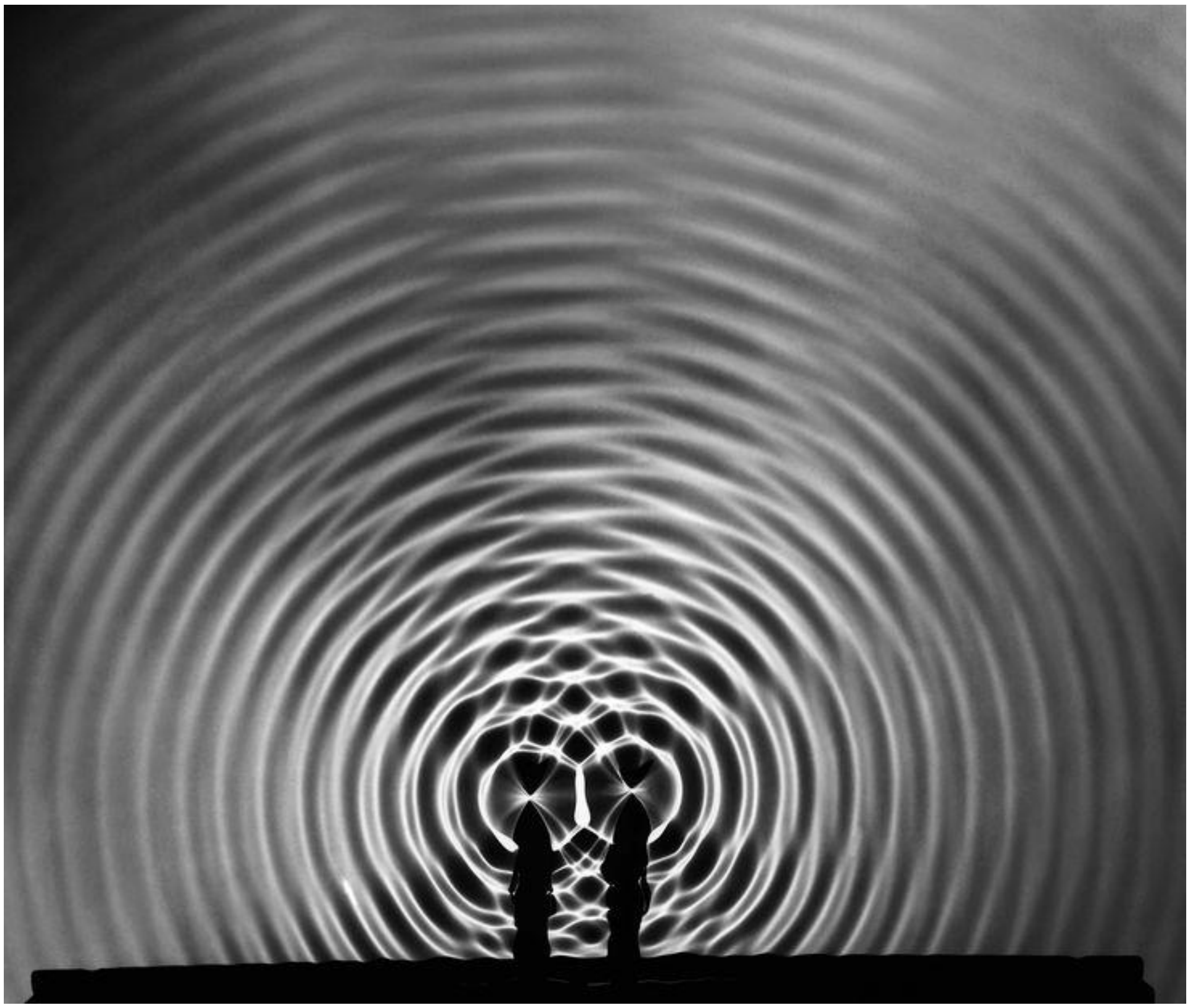
Beamforming and MIMO Digital Radio Baseband and Testbed for Next Generation Wireless System

Yiduo Xu

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2016-98
http://www.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-98.html

May 13, 2016



Capstone Report

Beamforming and MIMO Digital Radio Baseband and Testbed for Next Generation Wireless System

Niral Sheth, Yiduo Xu, Zhen Yuan

Advisors: Elad Alon, Vladimir Stojanovic

4/15/2016

University of California, Berkeley College of Engineering

MASTER OF ENGINEERING - SPRING 2016

Electrical Engineering and Computer Science

Physical Electronics and Integrated Circuits

BEAMFORMING AND MIMO DIGITAL RADIO BASEBAND AND TESTBED FOR NEXT GENERATION WIRELESS SYSTEM

YIDUO XU

This **Masters Project Paper** fulfills the Master of Engineering degree requirement.

Approved by:

1. Capstone Project Advisor:

Signature: ___________________________ Date ____________

Print Name/Department: ELAD ALON/EECS

2. Faculty Committee Member #2:

Signature: ___________________________ Date ____________

Print Name/Department: BORIVOJE NIKOLIC/EECS

Chapter 1: Technical Contribution

Yiduo Xu

Introduction

The past several decades have seen the development of wireless telecommunication industry at an unparalleled speed, which has connected more and more people together all over the world. This wouldn't be possible without the rapid evolution of wireless technologies like cellular data network. From the first generation (1G) to the current generation (4G), mobile phone data network has continued pushing its maximum speed limit, due to the growing demand of data-heavy tasks. Behind this trend lies fast generation evolution of supporting technologies and standards in the wireless industry. It has posed great challenges to developers and manufacturers of wireless products, since constantly designing and manufacturing new products are essential for them to keep up with the latest technology.

In light of this challenge comes our capstone project, which aims to design a hardware generator to produce flexibly structured wireless systems. With this generalized, easy-to-modify generator, wireless product companies would be able to reduce the time and monetary cost for next-generation product development. Then, consumers can enjoy the latest wireless technology at an earlier time with lower costs. Therefore, our research will benefit not only the producers, but also the consumers of wireless technologies. Specifically, within the project's digital radio baseband structure, our capstone team is responsible for building the beamforming and multiple input multiple output (MIMO) blocks. These blocks play an important role in the whole system, serving as the outermost device interface for transmitting and receiving wireless signals.

