A Hybrid Retrodirective Array Based on Delay-Locked Loop Hybrid Phase Conjugators

by

Michael Bolt

A dissertation submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Auburn, Alabama
August 8, 2020

Keywords: Antenna Array, Retrodirective Array, Phase Conjugator, Wireless Communications, Phased Array, Delay-Locked Loop, Phase-Frequency Detector

Approved by

Mark L. Adams, Chair, Associate Professor of Electrical and Computer Engineering
Stuart Wentworth, Associate Professor of Electrical and Computer Engineering
Victor Nelson, Professor Emeritus of Electrical and Computer Engineering
Thaddeus Roppel, Associate Professor of Electrical and Computer Engineering
Michael Fogle, University Reader, Associate Professor of Physics

Abstract

This dissertation discusses the design, construction, and characterization of a new Hybrid Retrodirective Array (HRA) as a possible beamforming technology for 5G wireless communication networks. A Delay-Locked Loop based Hybrid Phase Conjugator (DLL-HPC) capable of functioning as both an actively controlled phase shifter and a closed-loop phase conjugator is presented as the basis for the new HRA. The HRA is shown to be capable of fast acquisition times, ideal transmit and receive beamforming, automatic mobile target tracking, direction of arrival reporting, and array-geometry independent operation. Test results show initial acquisition times up to 50x faster than current beamforming standards and the ability to automatically track mobile users with no communication overhead. This work describes in detail the operation of the HRA system with special attention given to the theory, construction, and testing of a full prototype. Additionally, a novel rollover phase-frequency detector (R-PFD) circuit is presented alongside transistor-level integrated circuit design and simulations as an enabling technology for future implementations of delay-locked loop (DLL) based beamformers. This dissertation serves as the first steps towards HRA-based network architectures through several novel contributions: a new hybrid phase conjugator circuit topology which improves settling time and backwards compatibility over existing hybrid retrodirective arrays and a new R-PFD circuit capable of enabling a new class of DLL-based beamformers.

Acknowledgments

There are many people who have supported me in my studies and academic pursuits, each of which deserve more than a simple thank you on this page. Sadly, this is all that I have room for.

I would like to thank Dr. Adams for tirelessly supporting and encouraging my work and research efforts. You never held me back from an interesting topic or project, and for that I will always be grateful.

I would like to thank Craig, Tyler, Haley, Drew, Brent, Edward, Henry, George, Daylon, Andrea, and Joey for being the best peers possible. You were the reason the STORM Lab was such a great environment to work and help each other in.

I would like to thank Andrew, Drew, Ryan, Alex, Emma, John, Laura Grace, Justin, and all of the other undergraduates who I've gotten to work with over the years. You were all great helps to your projects and great friends to me.

I would like to thank Dr. Wentworth for always letting me bother him in his office with questions and distractions.

I would like to thank Dr. Nelson for encouraging my desire to teach and fostering a love of embedded systems.

I would like to thank the members of my committee for all of their time and work reviewing my drafts. This is a long document, and I'm sorry you had to read it more than once.

I would like to thank my family members for putting up with me being in school for so long and for supporting me at every step.

Most importantly, I would like to thank my wife Naomi. You've supported me since I first told you I wanted my Ph.D. in our freshman year some 7 years ago. Your support and love through years of school together and apart have meant more to me than anything else ever could. I could not have accomplished this without you.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Wireless communication systems are ubiquitous in the modern world. What's more, the way that we implement wireless communication must constantly evolve to new meet the new demands of an increasingly technologically centered society. In order for us to keep advancing, we need new approaches to wireless communication that are both backwards-compatible and forward-looking in their adaptability.

1.1 Our Interconnected World

As wireless technologies have become smaller, faster, lighter, and cheaper through innovation, they have come to occupy every facet of modern life. Since the first wireless transmission in 1866 [1], humans have found ways to integrate wireless connectivity into nearly everything.

Due to the recent trend of the Internet of Things, a "smart" version of almost any device can be purchased. Everything from a refrigerator [2] to a water bottle [3] to a microwave [4] can integrate a WiFi or Bluetooth connection to enable some additional features. In addition to gimmick devices, the Internet of Things has also given rise to Smart Home devices, such as thermostats and security cameras, which are used by 28% of consumers in America [5]. Each of these individual devices adds another wireless transmitter and receiver and consumes more bandwidth of the world's networks.

The entertainment industry is also increasingly relying on wireless connectivity. A 2019 report by Deloitte found that the average household contains eleven wireless connected devices, of which seven contain screens used for viewing and consuming media [5]. A separate survey by Parks Associates found that 71% of American households have an internet connected entertainment device [6]. These trends in connected hardware are matched by the rise of internet streaming giants such as Netflix, Disney+, Hulu, and YouTube, and the internet streaming market is anticipated to grow to a value of $184.3 billion by 2027 [7].

In addition to the current billions of wireless connected devices, the upcoming fifth generation of mobile data communication (5G) anticipates even more growth. 5G services are projected to have 5.9 billion subscriptions by 2024, and each of these subscription holders will bring multiple connected devices to these new networks [8]. In addition to the increase in number of devices, 5G networks aim to provide multi-gigabit data rates through the use of K_a band carrier frequencies [9, 10]. These new networks provide both significant opportunity for new user experiences and significant challenges for the engineers attempting to implement them.

1.2 Challenges for 5G Wireless Communications

5G standards are beginning to roll out across America, but advanced beamforming techniques have been identified as a key enabling technology without a full solution [9–13]. As carrier frequencies rise into the K and K_a bands from current 4G frequency bands, the radiated wavelength is shrinking from 37.5 cm to 1.2 cm at these "mmWave" frequencies [9]. This means that individual antennas will be able to shrink by a factor of around 30, making extremely narrow and highly directional beams possible through integrated antenna arrays [9, 14–17].

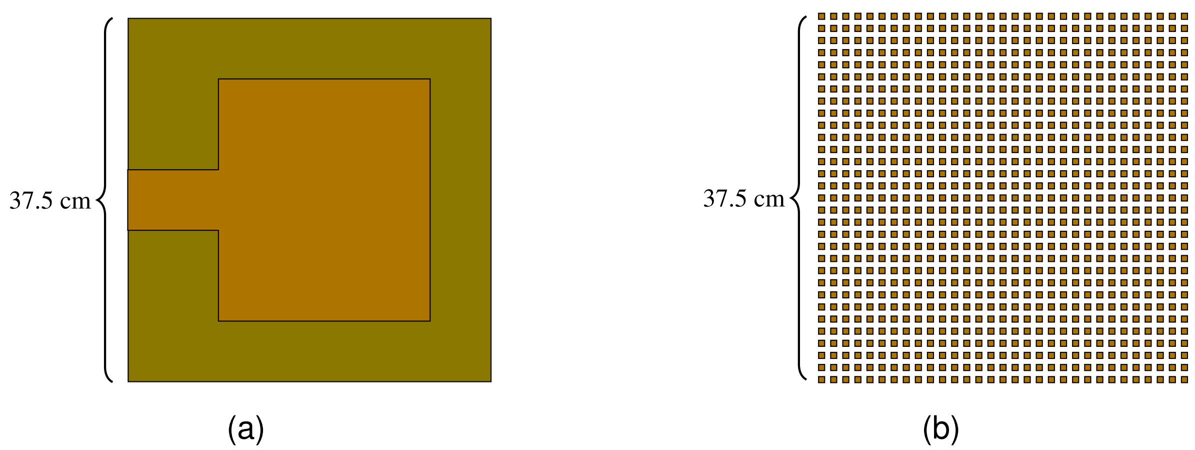


Figure 1.1: (a) 4G patch antenna; $f = 800$ MHz, $\lambda = 37.5$ cm (b) 5G 900 element patch antenna array; $f = 24$ GHz, $\lambda = 1.2$ cm

Ultra-dense antenna arrays will provide incredibly high gain, but this will be necessary to overcome the high path-loss of mmWave frequencies [11, 14, 18, 19]. Additionally, the narrow pencil beams created by these large-scale antenna arrays will help to reduce the amount of interference among many mobile users, allowing 5G wireless systems to be modeled as point-to-point connections rather than widespread transmissions [14, 15, 18–20]. By using mmWave carrier frequencies, 5G systems intend to enable multi-gigabit throughput among large numbers of users within small areas, such as a a home or office space [9–11, 14–16, 21].

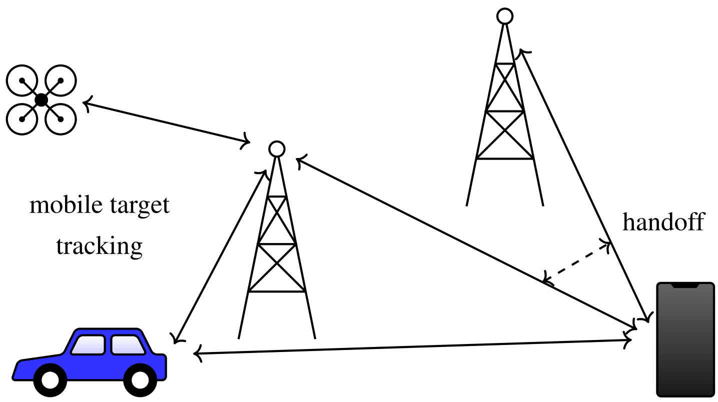


Figure 1.2: 5G mobile network challenges

These high frequencies and ultra compact arrays pose challenges that current beamformer strategies cannot overcome. The dynamic nature of a 5G environment means that connections will need to be established quickly and to continually update their beamformer point to track moving targets [14, 16]. Current techniques, such as those implemented in IEEE 802.11ad/ay for WiFi signals, have high costs associated with both acquiring and updating a beamformer point [11]. These methods rely exclusively on the transmission of beamformer training information between users and can require many communication frames to establish a strong connection; this raises communication overhead and reduces throughput [14]. Further, these methods only reevaluate the current point if the communication link degrades, making them unsuited for dynamic environments with mobile users [16]. Finally, the aforementioned pencil beams that will be utilized in mmWave 5G communications have been shown to lead to frequent beam switching and deafness when misaligned under these protocols, creating a tradeoff between beamwidth and data throughput [13, 15, 20, 21].

Although mmWave 5G promises unprecedented bandwidth and connectivity, no solution currently exists to handle the looming challenge of beamforming. The currently proposed techniques can take up to 1.8 seconds to establish a point among among antenna arrays [14], and cannot easily be expanded to antenna arrays with tens or hundreds of elements as pictured in Fig. 1.1. This challenge is the driving force for the research topic of this dissertation: a hybrid retrodirective array architecture.

1.3 A Hybrid Retrodirective Array

A retrodirective array operates on the principle of retrodirection, or transmitting a signal back along the direction of arrival of a received signal. These were first demonstrated in the 1960s to create radar transponders and range finders [22, 23], and showed the ability automatically lock onto a received signal and actively track movement without any communication between the users, making them a promising new technology for communications systems. However, the circuitry required to implement these first retrodirective arrays was clunky, expensive, and left no room for intelligence signal modulation. For these reasons, the retrodirective array fell into obscurity for some 40 years.

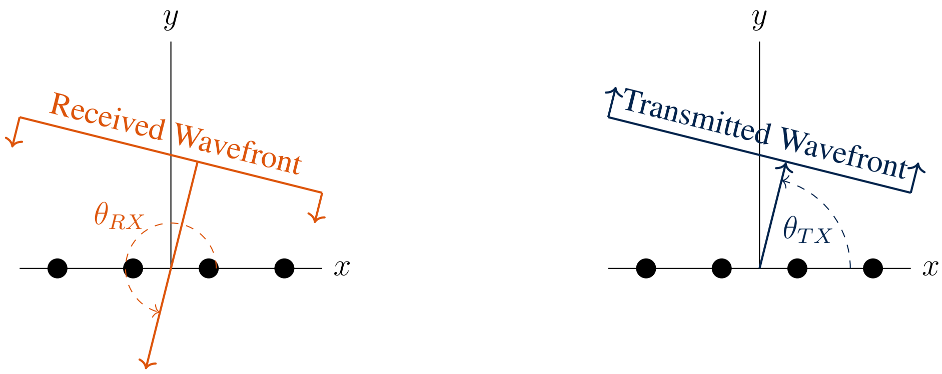


Figure 1.3: A retrodirective antenna array transmits back along the direction of arrival of a received signal

In the last two decades, research interest in the retrodirective array has been renewed due to cheaper, smaller, and better components [24]. New retrodirective array architectures have been shown that implement intelligence signal modulation and demodulation, making retrodirective array based communication systems possible [25]. However, these new architectures still suffer from the fundamental flaw of retrodirective arrays: they can only function properly when they are receiving a signal. This means that retrodirective arrays could only be used in communication systems which employ a pilot tone [26] or continuous, always-on data transmissions.

In 2018, the first hybrid retrodirective array capable of performing as a retrodirective array or actively controlled antenna array was published [27]. This new type of retrodirective array allowed for arbitrary intelligence signal modulation, automatic lock-on and tracking, and received signal dropout immunity by switching to an actively steered mode. With these improvements, a retrodirective array capable of handling time-division duplex and non-continuous data-streams became possible.

This dissertation details the theory, design, and characterization of a new hybrid retrodirective array architecture with faster acquisition times and backwards compatibility with previous beamformer protocols. A novel delay-locked loop based hybrid phase conjugator is presented as the basis for arbitrarily sized and shaped hybrid retrodirective antenna arrays, along with integrated circuit transistor-level designs and simulations of the unique system components. Conclusions and possible future research efforts are outlined at the end, with guidance for future hybrid retrodirective array designs.

Chapter 2

Electronic Beamforming in Communication Systems

Modern electronic communication systems use electronic beamforming to control the directivity of transmitted signals. However, the current standards and techniques have been found insufficient for upcoming 5G networks and new mobile applications [11]. Before diving into the theory of retrodirective arrays and their benefits, we will first discuss the basics of electronic beamforming for linear antenna arrays and some of the current techniques and standards used for modern communication systems.

2.1 Linear Antenna Array Beamforming

Let us assume there are two isotropic antennas placed some distance d apart as in Fig. 2.1. Because the antennas are isotropic, they radiate power equally in all directions [28]. This allows us to examine what happens when a signal is applied to both antennas simultaneously without worrying about the specific antenna pattern.

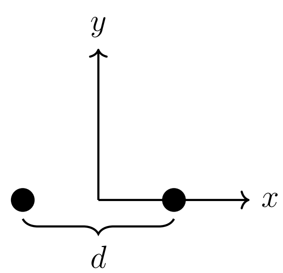


Figure 2.1: A two-element antenna array with spacing d

The electric field radiated by an isotropic element propagates radially from the element [28]. In the near-field, these waves will have a noticeable curve to them and be easily identifiable as a circle. However, in the far-field the radius of curvature becomes so large that the radiated wave can be approximated as a planar wavefront [29]. If we feed the same signal into both elements of a two-element antenna array that are one half-wavelength apart, the waves will propagate as shown in Fig. 2.2. From this figure we can see that there are some values for the view angle θ at which the waves will add constructively in-phase and some values where they will add destructively out-of-phase. Because the electric field strength is doubled along the directions of constructive interference, a four-fold increase in transmitted power occurs along these directions as well [30].

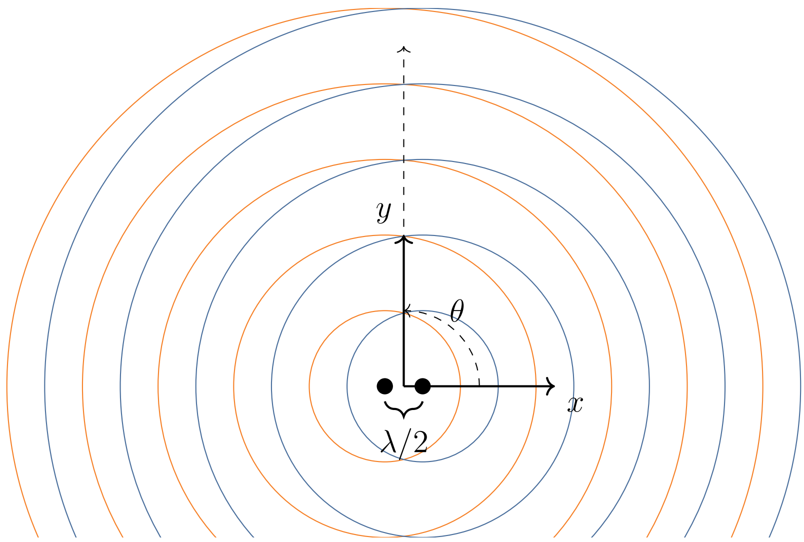


Figure 2.2: Two antennas one half-wavelength apart driven with an identical signal

In an electronically steered antenna array, an identical signal is not fed into each antenna element. Rather, the signal at each element undergoes a phase shift, or time delay, to allow the direction of maximum power transfer to be controlled. As the signals being transmitted are sinusoidal in nature, they take the form:

$$TX(t) = cos(2\pi f \cdot t + \phi) \tag{2.1}$$

By manipulating the ϕ term through a controllable phase shifter, the signal is transmitted from each antenna element at a different point in time. This staggering of transmission in the time domain will alter the view angle θ along which the signals add constructively, as shown in Fig. 2.3.

