
Total Cell Delay	6.251 (73.52%)
Total Interconnect Delay	2.252 ns(26.48%)

Table 3: Pipelined DFSS

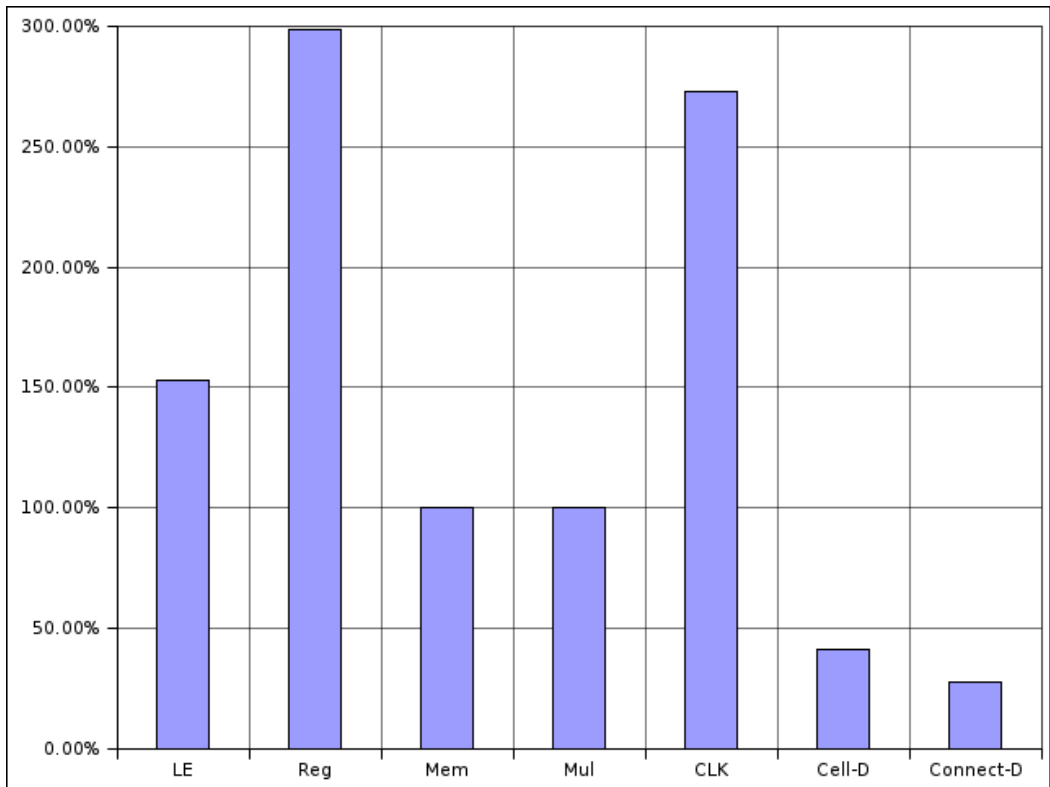


Figure 7: Pipeline Vs. Un-pipeline Implementation

## 7 Conclusion

In this project, we study a hybrid DDFS algorithm. Given an angle, this algorithm first read from two LUTs to get the angle's coarse position then a angle rotation method is used to fine turn the angle.

We first implement it in software to study the algorithm's characteristics. Then the algorithm is implemented in a low cost FPGA in an un-pipeline fashion. A 42.8 MHz output sample rate is achieved. The implementation is further optimized to a 6 stages pipeline scheme, which results in a 2.6x speed-up.

## References

- [1] R. Andraka. A survey of CORDIC algorithms for FPGA based computers. *In Proceedings of the Sixth ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '98)*, 1998.
- [2] P. W. Baker. Suggestion for a fast binary sine/cosine generator. *IEEE Transaction on Computer*, 25, 1976.
- [3] D. D. Caro, A. Giuseppe, and M. Strollo. High-performance direct digital frequency synthesizers in 0.25 um cmos using dual-slope approximation. *IEEE Journal of Solid-State Circuit*, 40(11), 2005.
- [4] I. Janiszewski, B. Hoppe, and H. Meuth. Vhdl-based design and design methodology for reusable high performance direct digital frequency synthesizers. *Proceedings of the 38th Conference on Design Automation*, 2001.
- [5] C. Y. Kang and E. E. Swartzlander. An analysis of the CORDIC algorithm for direct digital frequency synthesis. *Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures and Processors*, 2002.
- [6] C. Y. Kang and E. E. Swartzlander. Digit-pipelined direct digital frequency synthesis based on differential CORDIC. *IEEE Transactions on Circuit and System*, 53(3), 2006.

- [7] S.-W. Lee and I.-C. Park. Quadrature direct digital frequency synthesis using fine-grain angle rotation. *Proceedings of 2004 IEEE International Symposium on Circuit and System (ISCAS'04)*, 2004.
- [8] A. Madisetti, A. Y. Kwentus, and J. A. N. Willson. A 100-MHz, 16-b, direct digital frequency synthesizer with a 100-dBc spurious-free dynamic range. *IEEE Journal of Solid-State Circuit*, 34, 1999.
- [9] H. T. Nicholas and H. Samueli. A 150-mhz direct digital frequency synthesizer in 1.25-um cmos with -90-dbc spurious performance. *In IEEE Journal of Solid-State Circuit*, 26(12), 1991.
- [10] Y. Song and B. Kim. A 16b quadrature direct digital frequency synthesizer using interpolative angle rotation algorithm. *Proceeding of Symposium on VLSI Circuits*, 2002.
- [11] Y. Song and B. Kim. A quadrature direct frequency synthesizer/mixer architecture using fine/coarse coordinate rotation to achieve 14-b input, 15-b output, and 100-dBc SFDR. *IEEE Journal of Solid-State Circuit*, 39(11), 2004.
- [12] A. G. Strollo, E. Napoli, and D. D. Caro. Direct digital frequency synthesizers using first-order polynomial chebyshev approximation. *Proceeding of 28th European Solid-State Circuits Conference (ESSCIRC 2002)*, 2002.