To optimize the efficiency of our capstone project, we created a task plan with a mix of individual and cooperative tasks. Each individual task was assigned to a team member, and each cooperative task also had a person in charge. The cooperative tasks include Chisel learning, CORDIC (coordinate rotation digital computer) implementation and demodulator implementation. They were designed to familiarize us with the tools and knowledge needed in the project. The rest of the tasks were core designing tasks, which were split among the three team members. I worked on integrating the CORDIC block, Niral designed the MIMO matrix multiplication block, and Zhen developed the beamforming matrix block. The work breakdown structure in Figure 1 demonstrates the relationship of our project deliverables and reporting deliverables.

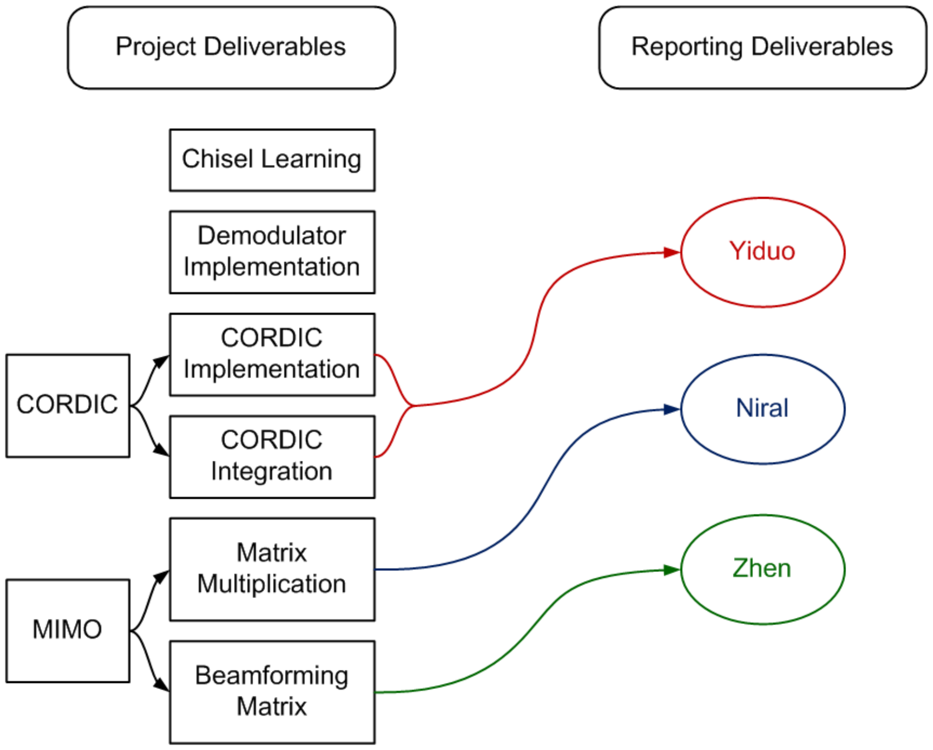


Figure 1. Work breakdown structure.

This technical contribution paper is focusing on the development and testing of an arithmetic calculation block called CORDIC. It was an essential component for many blocks in the whole digital radio baseband system, including the beamforming and MIMO blocks we built. In Niral's paper, the development of the MIMO matrix multiplication block is discussed, and Zhen's paper detailed the design of the beamforming matrix block.

The rest of this chapter is organized as follows: The second section describes the principle of the CORDIC algorithm; the third section presents the hardware structure designed for CORDIC; the fourth section illustrates the design realization in the hardware construction language Chisel; the fifth section explains the validation process and results; and the sixth section is a conclusion of this chapter.

CORDIC Algorithm

As an important basic block for the whole digital radio baseband system, COordinate Rotation DIgital Computer (CORDIC) is a collection of shift-add algorithms used for computing various arithmetic functions (Volder 1959). In our project, it was chosen as the basic block to perform calculations for many other blocks, including Carrier Frequency Offset (CFO), beamforming and MIMO.

The reasons why CORDIC was chosen over other algorithms were related to two aspects: the trend in telecommunication industry and the characteristics of CORDIC algorithm. In the past, digital signal processing (DSP) and communication industry was dominated by microprocessors. As technology involved, the microprocessors encountered a bottleneck in speed. With the development of reconfigurable hardware, people have been looking into hardware solutions to reach higher speed (Andraka 1998). Thus, we need to conduct research on hardware efficient algorithms and architectures to facilitate this transition. CORDIC is one of the hardware efficient algorithms. It is a class of iterative solutions for trigonometric, linear, logarithmic and other functions. The reason why it is hardware efficient is that it only uses shifts and adds to perform all the functions. It is also suitable for digital modulation and communication applications, because tasks like phase/frequency estimation and correction require a lot of trigonometric operations (Meher 2009). Thus, this algorithm is an ideal choice to be implemented in digital radio hardware systems.

The CORDIC algorithm is derived from the vector rotation formula. If rotating vector *(x, y)* for φ degrees results *(x', y')*, we have:

$$\begin{aligned} x' &= x\cos\varphi - y\sin\varphi \\ y' &= y\cos\varphi + x\sin\varphi \end{aligned} \tag{1}$$

By extracting *cos*φ, we get:

$$\begin{aligned} x' &= \cos\varphi(x - y\tan\varphi) \\ y' &= \cos\varphi(y + x\tan\varphi) \end{aligned} \tag{2}$$

In order to simplify the calculation, we limit the value of *tan*φ to $\tan\varphi = \pm 2^{-i}$. The rotation by an arbitrary angle can be achieved by performing a series of fixed angle rotations. The ± sign in *tan*φ can be represented with a direction variable d_i. Noticing that $\cos\varphi = \cos(-\varphi)$, the above equations can be expressed as an iterative rotation:

$$\begin{aligned} x_{i+1} &= K_i(x_i - y_i \cdot d_i \cdot 2^{-i}) \\ y_{i+1} &= K_i(y_i + x_i \cdot d_i \cdot 2^{-i}) \end{aligned} \tag{3}$$

Where

$$\begin{aligned} K_i &= \cos(\tan^{-1} 2^{-i}) = 1/\sqrt{1+2^{-2i}} \\ d_i &= \pm 1 \end{aligned} \tag{4}$$

During the iterations from *i* equals to 0 to *n*, we can ignore the constant K_i and add them back in the end of the computation. Then the iterations will have a systematic gain of A_n:

$$A_n = \frac{1}{\prod_n K_i} = \prod_n \sqrt{1+2^{-2i}} \rightarrow 1.647 (n \rightarrow \infty) \tag{5}$$

The gain converges to a constant value as the number of iterations approaches infinity.