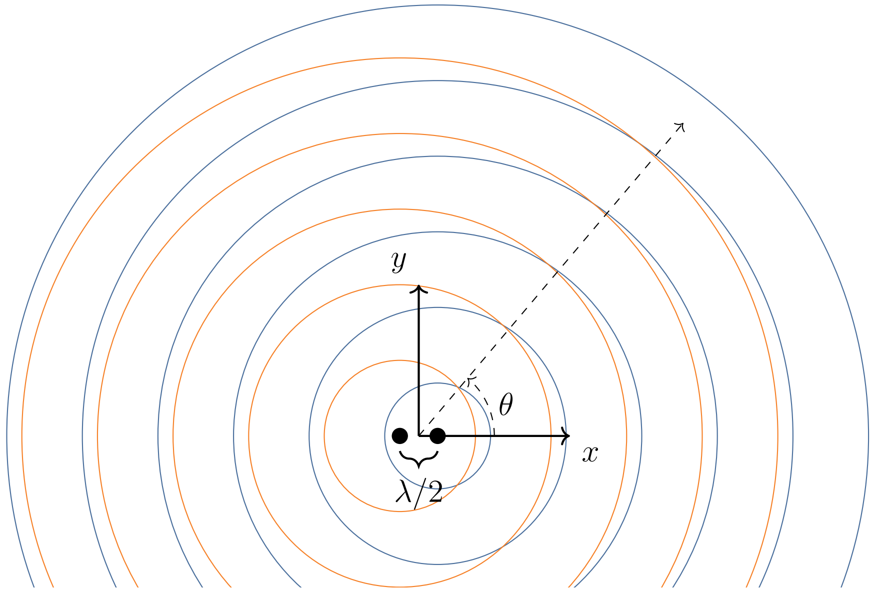


Figure 2.3: Two antennas one half-wavelength apart driven with a 108° phase difference

In order to reliably control the direction of maximum power transfer, we can examine the layout of a generic two element array with spacing d and determine what value of ϕ is necessary to transmit along a given direction θ. Because a phase shift exists as a relative time delay between the two elements, we will assign antenna a in Fig. 2.4 to be the reference element; that is, antenna a will have a phase of $0°$ and antenna b will have a phase of ϕ.

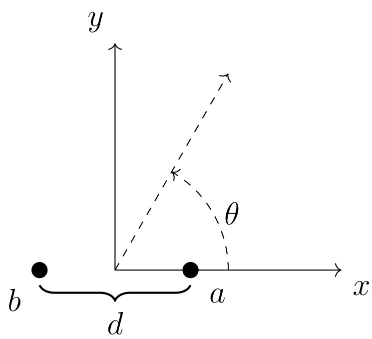


Figure 2.4: Two element antenna array

We can then use trigonometry to determine the amount of phase delay needed for the transmitted signals to add in phase along the direction θ as in Fig. 2.5. The physical distance that the time delay must implement can be calculated as

$$Distance = d \cdot cos(\theta) \tag{2.2}$$

A phase shift of 2π radians corresponds to a distance of 1λ, which allows us to directly calculate the phase shift ϕ needed to transmit along θ if the antenna spacing d has been defined in terms of the wavelength λ as:

$$\phi = 2\pi d \cdot cos(\theta) \tag{2.3}$$

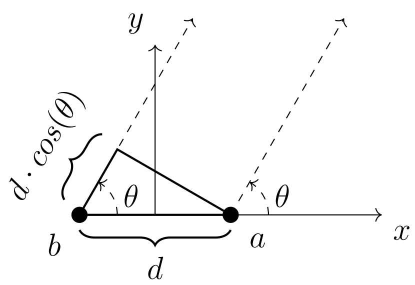


Figure 2.5: Two element antenna array

This analysis can be extended to any equally spaced linear antenna array of N elements as shown in Fig. 2.6. The phase shift ϕ between the n^{th} and $(n-1)^{th}$ element will be the same as ϕ calculated from (2.3). We can now define the relative phase shift between elements of an arbitrarily sized linear array with spacing d as:

$$\phi_n = 2\pi d \cdot n \cdot cos(\theta) \tag{2.4}$$

$$\phi_{TX} = 2\pi d \cdot cos(\theta_{TX}) \tag{2.5}$$

where ϕ_n is the absolute phase at the n^{th} antenna element, and ϕ_{TX} is the relative phase shift between each antenna element of the array to transmit along θ.

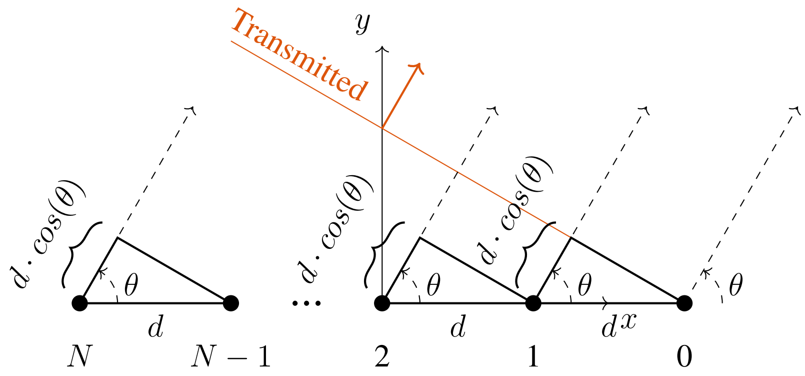


Figure 2.6: N element linearly spaced antenna array transmitting along θ

In addition to providing increased directivity of transmitted signals, electronic beamforming can be used to provide gain of received signals. The analysis so far has focused on the far-field wavefront generated by an antenna array, but the same analysis can apply to a planar wavefront received by an antenna array. Once again, we can calculate the phase shift that will be required at every antenna element by examining the phase delays needed to provide an in-phase signal as in Fig. 2.7.

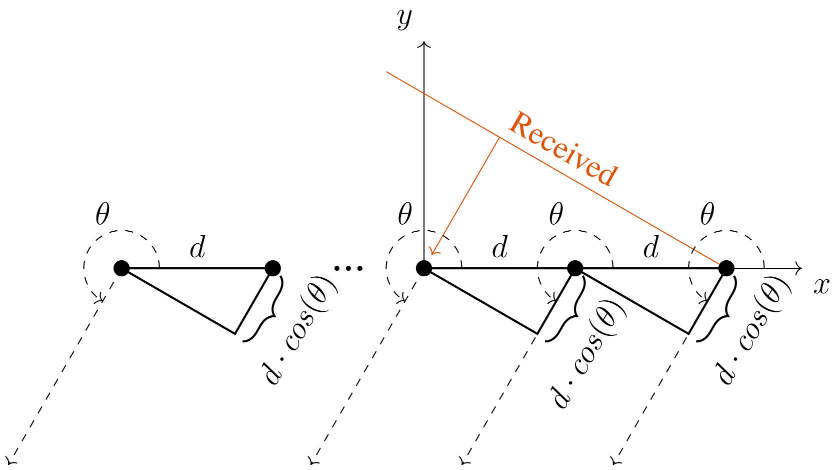


Figure 2.7: N element linearly spaced antenna array receiving along θ

In Fig. 2.7, the direction θ is the direction of propagation of the received wavefront. Using this definiton for θ, we can see that the equations for the relative phase shift between elements of a linear antenna array is:

$$\phi_n = 2\pi d \cdot n \cdot cos(\theta) \tag{2.6}$$

$$\phi_{RX} = 2\pi d \cdot cos(\theta_{RX}) \tag{2.7}$$

We can see that (2.7) and (2.5) are strikingly similar, and (2.6) $=$ (2.4). This is due to the way in which we defined the view angle θ for the received wavefront. If we instead defined the view angle θ for the received signal as the direction the signal propagates *from*, we can rewrite (2.7) as:

$$\phi_{RX} = 2\pi d \cdot cos(\theta + \pi) = -2\pi d \cdot cos(\theta) \tag{2.8}$$

The phase shift from these equations will provide an in-phase signal that can be summed for an electric field strength gain of N or power gain of N^2 [30].

We now see that by placing a phase shifter in front of each element in an antenna array as in Fig. 2.8, we can steer the antenna pattern in any direction we want with a maximum transmit/receive power gain of N^2 for an N-element antenna array. This generalized study of electronic beamforming applies not only to isotropic radiators, but to any antenna array composed of any type of antenna elements. This is due to the principal of *pattern multiplication*, which states that the total antenna array pattern will be the product of the individual antenna pattern and the array pattern, or:

$$P_{total} = P_{antenna} \cdot P_{array} \tag{2.9}$$

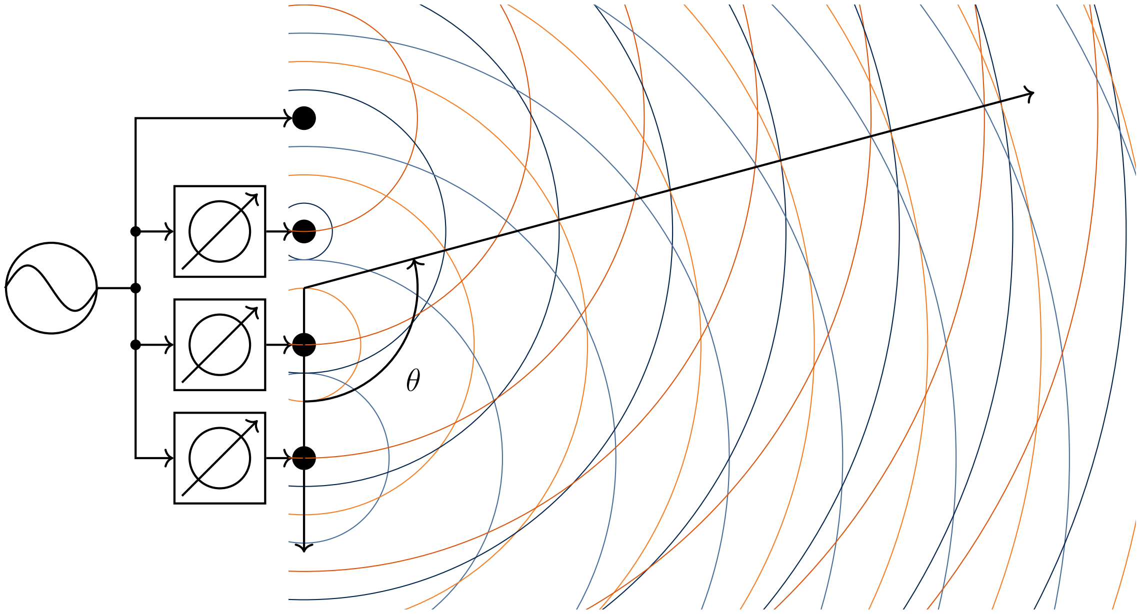


Figure 2.8: A 4-element half-wavelength linear antenna array transmitting along $\theta = 105°$ with $\phi_{TX} = -45°$

A linear antenna array represents the most basic form of antenna array. More complex geometries and drive strategies, such as planar arrays and amplitude tapering, are often implemented in modern systems. However, an understanding of the basics of electronic beamsteering is sufficient to understand this dissertation's novel hybrid retrodirective array and its functionality.

2.2 Proposed Electronic Beamforming Techniques for 5G Networks

Because antenna arrays provide a method to steer transmitted and received signals without any moving parts, they are a critical portion of the Medium Access Control (MAC) layer of wireless communication networks [11]. Currently, there is much debate on what protocols and techniques should be used to establish and maintain a phased array's beamformer point for mmWave 5G systems [14].

IEEE standards 802.11ad/ay are currently in development and aim to provide short-range Wireless Fidelity (WiFi) connections with data rates of 6-20 Gbps using 60 GHz carrier frequencies [11]. The proposed beamformer training methods in these standards rely on a two-part process which must be repeated constantly: a sector-level sweep (SLS) and beam-refinement protocol (BRP). When two transceivers seek to establish a wireless connection, they must begin the process by performing an SLS phase with the other transceiver in a quasi-omnidirectional configuration. Each transceiver is then able to communicate which beamformer sector performed best for the initial sweep. From here, a BRP phase must be performed for each data transmission interval (DTI) in order to further refine the connection. This process is illustrated in Fig. 2.9, and is only intended for use among static users; that is, the protocol does not support tracking of moving targets, as is the goal for mobile 5G systems [11, 16].

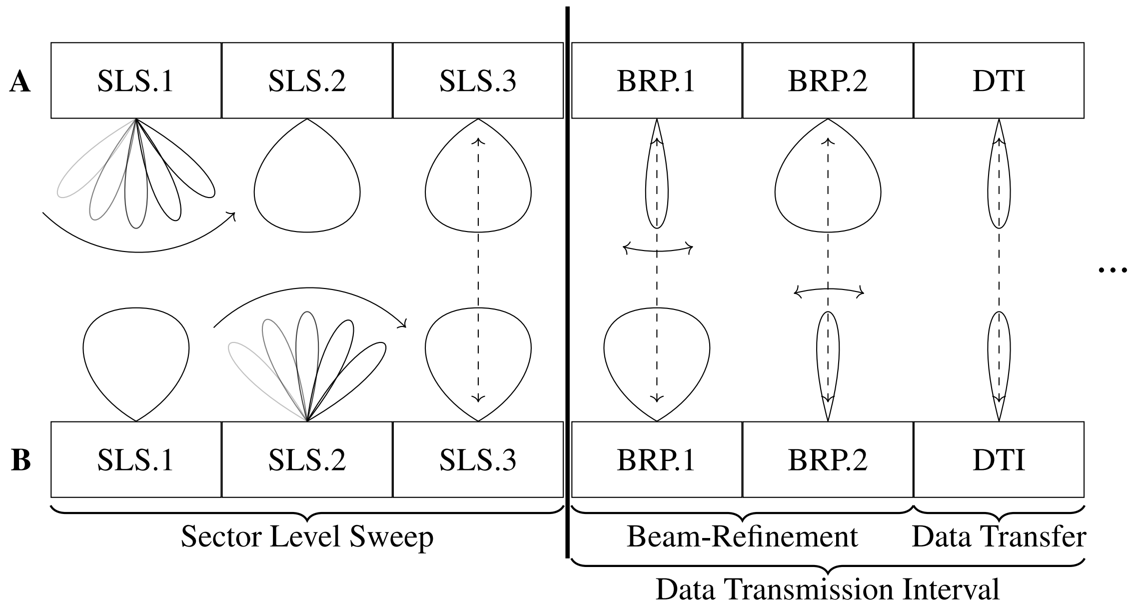


Figure 2.9: Illustration of beamformer training protocol for IEEE 802.11ad/ay

In addition to these new mmWave WiFi standards being developed, researchers have proposed several possible initial beamformer acquisition schemes for mobile 5G networks [14]. One possible solution is to simply perform an exhaustive search of the entire beamformer space and select the beamformer point which had the highest received signal strength indicator (RSSI) [31]. A more complex but potentially quicker scheme involves an iterative beamformer search process whereby large sectors are illuminated first before narrow refined beams within the chosen sector are selected with a binary search algorithm [32]; this process is similar in nature to the IEEE802.11ad/ay protocol shown in Fig. 2.9. Yet more complex is a proposed scheme to transmit GPS coordinates of all nearby stations through a coexistent 4G LTE band to allow for context-aware beamformer acquisition [33]; however, this technique has the added latency and energy cost of acquiring and communicating GPS coordinates through a coexistent network.

All of the methods discussed so far have only focused on the initial beamformer acquisition process and are mostly unsuitable to track a mobile target. Timing simulations of these initial acquisition techniques performed in [14] found a best-case acquisition time of around 10ms from a very close range for the context-aware protocol proposed in [33], but an average time of 200 ms was needed for mid-range beamformer acquisition at 95m; the acquisition times for an exhaustive search and iterative search ranged from 20 ms to 700 ms and 45 ms to 1800 ms respectively. These protocols provide no way to reevaluate the beamformer point to actively track moving targets until the communication link degrades by a significant point [11], and are therefore unsuited for dynamic mobile communication environments. Due to the high path-loss of mmWave frequencies, micro-cells with ranges on this order of magnitude will be necessary [16].

There is a strong need for new beamforming techniques and protocols, with this need identified as one of the final hurdles for mmWave 5G mobile networks [9]. In order to reduce the initial acquisition time, provide mobile target tracking, and eliminate the communication overhead of other proposed methods, this dissertation proposes a novel hybrid retrodirective array for wireless communications.

Chapter 3

Retrodirective Arrays

3.1 Theory of Retrodirection

Retrodirection is the transmission of a signal back along the direction of arrival towards the received signal's source. This term arises from *retroreflection*, which occurs when a device reflects the majority of incident light back to its source with minimum scattering; examples of this phenomenon can be seen in bicycle reflectors, running shoes, and even cat eyes as shown in Fig. 3.1. To understand retroreflection is to understand retrodirection, as visible light is simply one form of electromagnetic radiation: just like the RF waves used in wireless communications.

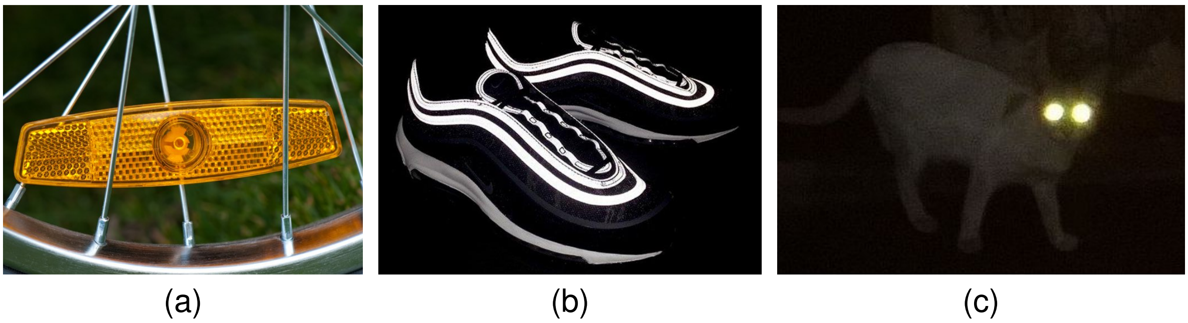


Figure 3.1: Retroreflectors: (a) bicycle reflector [34] (b) running shoe [35] (c) cat's eye [36]

Retroreflection is typically achieved through a geometric structure, such as the corner reflector shown in Figure 3.2. In this structure, the light reflects off of the walls of the structure according to the law of reflection such that $\theta_i = \theta_r$ for an incident wave [37]. As we can see from Figure 3.2b the angle of transmission θ_{TX} is opposite of the angle of arrival θ_{RX}, or simply $\theta_{RX} + 180°$, by the time the incident wavefront leaves the corner reflector.