Lastly, in order to calculate the total angle accumulated during the iterations, an angle variable z_i is added:

$$z_{i+1} = z_i - d_i \cdot \tan^{-1}(2^{-i}) \tag{6}$$

z_i can be expressed either in degrees or in radius.

The final equations we get are:

$$\begin{aligned} x_{i+1} &= x_i - y_i \cdot d_i \cdot 2^{-i} \\ y_{i+1} &= y_i + x_i \cdot d_i \cdot 2^{-i} \\ z_{i+1} &= z_i - d_i \cdot \tan^{-1}(2^{-i}) \end{aligned} \tag{7}$$

There are two ways to determine the direction of rotation in each iteration, which separate the algorithm into two modes. The rotation mode rotates according to the value of z_i, which rotates the vector for a specified angle; the vectoring mode rotates according to the value of y_i, which rotates the vector to align it with *x* axis.

In rotation mode, the desired rotation angle is input as z_0. The d_i is determined by the sign of z_i, in order to diminish the angle. In vectoring mode, the d_i is determined by the sign of y_i, which will diminish the vertical magnitude of the vector. If the initial angle z_0 was set to 0, the final angle z_n represents the total traversed angle of the process. The complete algorithm is summarized below in Figure 2.

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

where

d_i= -1 if $z_i < 0$, +1 otherwise

(a) rotation mode

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

where

d_i= +1 if $y_i < 0$, -1 otherwise.

(b) vectoring mode

Figure 2. CORDIC algorithm.

There is a limitation of rotation angle for the above algorithm. Because *i* starts with 0 in $\tan\varphi = \pm 2^{-i}$, the total traversed angle can only be [-π/2, π/2]. If the inputs exceed this region, they need to be rotated before passing into the algorithm. This is called pre-rotation, and there are many ways to do it. In this paper, a rotation of ±π is performed if the input is out of range.

By carefully setting inputs to specific values, or slightly changing the equations, the CORDIC algorithm can directly perform many trigonometric functions. Some examples include sine, cosine, arctangent, and polar/Cartesian transformation. Since the CORDIC generally adds one bit of accuracy in each iteration, the total number of iterations *n* is usually set to the number of bits of the inputs.

Hardware Structure

With the above understanding of the CORDIC algorithm, its hardware implementation was designed based on the basic structures proposed in Andraka's paper(1998). By combining two basic implementation structures, we developed the final mixed structure for our CORDIC block, which meets the flexible and parameterizable requirements of the final digital radio baseband system generator.

The first structure, called the iterative structure, is a straightforward hardware realization of the iterative CORDIC algorithm. It simply translates the three equations in the algorithm into shifters and adder/subtractors, shown in Figure 3. It performs one iteration in one cycle, and three registers are used to store the intermediate results after each iteration. A multiplexer (MUX) is placed at the input of each register to select appropriate value to put in. Before the algorithm starts, the MUXs select the block inputs x_0, y_0, and z_0, to initialize the registers; during the execution of the algorithm, the outputs of the registers go through the shifters and adder/subtractors, and the MUXs select the results after the adder/subtractors to put back in the registers, so the outputs from current iteration are fed back as inputs for next iteration. The number of bits shifted in the shifters is increased by 1 bit in each iteration. A Read-Only Memory (ROM) is used to store the angle values (in degrees or radians) corresponding to each arctangent value in each iteration. The address pointer to the ROM is also shifted by one address in each iteration to access the right angular value. Finally, the direction variable d_i is determined by the sign of y_i (vectoring mode) or z_i (rotation mode) in each iteration. In the last iteration, the final results can be read from the outputs of the adder/subtractors. Apparently, some control logic is necessary to realize the above described operation pattern.

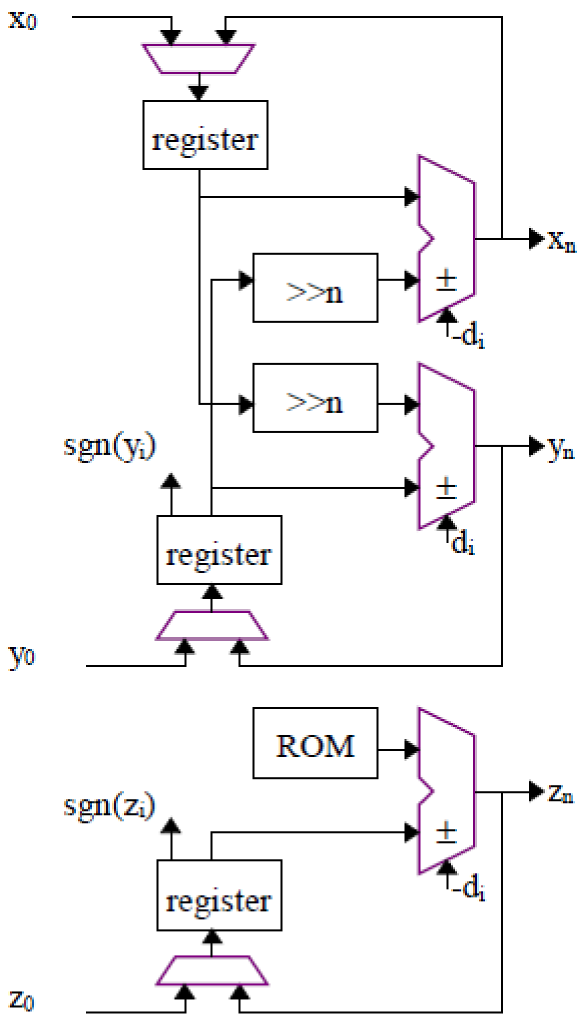


Figure 3. Iterative structure for CORDIC.

The second structure is called the unrolled structure, which is derived from the iterative structure to improve its performance. As concise and simple as the iterative structure is, its speed is limited because it can only produce one result after *n* cycles, so the data have to be passed in *n* times slower than the operating clock frequency. To finish all the calculations in one cycle, we can unroll the iterative structure, making *n* copies of it and connecting them together. This leads to the following unrolled structure in Figure 4. In this structure, the data are passed into the top adder/subtractors and shifters, go through *n* layers of the iterative structure, and then the results will appear at the outputs of the bottom adder/subtractors.

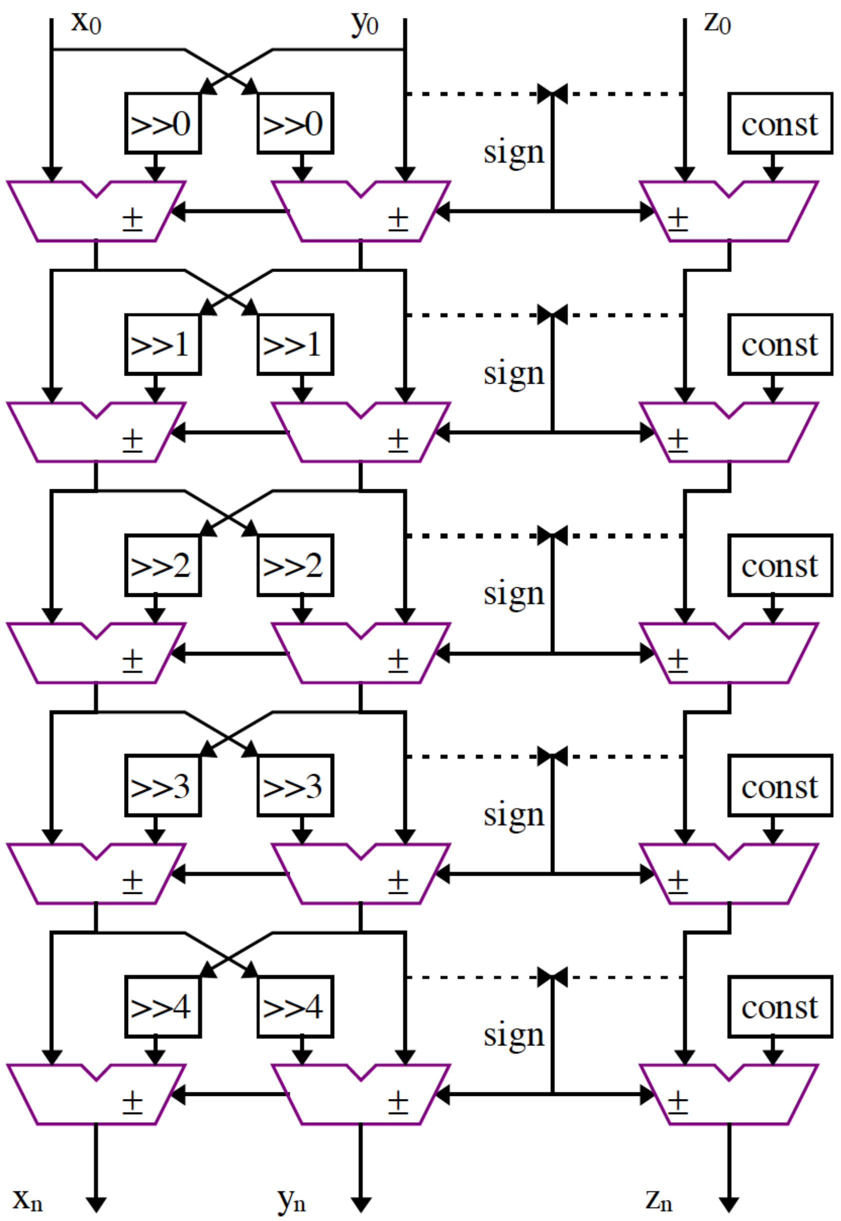


Figure 4. Unrolled structure for CORDIC.

Three major simplifications and related advantages resulted from the usage of the unrolled structure. First, all the shifters are now performing a fixed bit shift instead of a variable bit shift. Depending on where the shifter is in the unrolled chain, it still shifts the input by different bits; but each individual shifter doesn't change the bits shifted during the whole operation. This means they can simply be substituted by hard wiring. The second simplification is in the angle accumulation chain. Since the angle traversed in each iteration is fixed (ignoring the direction), the ROM in the iterative structure can be replaced with a set of constants connected to the angle adder/subtractor in each layer. These constants can also be hard wired, which will take much less space than a ROM. Because of these simplifications, the control logic can be reduced substantially. Third, the registers are no longer needed to keep the intermediate values, so the total delay to finish this algorithm will be reduced too.

However, this structure still has room for improvement. As the unrolled structure consists of the combinational logic entirely, its delay will be significant, which will limit the speed of the clock. Thus, registers can be asserted between the adder/subtractors along the same chain to pipeline this structure. Although this might worsen the total delay to the same level as the original iterative structure (*n* cycles with registers), the throughput will be improved by *n* times. As long as the data can be input at the clock frequency, this pipelined unrolled structure can produce one valid result in each time period needed to finish one iteration.

Based on the iterative structure and the pipelined unrolled structure, we came up with the mixed structure, which is the final design used in the project. As indicated by the name of the structure, it is a mixture of the previous two structures. A new term "CORDIC stage" was introduced to describe the components of the structure. Each CORDIC stage is an iterative structure with registers to hold values, and a series of CORDIC stages are connected to form the whole structure. The difference between this structure and pipelined unrolled structure is that the number of stages in mixed structure can be arbitrary. Each stage performs certain iterations, and then passes the data to the next stage. Also, the reference angle values are now stored in a lookup table (LUT), instead of ROM or hard wired. The total delay is still *n* cycles, and the throughput is dependent on the number of stages.