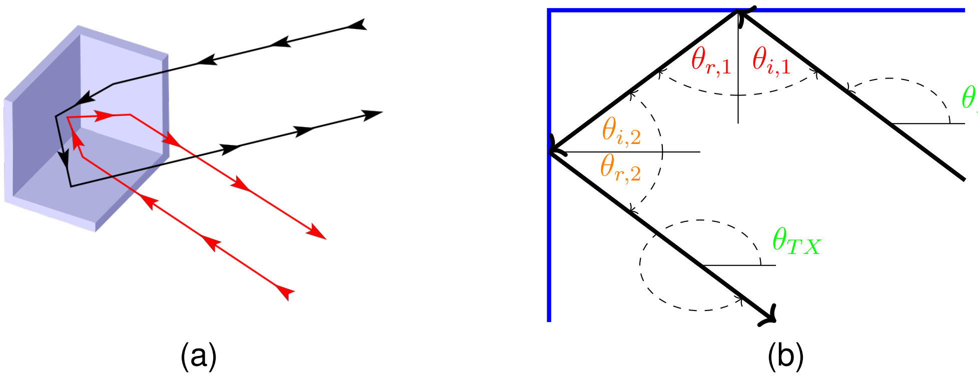


Figure 3.2: (a) A simple corner reflector structure [38] (b) 2-Dimensional example of retroreflection occurring in a corner reflector

Retroreflection can also be achieved through the use of a *phase conjugator*: a device that transmits/reflects the conjugate of the received signal such that $TX = RX^*$. A simpler way of writing this in terms of waves is that the phase must swap signs, or $\phi_{TX} = -\phi_{RX}$. In the case of optics, this can get very complicated and is beyond the scope of this dissertation. However, conjugation of RF signals can be very easy to understand by examining a simple antenna array of elements A and B with distance d between them, as in Figure 3.3. Here is where we will also begin discussing retro*direction* instead of retro*reflection*; the only difference being that retrodirection is what we call it when we generate the transmitted signal as opposed to reflecting a received signal. In general, an antenna array will absorb the received signal, perform some form of processing on it, and generate the transmitted signal separately. Because the antenna array is generating rather than reflecting, this is called retrodirection and not retroreflection.

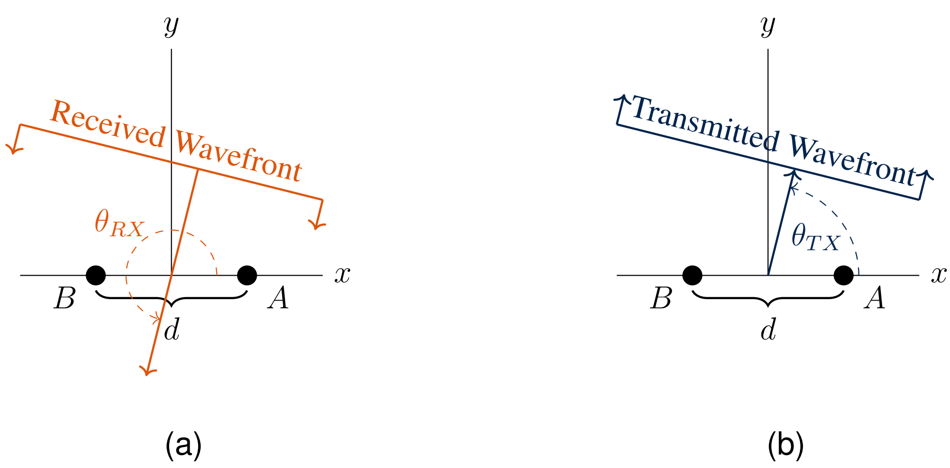


Figure 3.3: A simple two-element retrodirective array array (a) receiving an incident wavefront (b) transmitting a conjugated wavefront back along the direction of arrival

As discussed in Chapter 2, the relative phase ϕ between the elements of an antenna array are of particular interest. If we determine antenna A to be the reference element and define d in terms of wavelengths λ, we can calculate the received signal phase ϕ_{RX} at antenna B as follows:

$$\phi_{RX} = 2\pi d \cdot cos(\theta_{RX}) \tag{3.1}$$

This equation should look very similar to (2.5), with the only difference being the application of determining the received signal phase instead of choosing the transmitted signal phase.

In order to perform retrodirection, we will use (3.1) in conjunction with (2.5) and choose to transmit back along the direction of arrival; that is, we set $\theta_{TX} = \theta_{RX} + 180°$ or $\theta_{TX} = \theta_{RX} + \pi$:

$$\begin{aligned}
\phi_{TX} &= 2\pi d \cdot cos(\theta_{TX}) && (2.5)\\
\phi_{RX} &= 2\pi d \cdot cos(\theta_{RX}) && (3.1)\\
\theta_{TX} &= \theta_{RX} + \pi\\
\phi_{TX} &= 2\pi d \cdot cos(\theta_{RX} + \pi)\\
&= 2\pi d \cdot \Big(cos(\theta_{RX}) \cdot cos(\pi) - sin(\theta_{RX}) \cdot sin(\pi)\Big)\\
&= 2\pi d \cdot \Big(cos(\theta_{RX})(-1) - sin(\theta_{RX})(0)\Big)\\
&= -1 \cdot \Big(2\pi d \cdot cos(\theta_{RX})\Big)\\
\phi_{TX} &= -\phi_{RX} && (3.2)
\end{aligned}$$

Here we have arrived at (3.2): the criteria of *Phase Conjugation* which dictates that for an antenna array to retrodirect, or transmit back along the direction of arrival of a received signal, the relative phase of the transmitted signal must be the conjugate of the relative phase of the received signal.

An interesting property of (3.2) is that there is no dependency on the physical properties of the antenna array or direction of arrival. This means that an ideal phase conjugator can allow arbitrarily sized and shaped antenna arrays to retrodirect if conjugation is performed relative to the same element, as this can be thought of as the superposition of multiple two-element retrodirective arrays. Furthermore, the geometry of the antenna array itself need not remain constant: as long as phase conjugation is performed with respect to the same reference, the antenna array can change its layout without affecting retrodirective performance.

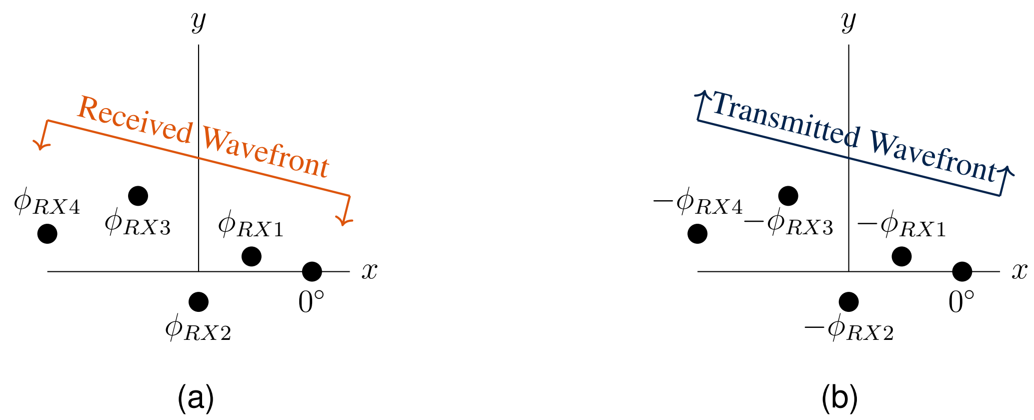


Figure 3.4: A multi-element retrodirective array (a) receiving an incident wavefront (b) transmitting a conjugated wavefront back along the direction of arrival

3.2 Retrodirective Array Properties

A *Retrodirective Array* is an antenna array driven by phase conjugators. When operating correctly, retrodirective arrays demonstrate several unique properties that are very advantageous to wireless communication systems: automatic lock-on, automatic mobile target tracking, and a potential for variable array structures. However, they also present a crucial flaw that has kept them from being adopted in large-scale communication systems: an inability to function when not receiving a signal.

3.2.1 Automatic Lock-On

Retrodirective arrays are capable of locking onto a received signal and performing retrodirection without *a priori* knowledge of the direction of arrival. This is due to the nature of retrodirection - the transmitted signal is constructed from the received signal. The constructed signal is the conjugate of the received signal and travels directly back along the direction of arrival, meaning that an ideal beamformer point to the other transceiver is established automatically. This property of retrodirective arrays means that there is no need for sector sweeping, beamformer training pattern communication, or any overhead in the initial establishment of a communication link as in current standard protocols [11, 14]. As soon as a signal is received, it can be conjugated and a retrodirective link can be established.

3.2.2 Automatic Mobile Target Tracking

The passive nature of a retrodirective array's beamforming allows for automatic tracking of mobile targets. The phase conjugators that make up a retrodirective array must operate continuously and change the transmitted signal phases in accordance with the changes in the received signal phases according to (3.2). This allows for a moving target to be tracked by a retrodirective array in a protocol transparent manner: that is, without the need for any communication between the two moving transceivers. This difference is illustrated in Fig. 3.5.

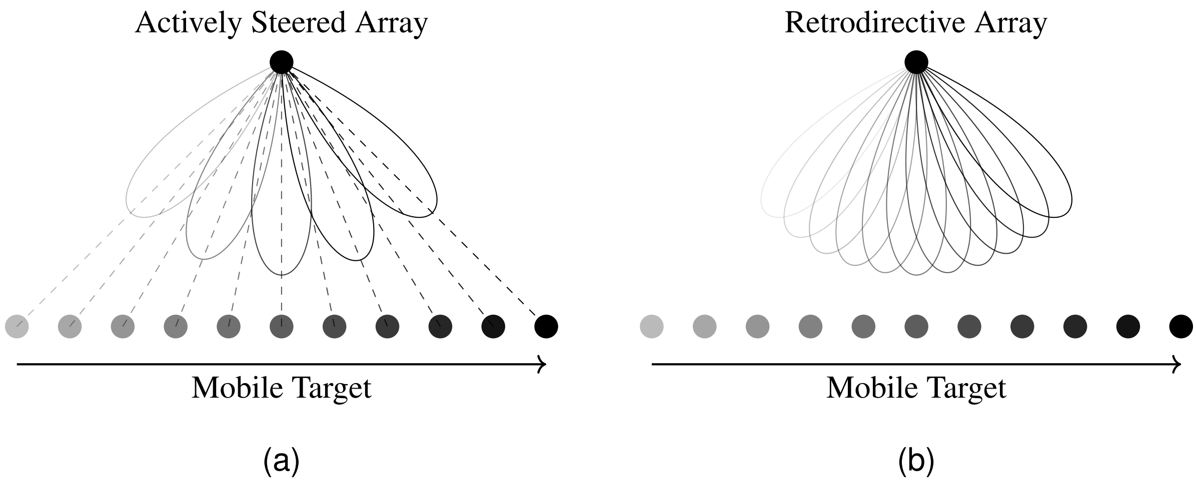


Figure 3.5: (a) An actively steered phased array must continually reevaluate its current beamformer point and receive information from a mobile transceiver in order to maintain a link (b) A retrodirective array automatically performs ideal beamforming at every instant in time without any communication overhead or calculation

3.2.3 Variable Array Geometries

As discussed in Section 3.1, retrodirection does not depend on the physical layout of the antenna array as long as phase conjugation is performed relative to the same antenna element. Because of this, the array geometry of an antenna array does not need to remain constant during operation: that is, the shape of the antenna array can change over time. As phase conjugation is performed continuously, any change in the received signal phases will be reflected in the transmitted conjugated signal as well at that instant in time. As long as both the transmit and receive arrays have the same geometry, accurate retrodirection will still be performed through phase conjugation.

As a proof for this, we can show that the received and transmitted signal phases will be accurately tracked by a two-element retrodirective array for any change in separation $\mathrm{d}d$ or received signal arrival angle $\mathrm{d}\theta_{RX}$ by making sure that phase conjugation is still being performed:

$$\phi_{TX} = 2\pi d \cdot cos(\theta_{TX}) \tag{2.5}$$

$$\phi_{RX} = 2\pi d \cdot cos(\theta_{RX}) \tag{3.1}$$

$$\phi_{TX} = -\phi_{RX} \tag{3.2}$$

$$\theta_{TX} = \theta_{RX} + \pi$$

$$\frac{\mathrm{d}\phi_{RX}}{\mathrm{d}d} = 2\pi \cdot cos(\theta_{RX}) \tag{3.3}$$

$$\frac{\mathrm{d}\phi_{TX}}{\mathrm{d}d} = 2\pi \cdot cos(\theta_{RX} + \pi)$$

$$\frac{\mathrm{d}\phi_{TX}}{\mathrm{d}d} = -2\pi \cdot cos(\theta_{RX}) \tag{3.4}$$

$$\frac{\mathrm{d}\phi_{TX}}{\mathrm{d}d} \stackrel{\checkmark}{=} -\frac{\mathrm{d}\phi_{RX}}{\mathrm{d}d}$$

$$\frac{\mathrm{d}\phi_{RX}}{\mathrm{d}\theta_{RX}} = -2\pi d \cdot sin(\theta_{RX}) \tag{3.5}$$

$$\frac{\mathrm{d}\phi_{TX}}{\mathrm{d}\theta_{RX}} = -2\pi d \cdot sin(\theta_{RX} + \pi)$$

$$\frac{\mathrm{d}\phi_{TX}}{\mathrm{d}\theta_{RX}} = 2\pi d \cdot sin(\theta_{RX}) \tag{3.6}$$

$$\frac{\mathrm{d}\phi_{TX}}{\mathrm{d}\theta_{RX}} \stackrel{\checkmark}{=} -\frac{\mathrm{d}\phi_{RX}}{\mathrm{d}\theta_{RX}}$$

This combination of proofs shows that the antenna array geometry of any retrodirective array may change at any point without affecting retrodirection. This is because the retrodirective array can be treated as a superposition of many two-element antenna arrays fully defined by their spacing d and angle of arrival θ_{RX}, as above. Any possible manipulation of the array geometry can be described as a scalar multiplication of the spacing d and rotation of the angle of arrival θ_{RX} for each individual two-element array, as in Fig. 3.6.

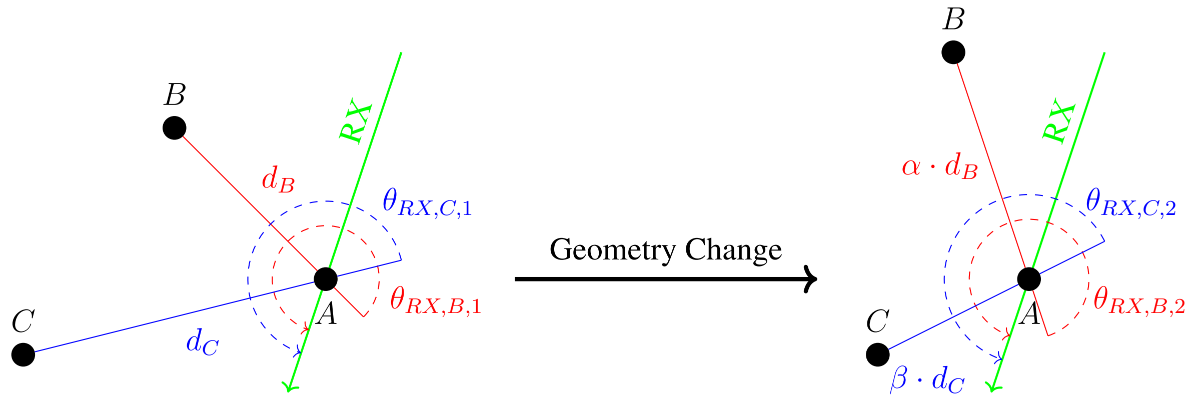


Figure 3.6: Any geometry change of a multi-element array can be described as a stretch and rotation of the constituent two-element arrays

3.2.4 Received Signal Sensitivity

A phase conjugator can only function if there is a signal to conjugate. While this may seem obvious, it poses one of the main obstacles in implementing a retrodirective array based communication system: a retrodirective array can only function as long as it is receiving a signal as illustrated in Fig. 3.7. This means that any communication scheme which cannot accommodate simultaneous transmission and reception, such as a time-division duplex (TDD) or handshaking scheme, cannot be directly implemented using retrodirective arrays. However, the use of an ever-present pilot tone to retrodirect provides a workaround at the cost of an extra signal [39]. Additionally, a retrodirective array cannot function in an environment with many blockers or obstructions to communications, as any momentary drop in the received signal will cause system decoherence and a failure of the communication link [27].

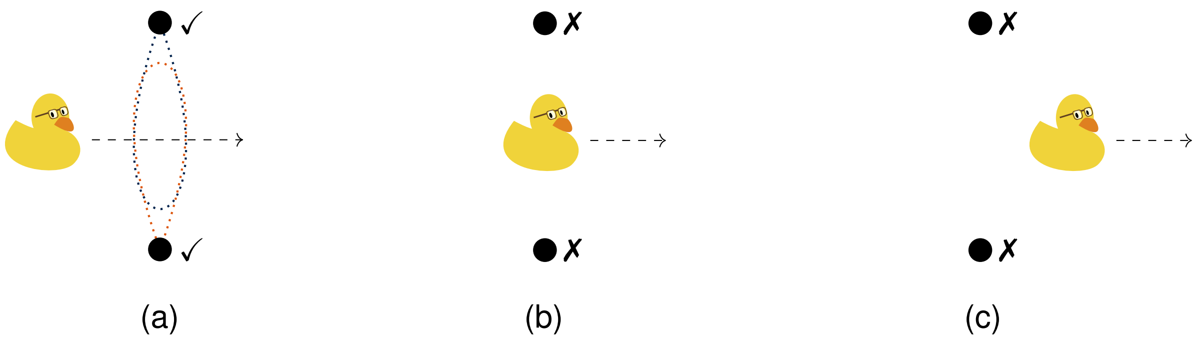


Figure 3.7: Two retrodirective arrays become decoherent due to a mobile obstacle

3.3 Phase Conjugation Methods & Circuits

The essential building block of a retrodirective antenna array is an electronic phase conjugator. Here we examine the different methods for constructing a phase conjugator and their advantages and disadvantages.

3.3.1 The Van Atta Reflector Array

The earliest form of the retrodirective antenna array was invented by Dr. Van Atta in 1960 [22]. This structure consists of a linearly spaced antenna array with transmission lines of equal length L connecting opposite antenna elements [22, 40], as shown in Figure 3.8.

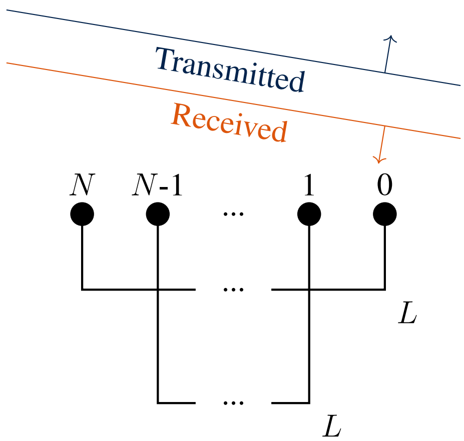


Figure 3.8: Linear Van Atta Reflector Array [22]

To understand the Van Atta reflector array's performance as a phase conjugator, we can examine the relative phases between elements upon reception and transmission. We can define the relative phase $\Delta\phi$ at antenna element n relative to the 0^{th} element as:

$$\Delta\phi_n = \phi_n - \phi_0 \tag{3.7}$$

Using (2.7) for the received signal phase of a linearly spaced antenna array, we can calculate the relative phase of the received signal for the n^{th} antenna element as:

$$\Delta\phi_{n,RX} = 2\pi(n \cdot d) \cdot cos(\theta_{RX}) - 2\pi(0 \cdot d) \cdot cos(\theta_{RX})$$
$$\Delta\phi_{n,RX} = 2\pi(n \cdot d) \cdot cos(\theta_{RX}) \tag{3.8}$$

The equal length transmission lines between opposite elements of the antenna array will add $2\pi L$ radians of phase delay to the received signal before transmission if L is given in terms of wavelengths, allowing us to define the transmitted signal phase as:

$$\phi_{n,TX} = \phi_{N-n,RX} + 2\pi L \tag{3.9}$$

Using (3.9) and (3.7), we can now define the relative phase of the transmitted signal as:

$$\begin{aligned}\Delta\phi_{n,TX} &= \phi_{n,TX} - \phi_{0,TX} \\ &= \Big(2\pi(N-n)d \cdot cos(\theta_{RX}) + 2\pi L\Big) - \Big(2\pi(N-0)d \cdot cos(\theta_{RX}) + 2\pi L\Big) \\ &= 2\pi(N-n-N)d \cdot cos(\theta_{RX}) + (2\pi L - 2\pi L) \\ \Delta\phi_{n,TX} &= -2\pi(n \cdot d) \cdot cos(\theta_{RX})\end{aligned} \tag{3.10}$$

Which allows us to see that phase conjugation is being performed:

$$\Delta\phi_{TX} = -\Delta\phi_{RX} \tag{3.2}$$
$$-2\pi(n \cdot d) \cdot cos(\theta_{RX}) \stackrel{\checkmark}{=} -\Big(2\pi(n \cdot d) \cdot cos(\theta_{RX})\Big)$$

The Van Atta array was originally designed as a retroreflective array, as it only retransmits the received signals and performs no amplification or modulation [22, 40]. However, the Van Atta array was improved upon and expanded over several years. Researchers were able to create multi-dimensional Van Atta arrays as shown in Fig. 3.9, such as planar [40, 41], circular [42], cylindrical [41], and even spherical [41]. Methods were also developed for creating retrodirective Van Atta arrays by inserting active components to amplify, frequency shift, and modulate the signal [40–43]. The Van Atta array was typically used for radar beacons and range finding applications and was even proposed for satellite transponders [40–42]. However, the Van Atta array's performance depended entirely on the physical layout of the antenna array as well as complex back-end electronics to perform any active functions [42]. This dependency on array geometry can be seen from the fact that our mathematical derivation depended on a linear spacing d of our antenna elements: if this weren't the case, the equations for the received signal phase at each element would not line up with those for the transmitted signal phase and phase conjugation would not be performed.

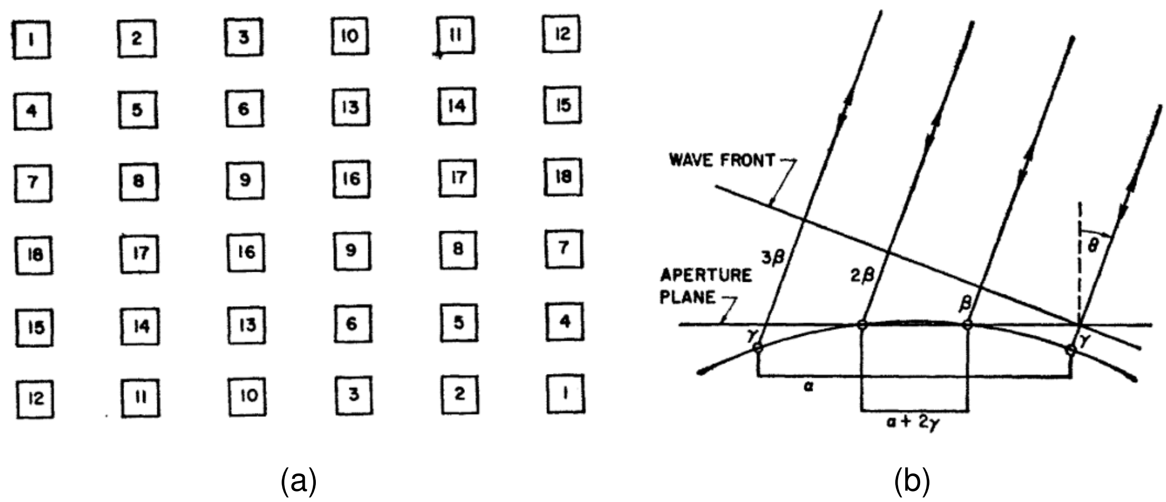


Figure 3.9: Non-linear Van Atta Array geometries in [41] (a) 36-element square array with connections listed (b) cylindrical array

Despite the restrictive geometry requirements for Van Atta arrays, they represent the simplest form of retrodirective array and remain a popular research topic [44–47], although the majority of modern works only examine the Van Atta array's performance with different radiating elements. The term "Van Atta Array" is now used to describe any retrodirective array whose operation is geometry dependent [48].

3.3.2 The Pon Array

The next innovation in retrodirective arrays was the Pon Array in 1964, which relied on heterodyne mixers to create a geometry-independent phase conjugator [23]. A heterodyne mixer multiplies two input signals to produce a single output signal. When two sinusoidal input signals are mixed, the output consists of two components at the sum and difference of their frequency components:

$$sin(2\pi f_1 t) \cdot sin(2\pi f_2 t) = \frac{1}{2}\left(cos\Big(2\pi(f_1 - f_2)t\Big) - cos\Big(2\pi(f_1 + f_2)t\Big)\right) \quad (3.11)$$

Pon found that if a local oscillator (LO) at twice the RF frequency was mixed with the received signal, the difference term would be the conjugate of the received signal:

$$sin(2\pi f_{RF} t + \phi_{RF}) \cdot sin(2\pi 2 f_{RF} t) = \frac{1}{2}\Big(cos(2\pi f_{RF} t - \phi_{RF}) - cos(2\pi 3 f_{RX} t + \phi_{RF})\Big) \quad (3.12)$$

From here amplification, filtering, and circulators can be used to remove the $3f_{RF}$ frequency component and transmit the conjugated f_{RF} frequency component to perform retrodirection. Pon's array structure is shown in Fig. 3.10.

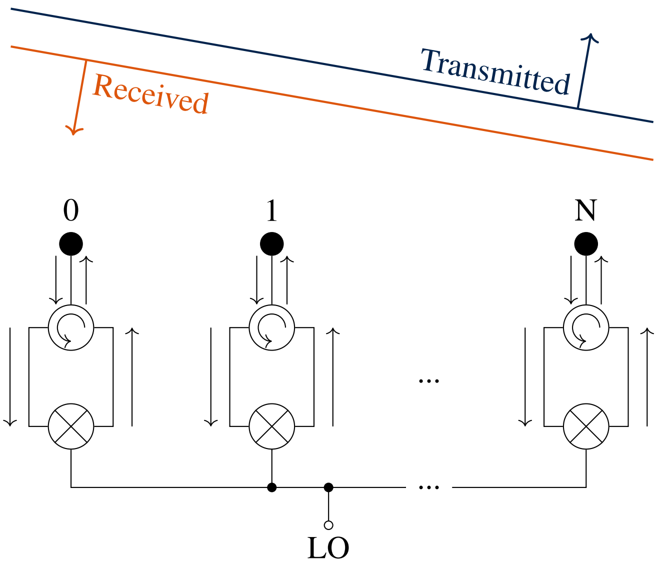


Figure 3.10: Pon retrodirective array

The Pon array removes any dependency on the antenna array geometry by performing phase conjugation of the received signal directly at the individual element. By having identical hardware at each element (circulators, mixers, transmission lines, etc.) and using the same LO signal for every mixer, any phase delay introduced by the hardware will be the same at every antenna element. This means that the relative phase differences between each element at transmission depend only on the relative phase differences between each element at reception of the signal.

A "Pon Array" is now used to describe any retrodirective array that uses a heterodyne mixer as the main method for phase conjugation [48]. Many variations of Pon arrays have been investigated, with incremental improvements in performance gained through slightly different system architectures or advances in components [49, 50]. However, the Pon array did not gain significant research interest for many years due to the necessity of a LO at twice the RF frequency, the losses associated with mixers, and the complexities of the per-element hardware [49, 51]. Additionally, Pon arrays have no way to provide received signal beamforming [52] or to track phase modulated signals [39].

3.3.3 Phase-Locked Loop (PLL) Arrays

The most diverse and functional class of retrodirective arrays is based on the phase-locked loop (PLL) phase conjugator, first introduced by Buchanan in 2004 [24]. The PLL conjugator seeks to provide a constant transmit power regardless of the received power level, which was a major shortcoming of the Pon array [51]. The most basic version of a PLL phase conjugator is outlined in Fig. 3.11, an implementation in which the receive and transmit frequencies can be chosen independently. A main advantage of this phase conjugator architecture is the generation of a pure sinusoidal signal at the transmission frequency: this allows for arbitrary modulation of the transmitted signal and makes arbitrary data transmission possible with a retrodirective array [52].

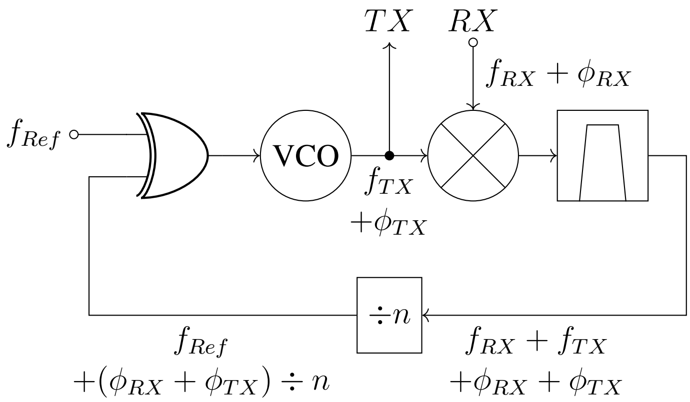


Figure 3.11: A basic PLL-based phase conjugator from [24]

To understand how the system in Fig. 3.11 functions as an element-wise phase conjugator, we will look at how the transmitted signal phase varies with respect to the received signal phase. The combination of phase detector and voltage controlled oscillator (VCO) acts as an integer-n PLL in this system, which means that in steady-state operation the two inputs to the phase detector will have a fixed phase difference between them (90° in the case of an XOR phase detector as shown). Therefore, any changes in the ϕ_{TX} and ϕ_{RX} terms must cancel themselves out, or:

$$\Delta\phi_{TX} = -\Delta\phi_{RX} \tag{3.2}$$

Because the output of the VCO is being used as a transmission source, the output power of the phase conjugator is not dependent on the input power [24, 39, 51]. It is important to note that if the transmit and receive frequencies are chosen to be different, separate receive and transmit antenna arrays must be used in this architecture to provide true retrodirection, as the criteria of phase conjugation for retrodirection depends on the transmit and receive antenna arrays having the same geometries with respect to their wavelength [24].

The basic PLL phase conjugator design was next improved upon to create cleaner transmitted carrier signals. In [51, 52], the received signal is first downconverted to a low-frequency IF signal. This IF signal still retains the received signal phase and is then buffered by a tracking PLL. The PLL output has a constant power level regardless of the received signal strength and the same phase as the received signal, which can be used to provide a phase conjugated transmission carrier when it is upconverted to the transmit frequency by a high-side LO. This approach allows for a cleaner transmitted signal by directly transmitting the local oscillator and PLL output, rather than splitting them within the feedback loop as in Fig. 3.11. This implementation represents a combination of the Pon heterodyning technique with a PLL.

All of the retrodirective array structures presented so far have been designed for use as transponders and have provided no guaranteed in-phase received signal to be summed for beamformer gain [29]. Therefore, the next major advance in PLL phase conjugator research efforts was to perform receive signal beamforming in addition to retrodirection. In [39], the low-frequency IF signal is tracked with a PLL using a reference clock at $4 \cdot f_{IF}$, which allows for received signal beamforming to be performed on modulation schemes such as QPSK and 256QAM. Furthermore, [25] developed a method for automatic received signal beamforming of arbitrary modulation schemes by using the IF signal of the chosen 0^{th} antenna as the PLL reference for IF signals from other antenna elements. This provides an in-phase IF signal to be summed for beamforming gain regardless of the modulation scheme while still providing a purely sinusoidal phase conjugated signal for transmitted signal modulation. A simplified version of the system in [25] is shown in Fig. 3.12.

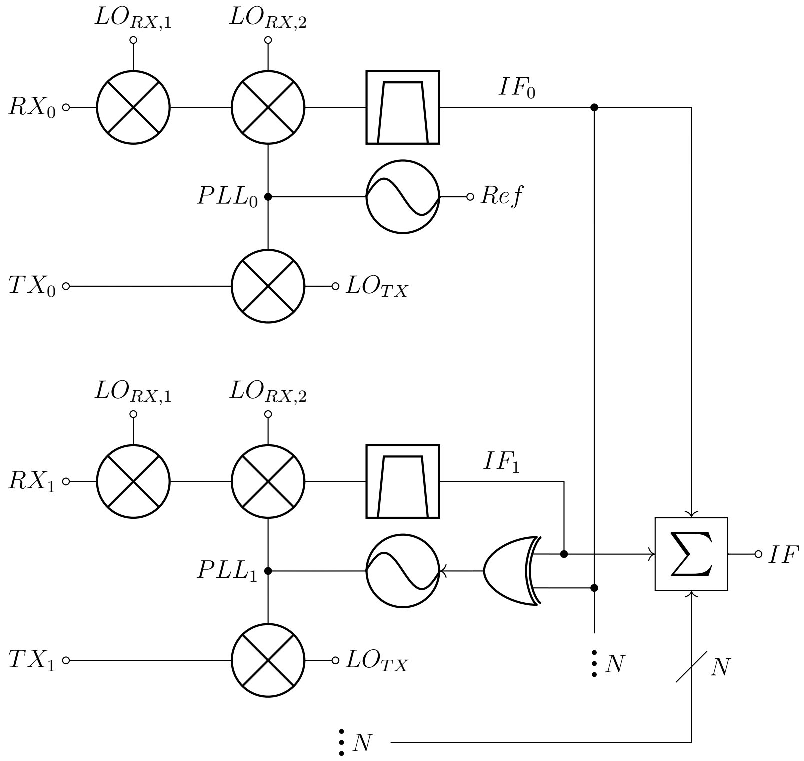


Figure 3.12: Simplified PLL phase conjugator/beamformer from [25]

3.3.4 The Hybrid Retrodirective Array

All of the retrodirective arrays discussed so far suffer from a dependency on receiving a signal in order to operate [48]. However, this is a fundamental flaw of a phase conjugator: there can be no valid output when there is no valid input. Instead, it would be useful to construct a *hybrid phase conjugator* that can function as either a phase conjugator or active phase shifter depending on the context. This can be achieved by including a digital element in the system which can switch between the two modes of operation. With such a hybrid phase conjugator, a *hybrid retrodirective array* could be constructed that is capable of performing context-dependent retrodirection or active beamforming.

Currently, only one hybrid retrodirective array design has been published in the literature [27] and was first demonstrated in 2018. Fig. 3.13 shows a simplified version of their hybrid phase conjugator, which is based upon the system in Fig. 3.12 [25, 27]. The system consists of a delay-locked loop (DLL) with a microcontroller in the feedback path which monitors the phase-detector output and directly sets the phase-delay control voltage. This scheme allows the microcontroller to maintain the last known valid phase delay value if the received signal drops out, thus maintaining phase conjugation along the last valid direction of arrival.