The mixed structure is ideal for our project, because it is generalized and provides flexibility for exploration. An unrolling factor *R* is used to parameterize this flexibility. It represents the degree of unrolling of the mixed structure. The number of CORDIC stages N_{stage} *(N)* in the structure equals to the unrolling factor *R* times total iterations *n*.

$$N_{stage} = R \times n \quad R \in (0,1] \tag{8}$$

If the unrolling factor is 0, there is only one stage in the block, so it equals to the area saving iterative structure; if the unrolling factor is 1, there are *n* stages which means the structure becomes pipelined unrolled with high performance. Therefore, by changing *R*, we can explore the tradeoff between area and performance for the CORDIC block. Depending on the applications, the optimal structure for CORDIC can be easily set with an appropriate unrolling factor.

Design in Chisel

The design of the mixed hardware structure of CORDIC was implemented in Chisel. Defined as a high-level hardware construction language, Chisel was developed at UC Berkeley (Bachrach 2012). It provides hardware construction flexibility by adopting parameterized generators and layered design. It also provides features like built-in standard interfaces and DSP support which comes handy in our project. Thus, it's very suitable for our project to make the hardware implementation flexible and generalized.

In this section, the high-level code structure of the CORDIC module will be described as well as the functions realized in the submodules. For detailed description of CORDIC module operation and how to set up parameters and input signals, please refer to Appendix A. The schematic structure of the whole CORDIC module in Chisel is illustrated in Figure 5. The Chisel hardware generator is in dashed lines. The main CORDIC block consists of *N* CORDIC stages in the pipeline, together with an overhead control block and peripheral blocks (Pre-rotation and scale multiplication blocks).

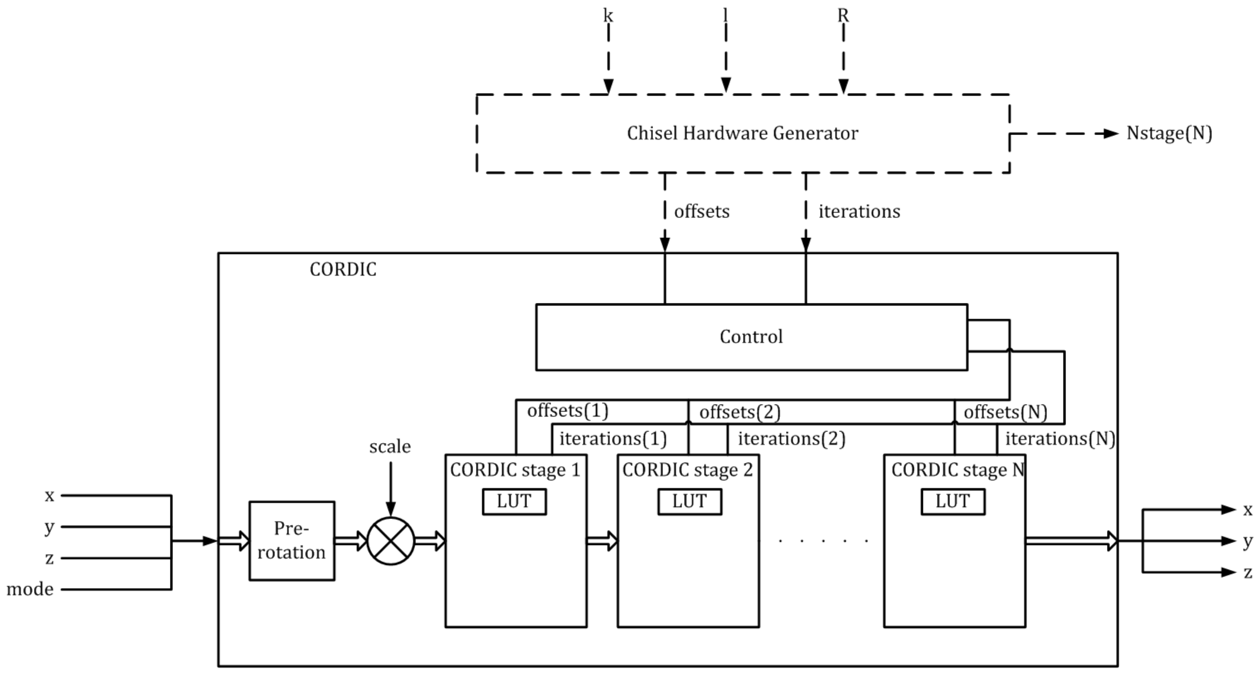


Figure 5. Chisel structure for CORDIC.

When the Chisel codes are compiled to generate hardware, the generator is run first to configure the hardware structure with a set of configuration parameters. Parameter *k* represents bit width of the vector *(x, y)*, parameter *l* represents bit width of the angle *z*, and

R represents the unrolling factor. The maximum of k and l determines the number of iterations n, and together with R, they can define the number of CORDIC stages *(N)* and the number of iterations each stage needs to perform. With this information, the offsets and iterations for each stage can be calculated. They are aggregated in two arrays of integers, and passed to the control submodule to control all the CORDIC stages.

After the configuration, the generated submodules perform different functions and work together to finish the CORDIC algorithm. As mentioned in hardware structure section, each CORDIC stage is one implementation of the iterative CORDIC structure in Figure 3. It takes in the inputs from the I/O interface with the previous block, calculates according to the look-up table (LUT) in it, and outputs the results after the number of iterations specified by the control block. The pre-rotation block is used to pre-rotate the input vectors if it is not within the range of the CORDIC algorithm, and the scale multiplication block compensates for the systematic gain before the start of the iterative calculation. Lastly, the control block utilizes offsets and iterations arrays to keep track of and control the CORDIC stage blocks. The same CORDIC stage block is reused for each stage from 1 to N, which is simpler than traditional hardware implementation. The top module and the stage modules also share the same uniformed control and data interface called *IO*, which is also essential to the flexible and general design of the whole radio system.

This design scheme achieved the flexibility goal of the project by utilizing the parameterized and hierarchical programming characteristics of Chisel. The configuration parameters consist of k, l, and R, which ultimately decide the number of CORDIC stages and the number of iterations each stage needs to perform. Thus, modification of the hardware structure can be done by simply tweaking these configuration parameters. The CORDIC module also supports two different data types, with the assist of a generic data type variable *gen*. It can be either *DSPFixed* or *DSPDouble*, and the rest of the code is valid in both conditions. Thus, with a single parameter during configuration, the generated hardware data type can be switched between fixed and double. This added another degree of flexibility. The design is also layered, with the instantiation of sub-modules like CORDIC stage within the top module CORDIC. This can reduce the complexity of the codes in each layer, and make programming more efficient and less vulnerable to bugs.

Validation

In order to verify the correct function of the CORDIC implementation in Chisel, a testbench was designed to thoroughly test all the possible operation conditions and make sure the results are as expected. This testbench was written in Scala language and can be used with C++ emulator, as well as CVS simulation for RTL, post-synthesis and post-place-and-route. In this section, the testbench is illustrated, and the results from the hardware generation and simulations are presented.

The testbench consists of two sets of tests. The first set verifies the rotation mode of the CORDIC and the second one tests the vectoring mode. Each set contains multiple individual tests which sets up a unit vector and a certain angle as inputs. The input angles of the tests in the same set can be either equally spaced out along the range [-π, π), or randomly generated.

The testbench can conduct these tests in three modes. The first mode is called serial mode. Inputs for a single test are passed into the circuit by the *poke* command, and then the tester tells the simulator to advance the simulation by certain cycles with the *step* command. At the cycle when the results are expected to appear at output ports, the tester uses the *check* command to compare them with reference values. The reference values are calculated in Scala with double precision floating point format and the original one step vector rotation formula. A flexible error tolerance threshold is used to make sure the error of the CORDIC module is reasonable. After the test passes, the tester proceeds to the next one until all tests are conducted. The second mode is very similar to the first one, except that the tests are run in a pipelined fashion instead of separately. This means the tester can feed new tests into the module whenever it is ready, without having to wait for the previous test to complete. In this mode, the CORDIC module's ability to handle multiple tasks in flight can be thoroughly tested. Some extra control in the tester is needed to keep track of the running tasks and check the results at the correct cycle. The last mode, bit accurate testing mode, is different from the previous two modes. As can be inferred from the name, the tester is evaluating the module's accuracy bit by bit. This is achieved by simulating the exact hardware computation in software, and comparing the results after each cycle. To match the bit width in hardware, *round/truncate* functions are used accordingly on the double type variables in software. This is the strictest mode of the testbench, in that it has zero error tolerance, which ensures the correctness of the module to the greatest extent.

This comprehensive testbench helped greatly during the debugging process. Different sets of tests and modes revealed different bugs in the Chisel codes for CORDIC module. It also automatically updates with the current hardware parameters, so the testbench can be easily re-run after modifying the Chisel codes.

The CORDIC module was verified by the testbench and pushed through the ASIC flow successfully. First, it passed the testbench in all conditions in the C++ emulator. After that, we chose a set of parameters and pushed it through the Design Compiler (DC) and the IC Compiler (ICC) to generate a complete circuit layout. The correctness of the circuit was verified after each stage with the testbench. Table 1 and Table 2 below summarize the setup and major results.

Table 1 CORDIC module ASIC flow input parameters

Width k for input signal x and y		
Integer bits	**Fractional bits**	**Sign bit**
5	10	1
Width *l* for input signal *z*		
Integer bits	**Fractional bits**	**Sign bit**
0	15	1
Unrolling Factor *R*	**Number of CORDIC Stages N**	**ICC Clock Period**
0.5	16 × 0.5 = 8	1.15ns

Table 2 CORDIC module ASIC flow result indexes

Area (um^2)			
Combinational	**Noncombinational**		**Total**
30736	3655		43611
Power (uW)			
Switch	**Internal**	**Leakage**	**Total**
1.79E+03	5.17E+03	3.37E+03	1.03E+04
Working Frequency (MHz)			
768			

The design point used in this flow operates with 16 bit wide signals and an unrolling factor of 0.5. The total occupied chip area is 43611 um^2 and the total power dissipated is 10.3 mW. The maximum working frequency is about 768 MHz, with a slack of -0.09 ns in the longest path for the specified 1.15ns clock period.

Conclusion

For the digital radio baseband generator that our capstone project was aiming to build, a calculation module CORDIC was designed, realized and verified. A flexible hardware structure for the algorithm was designed and implemented in Chisel. The complete design passed the comprehensive testbench, went through the ASIC flow and performed correctly in final stage simulation.

More research on the CORDIC block can be done in two aspects in the future. First, the design space of the block can be thoroughly explored with parameter sets, so as to find the optimal design point for a specific application. Second, the block can be integrated into the digital radio baseband system, and be tested together with other blocks. After the integration tests are finished for the whole system, it will be able to generate hardware for various wireless technologies.

Chapter 2: Engineering Leadership

Niral Sheth, Yiduo Xu, Zhen Yuan

Introduction

With the recent surge in wireless communications, the radio industry has seen an increase in the number of different communication standards, each requiring its own specific hardware and processing. Our project intends to address the need for radio interoperability with these various standards through the development of hardware generators for a Software-Defined Radio (SDR) system. These hardware generators will be created using Chisel (Bachrach 2012), a hardware construction language. When given a set of parameters or constraints, the hardware generators will output automated circuit designs for the given application, thereby accelerating the hardware design process and introducing a new methodology for multi-standard support. In this paper, we discuss topics relevant to bringing our project to market. These topics are divided into three sections: 1) the project's Intellectual Property (IP) approach, 2) the project's industry analysis, and 3) the project's market segment.

Trends and IP Strategy

With recent advances in semiconductor technology, the Integrated-Circuit (IC) industry has experienced rapid growth over the past few decades (Ulama 2015:6-9). However, the industry is now starting to stagnate due to the increasing complexity required in designing chips to provide competitive functionalities within demanding constraints (Sangiovanni-Vincentelli 2007: 467-68). In particular, new opportunities rising in the consumer electronics and Internet of Thing (IoT) domains have made time-to-market the primary concern for IC companies due to first-mover advantages (Smith 2014).

With the demand for shorter design cycles and higher volumes of functionality to be incorporated into designs, IC developers are facing costly project delays because changes in project requirements often necessitate large loop iterations due to the sequential nature of current industry design methodologies (Sperling 2014).