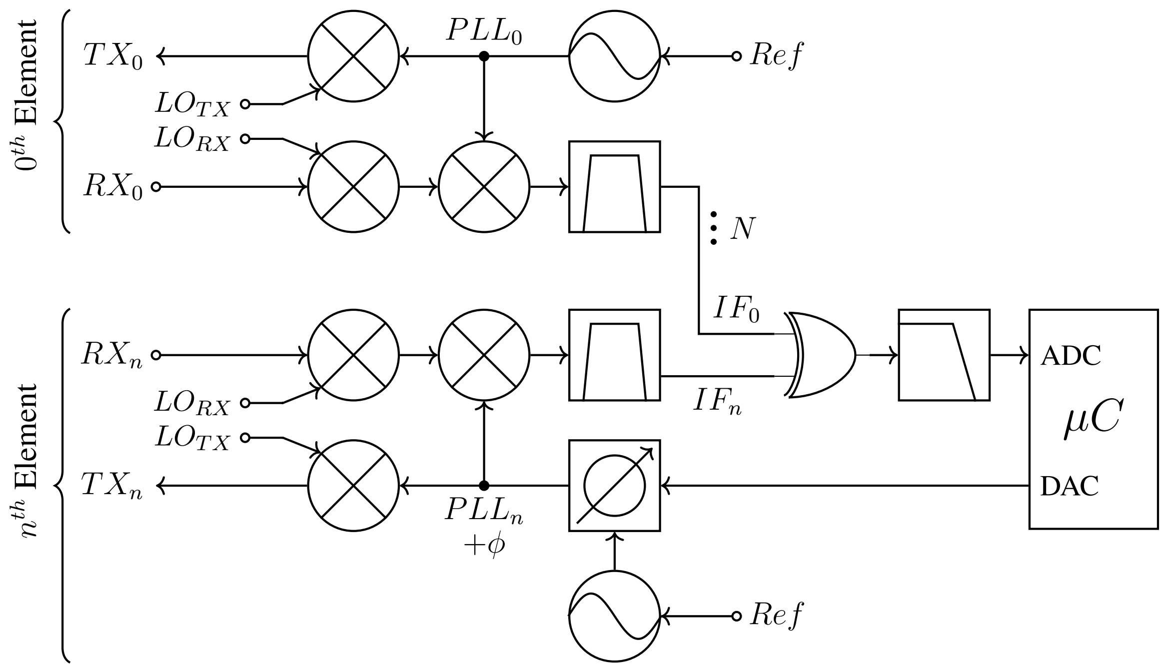


Figure 3.13: Simplified DLL Hybrid Phase Conjugator Design in [27]

The DLL hybrid phase conjugator in Fig. 3.13 performs phase conjugation through clever selection of LO and PLL frequencies. If we choose LO_{RX} to be at a higher frequency than our RX signal and the PLL frequency to be higher than the first downconverting mixer's output, then we can define the IF_n signal's frequency and phase as:

$$f_{IF_n} = f_{PLL} - (f_{LO_{RX}} - f_{RX})$$
$$f_{IF_n} = f_{RX} - f_{LO_{RX}} + f_{PLL} \tag{3.13}$$
$$\phi_n = \phi_{RX} - \phi_{LO} + \phi_{\mu C} \tag{3.14}$$

Here $\phi_{\mu C}$ is the phase delay induced by the microcontroller controlled phase delay block; as the 0^{th} element does not contain this block, its value is 0 for this element. Because all elements use the same LO_{RX} and LO_{TX} signals, we can define the relative phase difference between the IF_n and IF_0 signals as:

$$\Delta\phi_n = (\phi_{\mu C,n} - 0°) - (\phi_{LO} - \phi_{LO}) + (\phi_{RX,n} - \phi_{RX,0})$$
$$\Delta\phi_n = \phi_{\mu C,n} + \Delta\phi_{RX} \tag{3.15}$$

Much like the PLL phase conjugators discussed in Section 3.3.3, the steady state value of this phase difference will be constant in a DLL [53]. This means that the sum of $\phi_{\mu C,n}$ and $\Delta\phi_{RX}$ must be 0, or phase conjugation occurs as:

$$\Delta\phi_{\mu C,n} = -\Delta\phi_{RX,n} \tag{3.16}$$

Furthermore, as the transmitted signal's relative phase will be equal to $\phi_{\mu C,n}$, phase conjugation and retrodirection is performed as

$$\Delta\phi_{TX,n} = \Delta\phi_{\mu C,n} = -\Delta\phi_{RX,n} \tag{3.17}$$

This phase conjugator architecture is deemed a *hybrid* phase conjugator because it is able to actively control the transmitted signal phase and IF signal phase through $\phi_{\mu C,n}$. This allows the direction of transmitted and received beamforming to be chosen directly by the microcontroller if the situation calls for it [27]. This hybrid retrodirective array and actively steered array functionality represents the current state-of-the-art in the field of phase conjugators and retrodirective arrays.

3.4 Hybrid Retrodirective Array Unique Capabilities

A hybrid retrodirective array still exhibits automatic lock-on, automatic mobile target tracking, and the ability to function properly in variable array geometries as outlined in Section 3.2. Additionally, the presence of a digital control element in the feedback path allows for two more capabilities: received signal dropout immunity and direction of arrival reporting.

3.4.1 Received Signal Dropout Immunity

As already mentioned in Section 3.3.4, a hybrid retrodirective array's main attractive quality is its ability to continue functioning without a received signal. By storing the control voltage directly, the microcontroller in the DLL is able to output the previous value if it detects that there is no received signal as in Fig. 3.14. Further, if the values are stored over time the microcontroller can predictively track a moving target in a dynamic environment with mobile blockers [27]. This opens up the possibility for reliable, fully retrodirective array based networks to be constructed, as the individual systems will be able to tolerate blockers [14], deep channel fades [54], and any communication protocol or modulation scheme [27, 39].

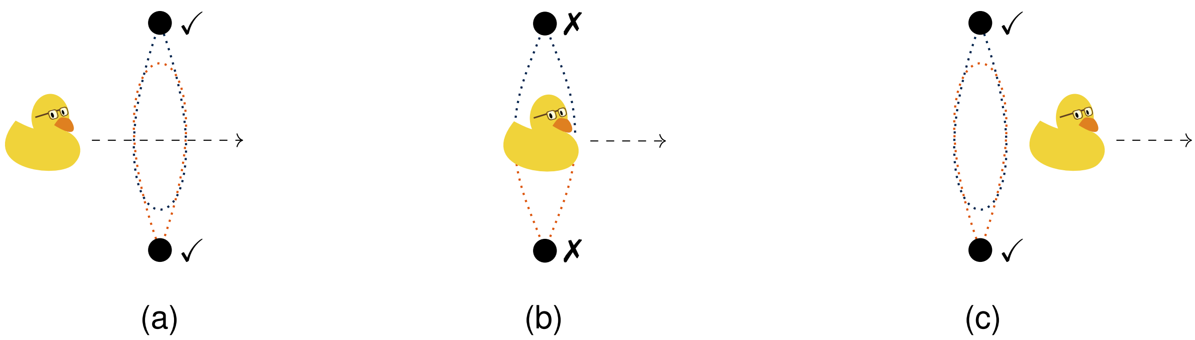


Figure 3.14: Two hybrid retrodirective arrays maintain coherence through a temporary block

3.4.2 Direction of Arrival Reporting

While mixer and PLL retrodirective arrays have no way of reporting the phase of the received signal [23, 49], a hybrid retrodirective array automatically samples this information on a per-element basis. If the array geometry of the system is known, this information can be used to calculate the direction of arrival of a signal. Furthermore, this information can be used by the hybrid retrodirective array to perform environmental analysis. The ability to accurately track mobile targets and have spatial awareness of other mobile users or blockers in real-time can be used to implement cognitive radio techniques, promote frequency reuse, and improve overall network throughput [55]. This capability would enable the use of a spatial division multiple access (SDMA) scheme as in Fig. 3.15.

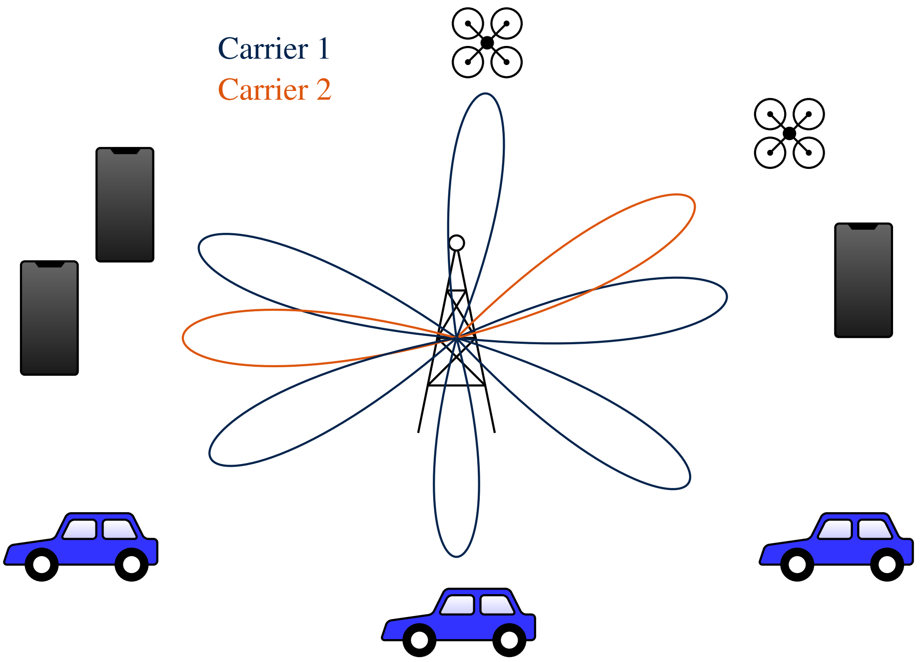


Figure 3.15: A spatial division multiple access (SDMA) scheme reuses frequencies based on user location [55]

Chapter 4

A Novel Hybrid Retrodirective Array

As discussed in Chapter 3, a Hybrid Retrodirective Array (HRA) may provide significant improvements to wireless communication networks. By definition, a HRA must be composed of hybrid phase conjugators (HPCs) - a system capable of operating as either a phase conjugator or an actively controlled phase shifter. The system concept for a DLL-based HPC (DLL-HPC) is discussed in section 4.1, and the receive and transmit signal paths are discussed in detail in Sections 4.2 and 4.3 respectively. Component selection and analog phase shifter design are discussed in Section 4.4, and the final DLL-HPC prototype design is presented in Section 4.5.

4.1 DLL-HPC Concept

The proposed HPC is based on a DLL beamformer that has been augmented with a microcontroller to provide two modes of operation: an open-loop actively steered mode and a closed-loop DLL beamformer mode. The closed-loop DLL beamformer mode allows the microcontroller to monitor and store the received signal phase, while the open-loop actively steered mode allows for active beamforming by choosing the phase shift applied to each signal. Additionally, the transmitted signal beamformer is implemented separately from the received signal beamformer while still being controlled by the same microcontroller. Because both the transmitted signal beamformer and received signal beamformer are controlled by the same microcontroller, two system-level modes of operation become possible:

1. *Phase Conjugator Mode:* The received signal beamformer operates as a closed-loop DLL beamformer. The microcontroller monitors the received signal phase ϕ_{rx} and performs phase conjugation by setting $\phi_{tx} = -\phi_{rx}$ through the transmitted signal beamformer.

2. *Actively Steered Mode:* The received signal beamformer operates as an open-loop actively steered beamformer by applying phase shifts to the received signals chosen by the microcontroller. The transmitted signal beamformer is controlled in a similar way.

Through these main operating modes, the DLL-based HPC (DLL-HPC) provides the retrodirective capabilities outlined in Section 3.2 as well as the additional capabilities outlined in Section 3.4. This DLL-HPC system can be conceptually visualized as in Fig. 4.1.

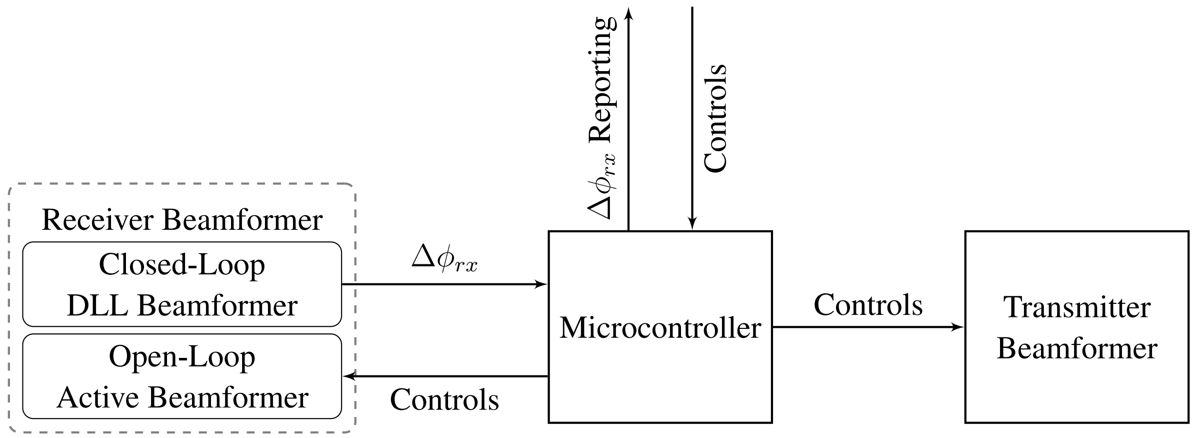


Figure 4.1: Conceptual Hybrid Phase Conjugator System.

4.2 DLL-HPC Received Signal Beamformer

Fig. 4.2 shows the received signal beamformer for the 0^{th} and n^{th} elements of the DLL-HPC. The system consists of a standard RF downconversion chain (amplifier $\rightarrow$ mixer $\rightarrow$ filter) to an intermediate frequency (IF) signal. This is very similar to the HPC system shown in Fig. 3.13 [27], but has two main differences:

1. The TX signal is completely detached from the received signal beamformer, allowing for separate received and transmitted signal beamforming to be performed. This also allows a retrodirective array constructed of the novel DLL-HPCs to be fully backwards compatible with other beamformer protocols [11], and to provide more flexibility in system-level communication schemes.

2. The microcontroller does not block the DLL feedback path. Instead, it uses a SPDT switch to choose whether the beamformer operates in the closed-loop DLL mode or open-loop actively steered mode as conceptualized in Fig. 4.1. This allows the closed-loop mode to take full advantage of the fast settling time and unconditional stability of a DLL [53] while still monitoring the control voltage to determine the current phase delay θ_n. This best illustrates the hybrid analog-digital nature of the system.

.

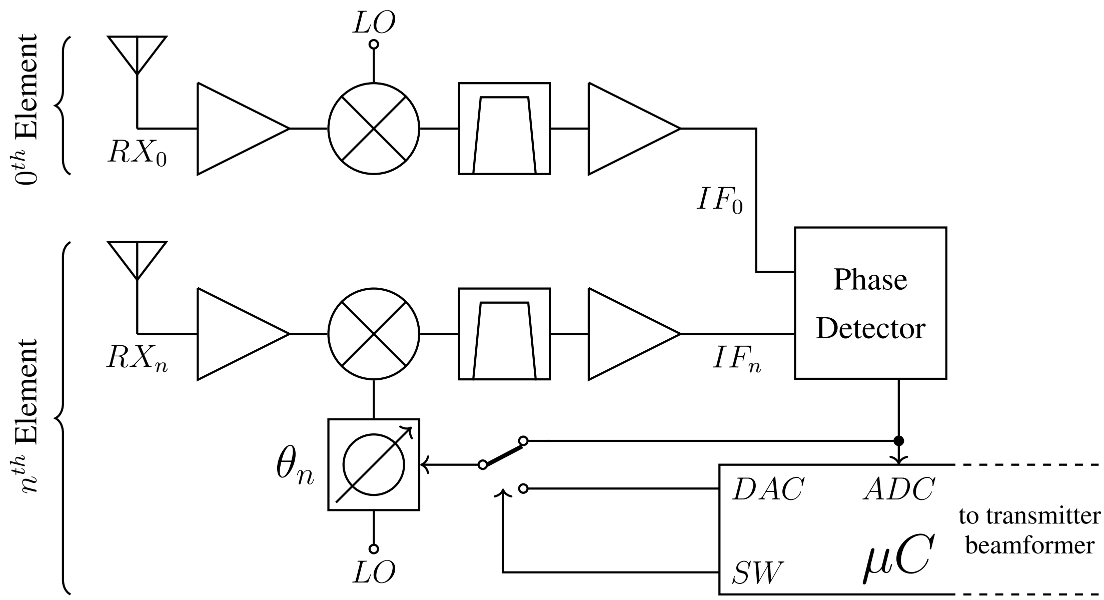


Figure 4.2: DLL-HPC received signal beamformer

If we assume that all local oscillators in the system are derived from the same reference source, then we can say that the DLL-HPC is frequency synchronized [27]. With frequency synchronization, we can define the frequencies and phases of the signals in Fig. 4.2 as shown in Table 4.1.