In addition to the design flow challenges, IC developers are facing problems with the role of IP in the semiconductor industry. Given the increasing complexity of chips, it is too costly and slow to develop all the functionalities from scratch. Hence, IC designers rely on licensing reusable system building blocks from an external party, known as IP blocks (Tamme, et al. 2013: 221). While these IP blocks can accelerate design cycles, the primary issue arises during system integration and verification. When incorporating a supplier's IP block into the system, no guarantee exists that the IP block will interact with other system components to provide correct functionality. Since these IP blocks are "black boxes", verification and modifications to the IP block to meet the developer's need become difficult, thereby creating delays and long design cycles.

Our project intends to address these problems in the wireless IC domain as it aims to implement a new design paradigm based on Agile and platform-based schemes. The development of flexible hardware generators achieves this by facilitating initial chip design to be independent of specific processes or hardware implementations such as IP blocks. By raising the level of design abstraction towards the desired functionality rather than a specific implementation, large loop iterations can be avoided since system components can dynamically change with requirements.

In bringing this project to market, our IP approach must maximize the project's impact on the wireless IC domain. A patent approach is not suitable for a few reasons. First, the project is part of ongoing research at the Berkeley Wireless Research Center, which follows a non-patent policy to encourage innovation. Second, the hardware generator design flow is based on Chisel, an open source language for creating circuit generators. Instead of obtaining a patent, we will be taking an open-source IP strategy to bring this technology to market. The primary motivation for this approach comes from the project's holistic goal of reshaping wireless IC design flows towards an Agile scheme to shorten design cycles and revive the growth of the IC and semiconductor industries. Taking an approach to protect the IP of this technology would only result in inhibited adoption of the new methodologies and limited growth of this new platform.

Industry Analysis

Within the broader wireless industry, our capstone project targets two specific technologies: Wi-Fi and cellular data. These two industry sectors were chosen as they contain common characteristics and challenges that our project addresses.

The first common characteristic of Wi-Fi and cellular data network is that they are both widely used. Wi-Fi is becoming the standard Internet access method in various environments such as households, offices and public places (Henry 2002). Cellular data service is also reaching more and more people with the rapid development of the smartphone industry. As we are building a completely new platform for the wireless industry, choosing Wi-Fi and cellular data will allow us to maximize the number of potential developers who will benefit from the adoption of our hardware generators.

Secondly, both Wi-Fi and cellular data have development patterns consisting of rapid generation iterations and continuous improvement potential. Since the introduction of first generation Wi-Fi in 1997, it has evolved to fifth generation within 15 years (Nagarajan 2012). Cellular data networks exhibit the same pattern, as the fifth generation is expected to be commercialized in the near future. These trends incentivize our design of flexible and parameterizable generators to reduce application redesign costs resulting from generation transitions.

The steep development curve and considerable future potential of Wi-Fi and cellular data networks have brought great challenges to the hardware design process. In the past, it would take engineers many years to design a series of new devices from scratch for each generation of wireless technology. This has delayed the new technology from reaching potential customers before the next generation emerges. In fact, some generations of the technology have suffered from a lack of supporting devices (Ferro 2005). Our project aims to ease this transition process by providing a flexible and generalized design framework. Our generators will consider the key factors that change between generations of technologies and will make them into parameters. Different hardware designs can then be produced by the generators, thereby reducing the development time for new device design and old device upgrade.

With the understanding of our industry above, we analyzed the five market forces (Porter 2008) on the cellular data industry to determine the profitability of entering the market. To be more specific, we are considering the market from the perspective of a hardware company that sells signal processing chips for smart phones.

First, the **threat of new entrants** would be weak. This is because the cellular data network industry greatly relies on technology, which makes it difficult to enter without substantial expertise of this area. New entrants would also struggle with the lack of credibility, which is essential for selling products to the customers in this industry. This leads to our second force, the **bargaining power of buyers**. The buyers of our signal processing chips would be major mobile phone companies like Apple and Samsung. The size of these companies indicates their strong bargaining power, because they could compare the reliability, price, and performance of our product with many other alternative offers. The third force, **threat of substitutes**, is weak according to our analysis. Even though people can use Wi-Fi to connect to the Internet with their smartphones, the cellular data connection is an indispensable feature for any smartphone nowadays. Thus, there is almost no substitute technology. Fourth, the **bargaining power of suppliers** is also weak. The fabrication process for integrated circuit chips is standardized and many fabrication factories exist, thus allowing control of supplier costs. Lastly, the **rivalry among existing competitors** would be strong and feature-based. With the rapid development of wireless technology, the chip company that develops the first next-generation chip would obtain the biggest share of the market. Before other companies can catch up, enter the market and bring down the price, the industry might have already moved into the next generation.

As a whole, the three weak forces and a strong feature-based rivalry suggest promising profitability in this industry. Since our project would serve as a platform for this industry's developers, these results are great motivations for us.

Market Strategy

According to the end-user industries, the SDR market is mainly subdivided into telecommunication, defense and public safety (Saha 2015). Considering SDR and Chisel, we focus our market segment on the telecommunication industries for a few reasons. First, our platform will be open-source, which heavily relies on a substantial contributor base. For commercialized industries like telecommunication, there are many engineers contributing to the open source community. However, for other industries such as conventional defense and public safety, the aim of the communication system design is confidentiality and reliability rather than commercialization, so it is difficult to work on open source code. Second, the telecommunication industry has a big group of customers, so there will be extensive user feedback regarding the products which utilize our platform. Last, the competition among telecommunication industries is stronger than that in other industries. In order to obtain a competitive advantage in this market, companies are in great need of higher product quality and shorter design cycles, which can be achieved by using our hardware generators.

Before going to market, the users of our hardware generators must be defined. The two main categories of users that benefit from our project are university researchers and industry engineers. They are responsible for developing code, verifying it, and improving their design. University researchers can take advantage of the generators when designing new frameworks for the communication system. On the other hand, industry engineers can more effectively keep their designs up to date by using our generators, making it easier to go to market.

Works Cited

Andraka, R.

1998 A survey of CORDIC algorithms for FPGA based computers. Proceedings of the 1998 ACM/SIGDA sixth international symposium on Field programmable gate arrays, ACM, 191-200.

Bachrach, J., Vo, H., Richards, B., Lee, Y., Waterman, A., Avižienis, R., ... and Asanović, K.

2012 Chisel: constructing hardware in a scala embedded language. Proceedings of the 49th Annual Design Automation Conference, ACM, 1216-1225.

Ferro, E. and Potorti, F.