Signal Name	Element	Frequency	Phase
Received Carrier (RX)	0^{th}	$f_{RX,0}$	$\phi_{RX,0}$
	n^{th}	$f_{RX,n}$	$\phi_{RX,n}$
Local Oscillator (LO)	0^{th}	$f_{LO,0}$	$\phi_{LO,0}$
	n^{th}	$f_{LO,n}$	$\phi_{LO,n}$
Intermediate Frequency (IF)	0^{th}	$f_{IF,0}$	$\phi_{IF,0}$
	n^{th}	$f_{IF,n}$	$\phi_{IF,n}$
Phase Delay (θ)	0^{th}		$0°$
	n^{th}		θ_n

Table 4.1: Frequency and phase definitions for DLL-HPC downconversion chain

By using a high-side LO and an appropriately designed IF filter and amplifier, the IF signal frequency and phases will be:

$$f_{IF} = f_{LO} - f_{RX} \tag{4.1}$$

$$\phi_{IF} = (\phi_{LO} + \theta) - \phi_{RX} \tag{4.2}$$

From (4.2), the phase difference between the nth and reference IF signals can be written as:

$$(\phi_{IF,n} - \phi_{IF,0}) = (\phi_{LO,n} - \phi_{LO,0}) + (\theta_n - \theta_0) - (\phi_{RX,n} - \phi_{RX,0}) \tag{4.3}$$

(4.3) can be simplified by substituting $0°$ for θ_0, as the reference downconversion chain contains no controllable phase delay element. Further, the ϕ_{LO} terms can be removed in favor of a Δi term. This new term encompasses any phase difference between the IF signals that is a product of the physical system: such as signal routing, cable lengths, and phase difference between various LOs. Because all downconversion and carrier generation chains are frequency synchronized, this Δi term is a constant that depends only on the physical system [27]. The final equation for the phase difference between the nth and reference IF signals is now:

$$\Delta\phi_{IF,n} = \theta_n - \Delta\phi_{RX,n} + \Delta i_n \tag{4.4}$$

In steady-state operation the DLL will drive the downconversion chain phase delay θ_n so that the IF signals will be in phase [53], or

$$\begin{aligned} \Delta\phi_{IF,n} = 0^\circ &= \theta_n - \Delta\phi_{RX,n} + \Delta i_n \\ \theta_n &= \Delta\phi_{RX,n} - \Delta i_n \end{aligned} \tag{4.5}$$

In this state the IF signals can be summed to provide maximal beamforming gain, as $\Delta\phi_{IF} = 0^\circ$ [29]. This operation is characteristic of a DLL beamformer [27, 56].

Alternatively, θ_n can be controlled directly by the microcontroller with a digital-to-analog converter through the SPDT switch. In this configuration, active beamforming techniques may be used. As the Δi_n term is a constant determined by the physical system, it can be calibrated out by a fixed offset when performing active beamforming. This allows the microcontroller to have full control of the IF signal phases to be summed through the θ_n phase delay block.

In addition to performing received signal beamforming, the system in Fig. 4.2 allows the microcontroller to measure and record the received signal phase directly. If the LO phase delay block is well characterized, a look-up table (LUT) can be derived that relates the measured control voltage to the induced phase delay θ_n. With this LUT the instantaneous value of θ_n can be found without any calculations [52, 57]. This value of θ_n can then be used to calculate the relative received signal phase $\Delta\phi_{RX,n}$ through a single arithmetic operation to remove Δi_n according to (4.5). This requires the Δi_n term to be known, which can be found by measuring θ_n while applying a known value for $\Delta\phi_{RX}$. A process for determining the relative received signal phase with only a single LUT and subtraction operation is given in Algorithm 1.

With the simple operations outlined in Algorithm 1, the DLL-HPC's received signal beamformer portion is capable of providing automatic, ideal beamformer gain while simultaneously monitoring the received signal's phase. This allows for both the direction of arrival reporting and received signal dropout immunity described in Section 3.4.

Algorithm 1: DLL-HPC received signal phase measurement

Required LUTs:

LUT_θ maps a measured ADC value x to an angle θ: $\theta = \text{LUT}_\theta[x]$

Find Δi_n: *Perform once*

Set $\Delta\phi_{rx,n} = 0°$

Measure $V_{ADC} = x$

Store $\text{LUT}_\theta[x] = \theta = \Delta\phi_{RX,n} - \Delta i_n = -\Delta i_n \to offset$

Measure $\Delta\phi_{RX,n}$: *Perform continually*

Measure $V_{ADC,n} = x$

Store $\text{LUT}_\theta[x] \to \theta_n$

Store $\theta_n - offset = \Delta\phi_{RX,n} - \Delta i_n - (-\Delta i_n) \to \Delta\phi_{RX,n}$

Goto: 1

4.3 DLL-HPC Transmitted Signal Beamformer

With $\Delta\phi_{RX}$ known from the DLL-HPC's received signal beamformer, a simple topology can be chosen for the carrier generation portion. Figure 4.3 shows the transmitted signal beamformer system for the 0^{th} and n^{th} elements.

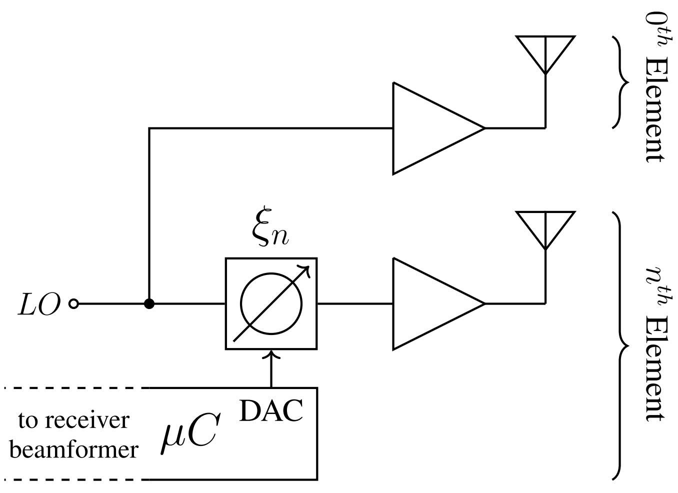


Figure 4.3: DLL-HPC transmitted signal beamformer

The system in Fig. 4.3 needs little explanation due to its simplicity. The same microcontroller that measures and calculates the received signal phase differnece $\Delta\phi_{RX}$ is responsible for controlling the transmitted signal's phase ξ. Once again, a LUT can be developed for a well-characterized phase delay block so that controlling ξ_n takes extremely few resources. Due to frequency synchronization across the system, we can write the relative transmitted signal phase as:

$$\Delta\phi_{TX,n} = \Delta\xi_n + \Delta j_n \tag{4.6}$$

Here we introduce a Δj term to encompass any phase differences that may arise from the physical system implementation, much like the Δi term introduced in Section 4.2. This Δj_n term can also be directly measured and removed by applying a known value to ξ_n and measuring $\Delta\phi_{TX,n}$ directly. Once the value is know, it can be removed with a single arithmetic operation just like Δi_n.

This transmitted signal beamformer scheme allows for arbitrary modulation of the transmitted signal, as it provides a pure carrier tone that can be modulated. This modulation could be implemented before the transmitted signal phase delay blocks, or could alternatively be mixed in after the phase shift is applied.

Additionally, as briefly noted in Section 4.2, the transmitted signal beamformer is entirely separate from the received signal beamformer. Due to this separation, the transmit and receive arrays can be electronically oriented to point in different directions. This allows for more system-level flexibility and more dynamic resource allocation capabilities.

4.4 Component Selection

A proof-of-concept DLL-HPC prototype has been designed and fabricated. This section details the component selection and analog phase shifter design for the final prototype. The prototype is designed for the 5G NR-FR1 band, more commonly known as the 2.4 GHz industrial, scientific, and medical (ISM) band, due to the abundance of commercially available components that operate at this frequency.

4.4.1 Commercially Available Components

In order to construct a prototype of this new DLL-HPC, the abundance of commercial off-the-shelf (COTS) components available at the 5G NR-FR1, or 2.4 GHz ISM, band was leveraged where possible. The use of COTS components allowed well-defined system-level blocks to be used and greatly sped up the construction and characterization of the prototype. The final prototype design used the components listed in Table 4.2 for the various components shown in Figures 4.2 and 4.3.

Phase-Locked Loop (PLL)	LTC6946-4
Mixer	LTC5562
Phase Detector	ADF4002
RF Amplifier	PMA3-83LN+
IF Amplifier	LTC6253
Microcontroller	TM4C1294NCPDT
Digital-Analog Converter	MCP4725
Analog-Digital Converter	built in

Table 4.2: Components used and characterized in prototype designs.

4.4.2 Analog Phase Shifter Design

In order to leverage the hybrid analog/digital nature of the received signal beamformer outlined in Section 4.2, an analog phase shifter is needed for the DLL feedback loop. Without an analog phase shifter, precision would be lost due to quantization errors in both the measured receive signal phase and the transmitted signal phase [58]. However, analog phase shifters in the 5G NR-FR1 band were either non-existent or prohibitively expensive for iterative prototyping. For this reason a modified implementation of the analog phase shifter found in [27] was used.

In order to provide an accurate and controllable phase shift at high frequencies, a low-frequency phase shifter was implemented on the PLL reference signals. Because an integer-n PLL's output tracks the reference signal, any phase shift present in the reference will be multiplied by n, or:

$$\phi_{PLL} = n \cdot \phi_{Ref} \tag{4.7}$$

This approach can be visualized as in Fig. 4.4 and requires a separate PLL for each phase shifted signal. The increase in power and area consumption by requiring multiple PLLs instead of a single local oscillator source to be split and phase shifted separately is partially offset by the flexibility of this approach. Because the phase shift is scaled up by a factor of n, a low-frequency reference signal only needs a very small controllable phase shift to provide a full 360° of phase shift at a higher frequency. For the presented prototype a 100 MHz reference signal is used to generate a 2.410 GHz signal; therefore, the 100 MHz reference signal only needs a maximum phase shift of $360°/24.1$ or 15°. Additionally, the phase shifter only needs to be characterized a single time for any PLL output frequency, as a change in the scalar n from (4.7) does not affect the phase shifter's performance.

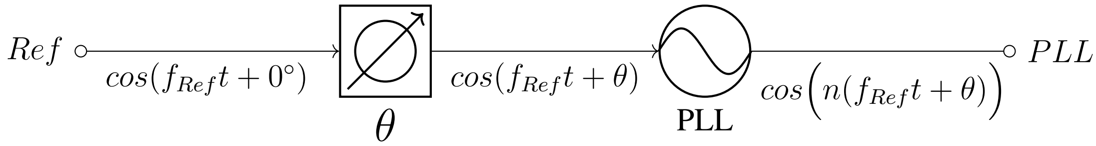


Figure 4.4: PLL output phase being controlled by reference signal phase

The final phase shifter design is shown in Fig. 4.5 alongside simulation results from ADS. The varactor implemented is an Infineon BB639 variable capacitor diode which has a tuning range from approximately 20pF to 38pF when sweeping the reverse bias voltage from $1V$ to $3.3V$. Given the target PLL output of 2.410 GHz, the simulated phase shifter's range of exactly 15° was deemed sufficient.

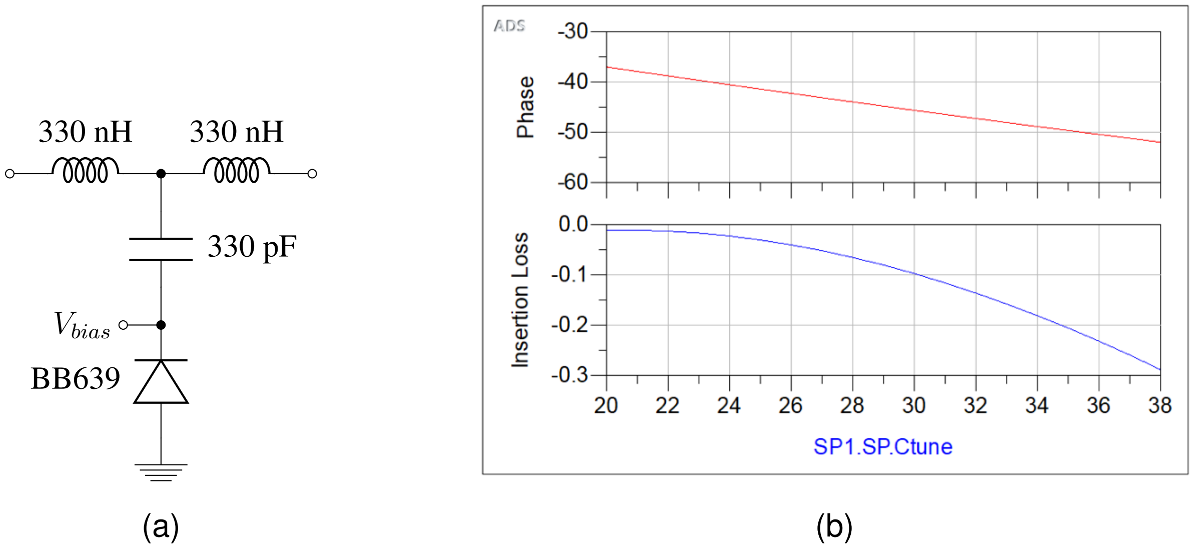


Figure 4.5: (a) Reference signal analog phase shifter (b) Simulated phase shifter performance for a 100 MHz input with 50 Ohm ports

4.5 DLL-HPC System Prototype

A 4-element DLL-HPC prototype was designed and fabricated on several individual PCBs. The received signal beamformer outlined in Section 4.2 and transmitted signal beamformer outlined in Section 4.3 each occupy a single 6-layer PCB. Additionally, a 2-layer PCB was fabricated according to the Texas Instruments BoosterPack standard [59] to allow easy connection of the TM4C microcontroller to both beamformers as in Fig. 4.1. System level implementation of each of these PCBs and the software required to operate the DLL-HPC are presented here, while system level test and characterization results will be presented in Chapter 5. Detailed design documents, component level characterization, and software for this prototype can be found in Appendix A.

4.5.1 Received Signal Beamformer Design

The DLL-HPC prototype's downconversion beamformer was designed to function in a cascaded manner; that is, the reference signal for the n^{th} element is the $(n-1)^{th}$ element rather than the 0^{th} element. This was done to simplify the design of a multi-element prototype and relieve signal routing constraints on a single 4-element DLL-HPC PCB. Fig. 4.6 shows the design of the 0^{th} and 1^{st} element in a 4-element DLL-HPC received signal beamformer; the 2^{nd} and 3^{rd} elements are not pictured as they are identical to the 1^{st} element.

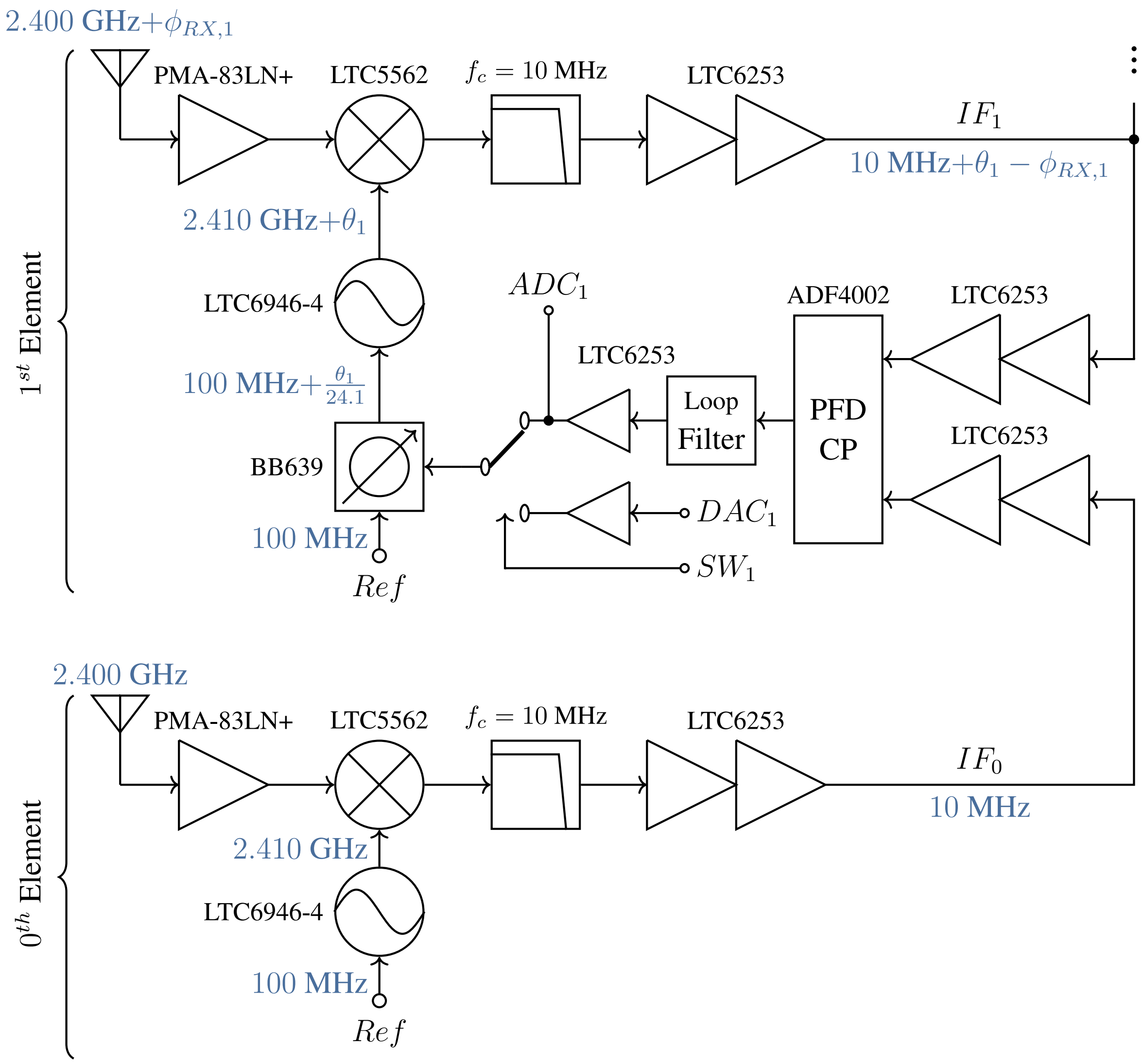


Figure 4.6: DLL-HPC received signal beamformer prototype design

The operation of the system in Fig. 4.6 has already been explained in Section 4.2, but a few points of the hardware implementation bear mentioning. A phase-frequency detector (PFD) / charge pump (CP) is used as the phase detector in this system due to its 0° steady state error and fast settling time [60]. This choice of phase detector drove the decision to include multiple IF amplifiers in an attempt to saturate the IF signal. PFDs function better with square wave inputs due to their inherently digital design [61], and this allows the DLL-HPC to accurately track and receive weaker received signals. Additionally, unity gain buffer amplifiers are implemented before the SPDT switch to provide a high input impedance to the CP loop filter and DAC when driving the analog phase shifter discussed in Section 4.4.2.