2005 Bluetooth and Wi-Fi wireless protocols: a survey and a comparison. Wireless Communications, IEEE, 12(1):12-26.

Henry, P. S., and Luo, H.

2002 WiFi: what's next?. Communications Magazine, IEEE, 40(12):66-72.

Meher, P. K., Valls, J., Juang, T. B., Sridharan, K., & Maharatna, K.

2009 50 years of CORDIC: Algorithms, architectures, and applications. Circuits and Systems I: Regular Papers, IEEE Transactions on, 56(9), 1893-1907.

Nagarajan Vijay

2012 5G WiFi: Introducing a Wi-Fi Powerful Enough to Handle Next-Gen Devices and Demands. http://www.broadcom.com/blog/wireless-technology/5g-wifi-introducing-a-wi-fi-powerful-enough-to-handle-next-gen-devices-and-demands/, accessed October 18, 2015.

Porter, M. E.

2008 The five competitive forces that shape strategy. Harvard business review, 86(1):78-93.

Saha, Sudip

2015 FMI: Software Defined Radio (SDR) Market Analysis, Segments, Growth and Value Chain 2014-2020. https://www.newswire.com/press-release/fmi-software-defined-radio-sdr-market-analysis-segments-growth, accessed February 27, 2016.

Sangiovanni-Vincentelli, Alberto

2007 Quo vadis, SLD? Reasoning about the trends and challenges of system level design. Proc. IEEE, 95(3):467-506.

Smith, Randy

2014 Is IC Design Methodology At The Breaking Point? Semiconductor Engineering. http://semiengineering.com/is-ic-design-methodology-at-the-breaking-point/, accessed February 7, 2016.

Sperling, Ed

2014 Time To Market Concerns Worsen. Semiconductor Engineering. http://semiengineering.com/time-to-market-concerns-worsen/, accessed February 7, 2016.

Tamme, S., S. Schott, D. Gunes, J. Wallace, R. Boadway, F. Razavi, and M. Pépin

2013 Trends And Opportunities In Semiconductor Licensing. les Nouvelles : 216-228.

Ulama, Darryle

2015 IBISWorld Industry Report 33441a Semiconductor & Circuit Manufacturing in the US. http://www.ibis.com, accessed October 18, 2015.

Volder, J. E.

1959 The CORDIC trigonometric computing technique. Electronic Computers, IRE Transactions on, (3), 330-334.

Appendix A Instructions for CORDIC Generator Users

CORDIC Generator

This repository contains the ChiselDSP code used to generate CODRIC module. The CORDIC generator can generate the module with a configurable mixed structure, Fixed/Double type and arbitrary bit width.

Generator Parameters

See ChiselDSP Readme for ChiselDSP-specific JSON parameters. Additionally, the CORDIC requires the following fields (see JSON):

- **Cordic**
    - **zIntBits**: Number of integer bits of signal **z**.
    - **zFracBits**: Number of fractional bits of signal **z**.
    - **R**: Unrolling factor. *Nstage = ceil(R × iterations). (R ∈ (0, 1])* When **R = 0**, the number of stages is 1, the structure of the generated CORDIC is iterative; when **R = 1**, the number of stages is the number of iterations, the structure of the generated CORDIC is fully unrolled.

Note #1: The bit widths of signal **x** and **y** have to be the same. They are defined by the **intBits** and **fracBits** parameters in the default **complex** field in the JSON file.

Note #2: The total width of a *Fixed* type signal is equal to integer width + fractional width + sign bit (1).

Note #3: The number of iterations is the maximum of **x/y** width and **z** width.

Note #4: The CORDIC module achieves highest accuracy when the width of signal **x** and **y** is the same as **z** (the position of the decimal point can be different). If you modify them to be not the same, you may need to adjust the error tolerance **error_Vec/error_Ang** in CORDICTester to pass the tests. I am not sure if it is due to the nature of CORDIC algorithm or the hardware implementation.

External Interface

INPUTS

- **reset**: Only used for setting default values.
- **clk**: Calculation clock (See **Note #2**).
- **io_in_valid**: Standard valid IO interface with upstream. *It should only go high for 1 clk cycle for each input.*
- **io_in_bits_x**: Input signal **x**.
- **io_in_bits_y**: Input signal **y**. **io_in_bits_x** and **io_in_bits_y** forms the input vector (**x**, **y**).
- **io_in_bits_z**: Input signal **z**. **z** is the input angle.
- **io_in_bits_mode**: This *DSPBool* signal determines CORDIC operation mode. *True (1)* for ROTATION mode and *false (0)* for VECTORING mode.

OUTPUTS

- **io_out_valid**: Standard valid IO interface with downstream. It will go high when an output is valid, and the downstream is expected to collect the data right away (ie. the CORDIC module won't hold data for downstream).
- **io_out_bits_x**: Output signal **x**.
- **io_out_bits_y**: Output signal **y**. **io_out_bits_x** and **io_out_bits_y** forms the output vector (**x**, **y**).

- **io_out_bits_z**: Output signal **z. z** is the output angle.
- **io_out_bits_mode**: CORDIC operation mode output (mainly for debugging purpose).

Note #1: Inputs are asserted slightly after the **clk** *rising* edge. The testbench also checks the outputs slightly after the **clk** *rising* edge.

Note #2: The throughput of the CORDIC calculation is not always the inverse of the clock. See the following **Timing Details** section.

Timing Details

- The maximum throughput of the CORDIC module is related to the unrolling factor **R** and the total number of iterations. Approximately, new data can be input every **1/R** cycles, and a set of valid outputs are produced every **1/R** cycles too. (See **Note #1**) For example:
  - **R = 1**, **16** bits signal: New data can be fed in every (**1**) cycle.
  - **R = 0.5**, **16** bits signal: New data can be fed in every two (**2**) cycles.
  - **R = 0**, **16** bits signal: New data can be fed in every **16** cycles.
- The latency of the CORDIC module is always the number of iterations plus two (2) cycles (for pre-rotation and scale multiplication). For example, for **16** bits signal, the latency is **18** cycles regardless of **R**.

Note #1: The actual data processing rate is T = ceil(iteration/Nstage) = ceil(iteration/ceil(R × iteration)).