In order to verify that a cascaded configuration of DLL-HPCs can still provide proper phase conjugation, we can define the phase difference at the input to the PFD in terms of the n^{th} and $(n-1)^{th}$ received signal phase in a manner similar to (4.3) as:

$$\Delta\phi_{IF,n} = (\theta_n - \theta_{n-1}) - (\phi_{RX,n} - \phi_{RX,n-1}) + \Delta i_n \tag{4.8}$$

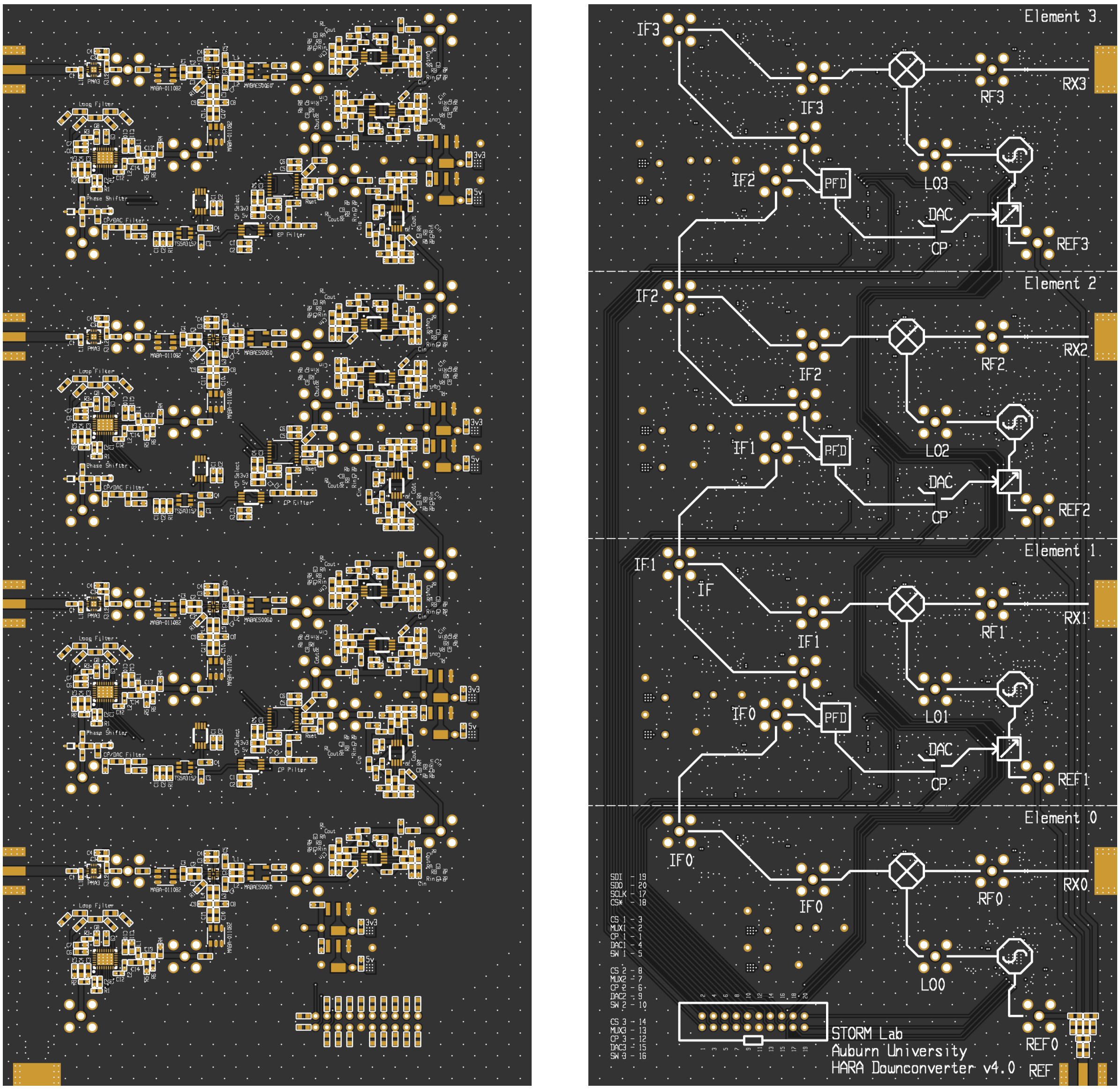
$$\Delta\phi_{IF,n} = \Delta\theta_{n,n-1} - \Delta\phi_{RX,n,n-1} + \Delta i_n \tag{4.9}$$

Here we once again employ a single Δi term to encompass any phase difference caused by the physical system which will be constant due to frequency synchronization. To simplify our examination of the cascaded system, we will assume that the Δi term has been measured and removed by calibration and can therefore be removed from (4.9). During closed-loop steady state operation, the phase delay block will be driven according to (4.5) such that $\Delta\phi_{IF,n} = 0°$ and therefore $\Delta\theta_{n,n-1} = \Delta\phi_{RX,n,n-1}$. This means we can define the absolute value of θ_n relative to the 0^{th} element by taking the sum of the $\Delta\theta_n$ terms at each cascaded element:

$$\begin{aligned}
\Delta\theta_{n,n-1} &= \Delta\phi_{RX,n,n-1} \\
\Delta\theta_{n,n-1} &= \phi_{RX,n} - \phi_{RX,n-1} \qquad (4.10) \\
\theta_n &= \sum_{k=0}^{n} \Delta\theta_{n,n-1} \\
\theta_n &= \sum_{k=0}^{n} \phi_{RX,n} - \phi_{RX,n-1} \\
\theta_n &= \phi_{RX,n} \qquad (4.11)
\end{aligned}$$

Here we can see that the received signal phase will still be measured correctly by the DLL-HPC's microcontroller, as the instantaneous value of θ_n is the value measured and used for phase conjugation as discussed in Section 4.2. Because this phase is directly conjugated and written to the transmitted signal beamformer by the microcontroller, the transmitted signal will still be able to perform phase conjugation according to (3.2).

Renders of the DLL-HPC received signal beamformer prototype PCB are shown in Figures 4.7 and 4.8. All components are placed on the front-side in order to simplify assembly and characterization, while the back-side contains SMA test points and silkscreen diagrams to clarify operation. Detailed schematics and other design documents can be found in Appendix A.



(a) (b)

Figure 4.7: DLL-HPC received signal beamformer PCB renders (a) front (b) back

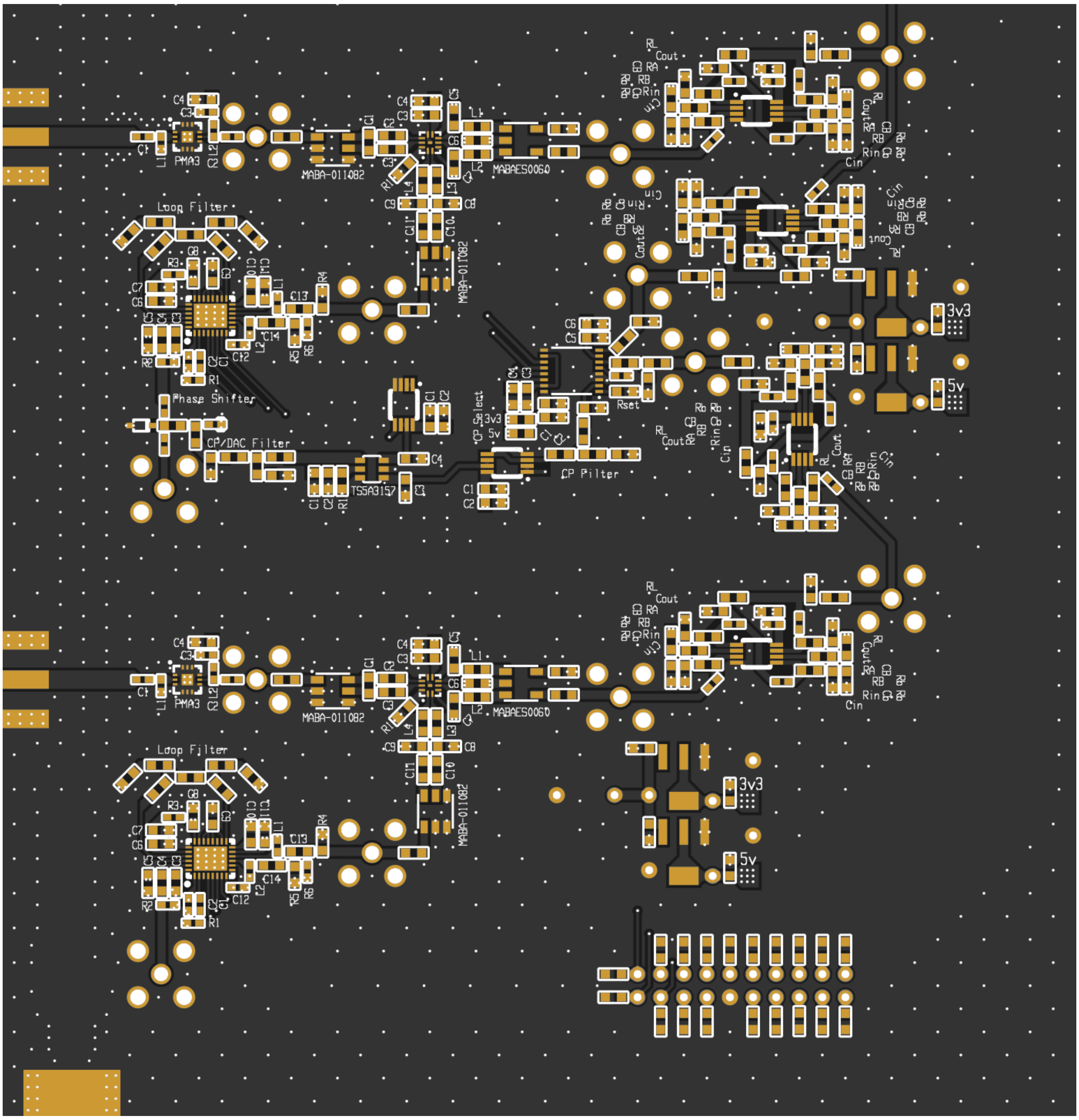


Figure 4.8: Close-up render of DLL-HPC received signal beamformer PCB 0^{th} and 1^{st} element

4.5.2 Transmitted Signal Beamformer Design

The DLL-HPC prototype's transmitted signal beamformer was designed as close to the system outlined in Fig. 4.3 as possible and is shown in Fig. 4.9. Only the 0^{th} and 1^{st} transmitted signal beamformers are shown, as the 2^{nd} and 3^{rd} are identical to the 1^{st}. The prototype system presented in Fig. 4.9 differs from that presented in Fig. 4.3 only by the phase shifter location and addition of a unity gain buffer amplifier to isolate the DAC control voltage from the analog phase shifter itself and prevent signal leakage.

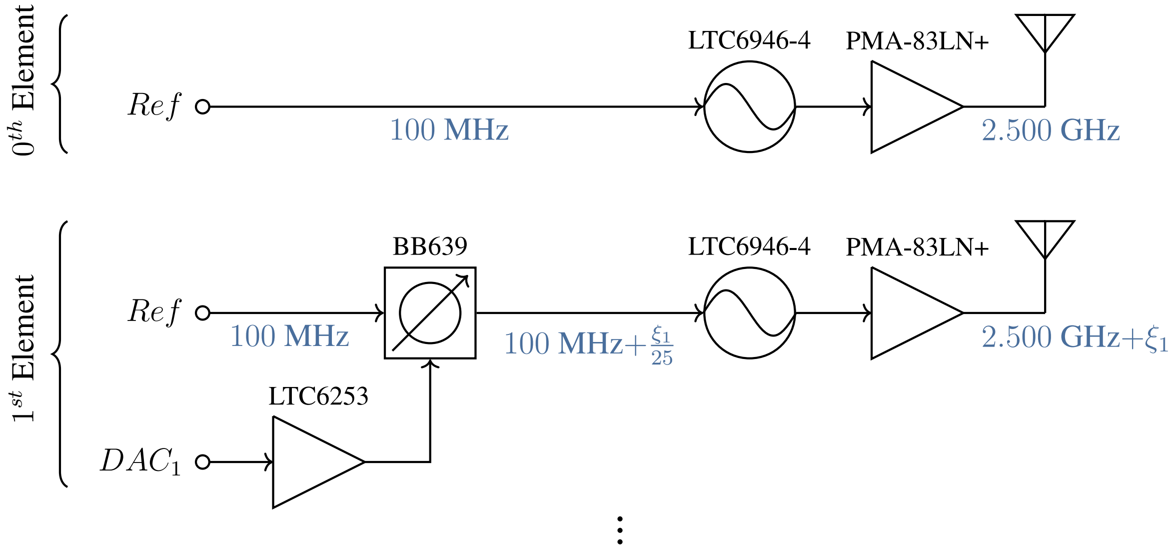
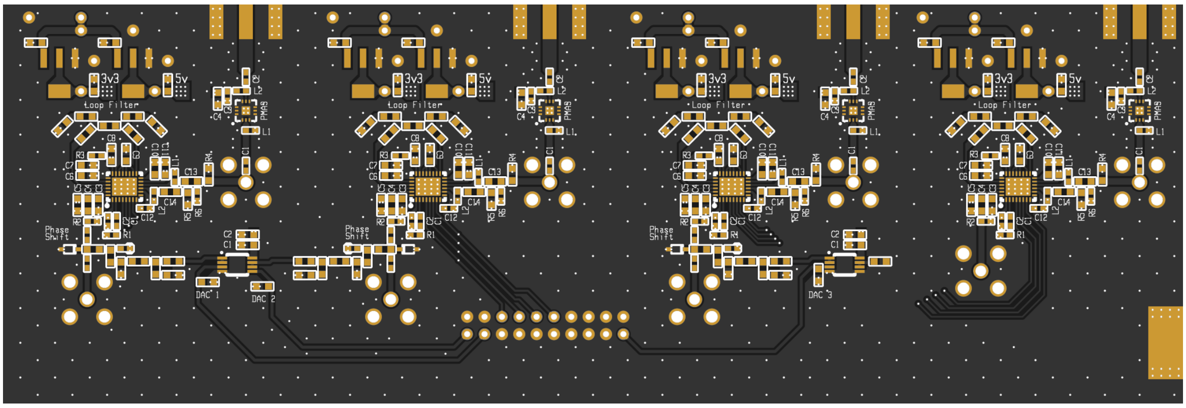


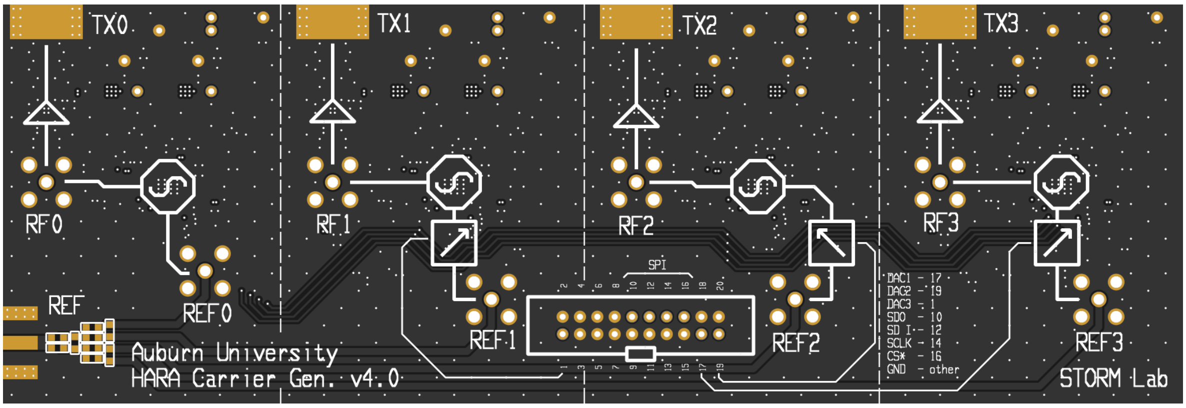
Figure 4.9: DLL-HPC transmitted signal beamformer prototype design

As mentioned in Section 4.4.2, the designed analog phase shifter requires a separate PLL for each phase shifted signal. This significantly raises the power consumption and size of the prototype, but allows for an arbitrary transmit signal frequency choice without re-characterizing the phase shifter. Different transmit frequencies can be selected by modifying the LUT's output with a simple scalar as shown in (4.7).

Renders of the DLL-HPC transmitted signal beamformer prototype PCB are shown in Fig. 4.10. All components are placed on the front-side in order to simplify assembly and characterization, while the back-side contains SMA test points and silkscreen diagrams to clarify operation. Detailed schematics and other design documents can be found in Appendix A.



(a)

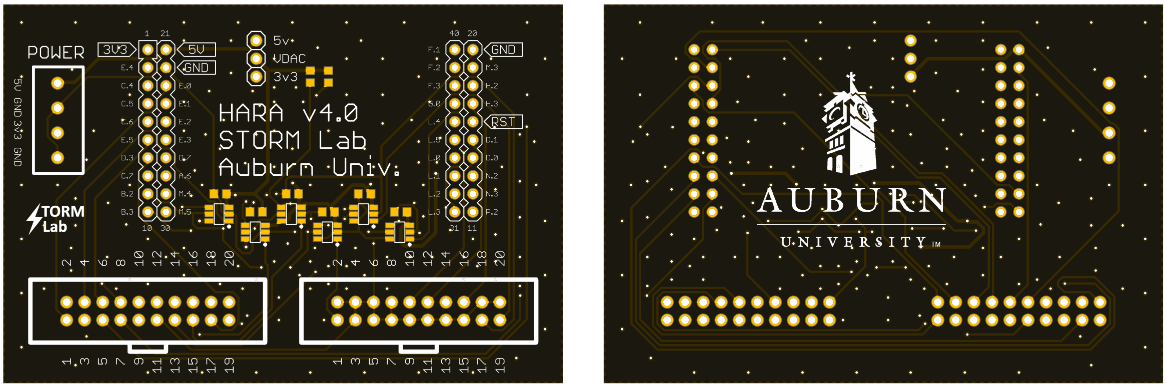


(b)

Figure 4.10: DLL-HPC transmitted signal beamformer PCB renders (a) front (b) back

4.5.3 TM4C BoosterPack Design

In order to control the DLL-HPC prototype boards with the chosen TM4C microcontroller, a simple ribbon-cable interface board was designed according to the Texas Instruments BoosterPack standard [59]. The BoosterPack contains six MCP4725 digital-analog converters (DACs) interfaced with the I^2C bus to provide the needed DAC signals for each phase shifter. Additionally, the board provides a separate 20-pin ribbon cable connection for the transmitted signal beamformer board and received signal beamformer board, each containing the necessary DAC signals to control the various phase shifters and SPI bus signals to interface with the PLL and PFD chips. Fig. 4.11 shows the front and back sides of the BoosterPack PCB; detailed schematics and other design documents can be found in Appendix A.



(a) (b)

Figure 4.11: DLL-HPC interface BoosterPack PCB renders (a) front (b) back

4.5.4 Firmware Design

The TM4C1294NCPDT microcontroller was chosen for this prototype due to its large number of communication interfaces, high clock speed, and availability as a prototyping LaunchPad platform. The TM4C1294NCPDT has a maximum clock speed of 120 MHz and sufficient integrated resources for DLL-HPC operation: an I^2C interface to control the necessary DACs, an SPI interface to control the PLLs and PFDs, a UART interface to communicate with a host computer via USB, and a reconfigurable 20-channel analog-digital converter (ADC) [62].

The firmware to control the DLL-HPC was created to work alongside the user-interface detailed in Section 4.5.5. As such, the microcontroller implements two threads: a high-priority periodic sampling/reporting thread and low-priority event processing queue. These two threads are discussed in general here, and the documented source code is provided in Appendix A.

The high-priority periodic thread is implemented using a 100 Hz timer interrupt to perform time-sensitive tasks at different rates. This frequency was chosen as the relative phase of received signals is very unlikely to vary significantly over the course of a second, making 100 samples per second sufficient to capture any changes. The ADC is configured to sample the current phase delay control voltage value of each received signal beamformer at 100 Hz and keep track of the last 10 samples for each value. If the system is currently configured to perform phase conjugation, these ADC values are used to determine the current received signal phase and write the conjugate of this value to the transmitted signal phase delay blocks. The current state value of the PFD in each received signal beamformer is sampled at 10 Hz to determine if the closed-loop DLL has achieved steady state lock. Additionally, a packet of information containing the current state of the system and values of all phase shifters is constructed and sent through the UART USB connection to the user-interface at a rate of 10 Hz; the contents of this periodic update will be discussed in Section 4.5.5.

The low-priority event processing queue is implemented in the main loop of the program, remaining in a low-power state unless an event has been queued. An internal first-in first-out (FIFO) buffer holds 16-bit values with opcode corresponding to different commands to be executed. These commands may be queued by received UART commands from the user-interface or may be queued by the system depending on the current operating mode. As an example, the user may add commands to reset the internal counters of a PFD chip through the user-interface, or the microcontroller may add this same command to the event queue if it detects that the PFD has saturated. The following events may be queued:

1. Manually adjust the $\Delta i/\Delta j$ term for an individual element by a chosen amount.
2. Automatically set Δi term according to Algorithm 1.
3. Enable / disable actively steered phased array mode.
4. Enable / disable retrodirecting phase conjugator mode.
5. Write a phase value directly to an individual θ or ξ phase shifter.
6. Write a value directly to an individual DAC.
7. Reconfigure or recalibrate a PLL.
8. Reconfigure a PFD chip's settings.
9. Query a PFD chip's current state.

4.5.5 Graphical User-Interface Design

A simple graphical user-interface (GUI) was made to interface with the TM4C microcontroller through a UART USB connection using Python and TkInter. The main purpose of this application is to provide methods to both view and log the microcontroller's status in real-time as well as to interface with the DLL-HPC components and operating modes. The GUI consists of a set of tabs for different commands and several continuously updated visualizations that are always on-screen. Documented source code for the GUI can be found in Appendix A.

A screen-capture of the GUI is shown in Fig. 4.12 where several real-time functionalities are available. On the right side of the GUI, the current ADC sample for elements 1-3 are shown alongside a 0^{th} ADC channel which is unconnected and can be used as a real-time voltmeter. On the lower portion of the GUI a table of the current status values for each element is shown: this includes the PFD's current lock state, the current ADC value, the two phase delay values θ and ξ, the calculated received signal phase ϕ, the current Δi and Δj offset values, and the current SPDT switch state. These values can all be logged alongside computer timestamps into a comma-separated values (CSV) file using the Log button. Additionally, interesting samples can be annotated within the CSV file by pressing the Log Flag button to add a flag to the current sample; this was implemented to assist in noting and finding interesting data points during operation. These real-time visualization and logging tools are always visible and active within the GUI.

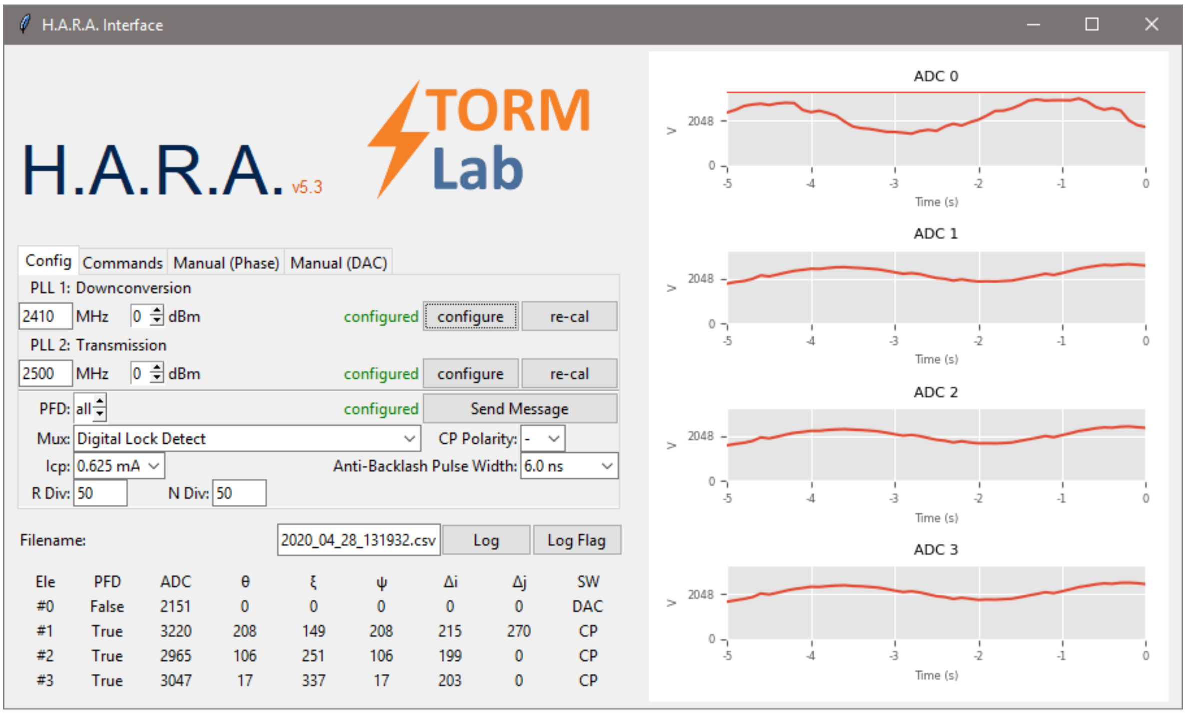


Figure 4.12: DLL-HPC graphical user-interface application

The Configuration tab contains commands for configuring hardware within the DLL-HPC prototype. The downconversion chain and transmission PLL output power and frequency can be configured from this tab, and recalibration messages can be sent to the PLLs if their outputs begin to drift. Additionally, the PFDs can be configured individually or as a group using this tab. The polarity, charge pump current, status indicator value, and divider values can all be manipulated on a per-chip basis.

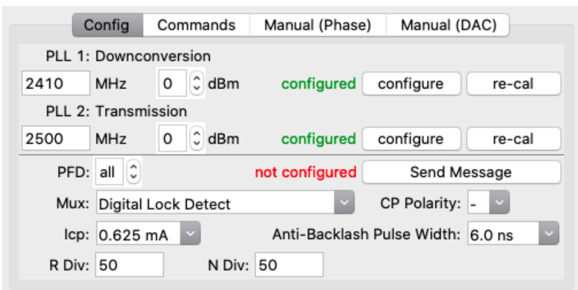


Figure 4.13: DLL-HPC GUI Configuration tab

The Commands tab contains commands for operating the DLL-HPC as a system. Commands are present to adjust the Δi and Δj terms for a single element or multiple elements at once, and an additional command is available to automatically set Δi according to Algorithm 1. In addition to calibrating the system offsets, two modes of operation can be entered from this tab: retrodirection mode and active steer mode. The retrodirection mode performs phase conjugation as outlined in Sections 4.2 and 4.3, while the active steer mode applies pre-calculated phase shifts to the IF and TX signals to perform beamforming along a chosen view angle. The active steer mode is implemented assuming a linearly spaced antenna array with half-wavelength spacing.

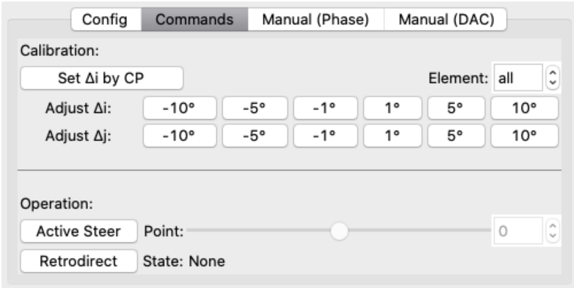
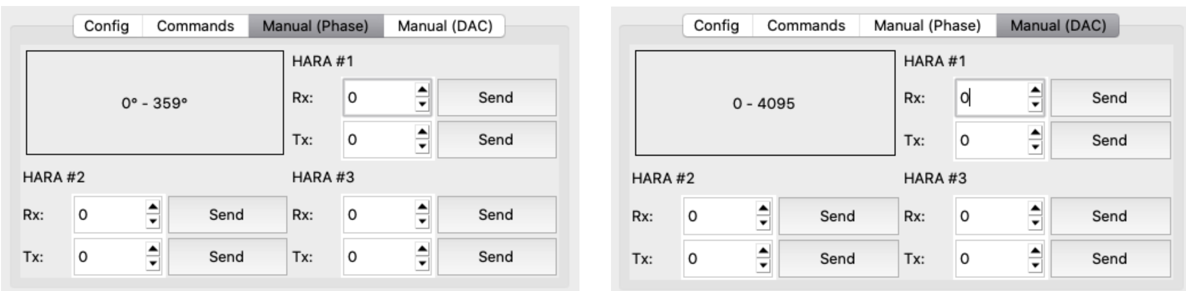


Figure 4.14: DLL-HPC GUI Commands tab

The two Manual tabs allow the user to write a specific value to one of the phase shifters or DACs. If a value is written to the DAC, the integer value is limited to the range of $0 - 4095$ before being written directly to the chosen 12-bit MCP4725 DAC. If a value is written to a phase shifter, the value is first wrapped to a range of $0° - 359.9°$ before being used as the input to the corresponding LUT for the chosen phase shifter. This affords the user both fine control of the phase shifters through direct DAC manipulation and coarse control of the phase shifters by relying on the calculated LUTs.



(a) (b)

Figure 4.15: DLL-HPC GUI Manual (a) phase and (b) DAC manipulation tabs

Chapter 5

Prototype Hybrid Retrodirective Array Performance

The prototype Delay-Locked Loop based Hybrid Phase Conjugator (DLL-HPC) described in Section 4.5 was used to construct a 4-element Hybrid Retrodirective Array (HRA). This chapter details the relevant test and characterization data to prove system-level operation of the prototype. All tests will be described in detail, with additional results for each test available in the Appendices.

5.1 Phase Shifter Characterization

In order for the DLL-HPC to function properly, the phase shifters must be well characterized. Each received signal beamformer phase shifter needs two LUTs: one to map a desired phase shift value to an appropriate DAC value and one to map a measured ADC value to the implemented phase shift θ. The transmitted signal beamformer only needs a single LUT to map a desired phase shift value ξ to an appropriate DAC value, as the transmitted signal beamformer always operates in a single mode.

In order to characterize the analog phase shifters and calculate these LUTs, several tests were performed using a Tektronix MSO64 8 GHz oscilloscope to measure the generated signal phases. SMA test points were designed into the DLL-HPC prototype PCBs at the output of each PLL chip, as shown in Fig. 4.7 and 4.10, so that the phase-shifted PLL outputs could be measured directly as in Fig. 5.1.

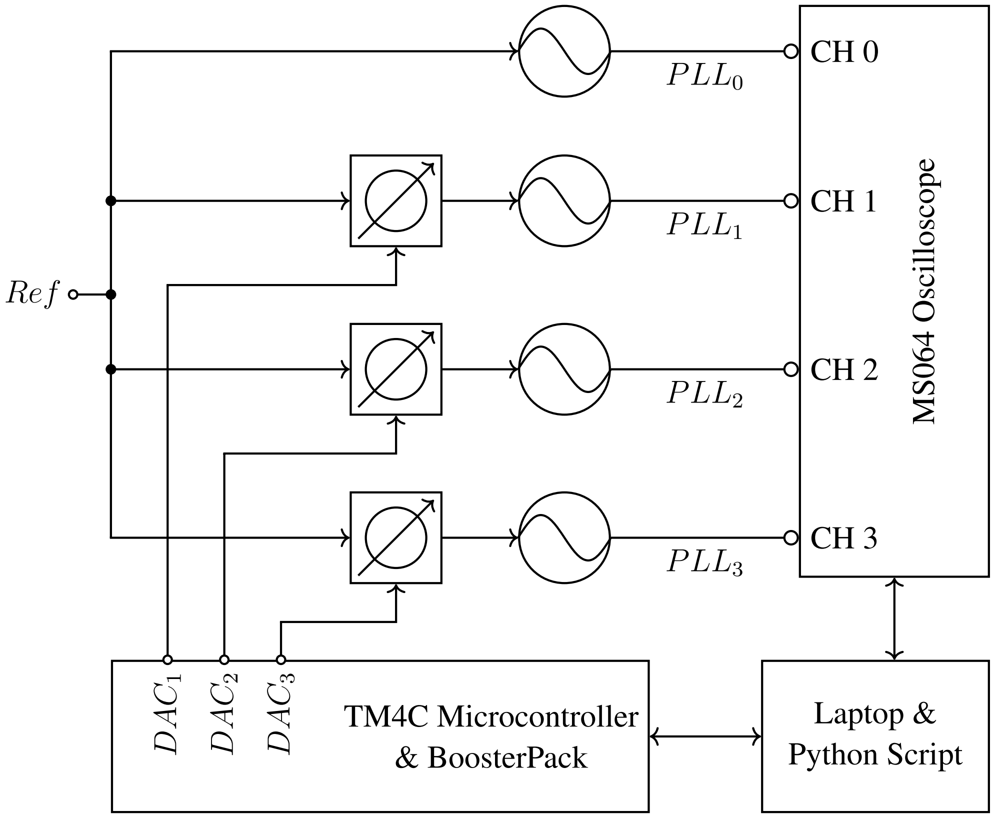


Figure 5.1: Phase shifter characterization test setup

A Python script was written to automate the collection of data for this test. The DAC values for each phase shifter were swept linearly over the full range of output values with a user-selectable step size. At each point 10,000 phase measurements relative to the 0^{th} element were taken and the mean, max, minimum, and standard deviation for this sample set was recorded in a CSV file. This sweep was performed four times (twice ascending and twice descending) to ensure that the phase shifter performance was constant over time. Depending on the selected step size, the test could take up to 13 hours to simultaneously characterize all three phase shifters.

Several MATLAB scripts were used to analyze the recorded data and generate LUTs for the TM4C firmware. A first MATLAB script analyzed the generated CSV files and created a single table of data relating the measurements to the corresponding DAC values. A second MATLAB script was used to plot the measured values, calculate a curve of best fit for LUT generation, and store these coefficients for the dataset in a separate file. A final MATLAB script used the generated coefficients for the best test sets to generate the appropriate LUTs for the DLL-HPC prototype and write the corresponding C source code for the TM4C firmware. All MATLAB and Python source code used for testing and characterizing the phase shifters is included in Appendix B.

Figures 5.2 and 5.3 show the data sets used to generate the final LUTs, while several other data sets can be found in Appendix B. The measured phase values are presented alongside the standard deviations and calculated curve of best fit for LUT generation. Ideally, the phase shifters that are not being swept would remain constant in these tests, but slight variations can be seen among other phase shifters as the control voltage is varied. These variations reach a worst-case of around 45°, as seen in the DAC 3 Mean plot in Fig. 5.2, and are likely due to parallel phase shifters being inductively coupled together according to the design in Fig. 4.5. Luckily, only one full phase rotation of 360° is necessary to generate a LUT to control the phase shifters, and the start-stop range of the DAC values for a given LUT can be chosen where the other phase shifters vary the least. However, these variations may introduce errors into the LUT used to map a measured ADC value to a phase shift θ, as this LUT must span the entire range of possible DC control voltages.

The phase shifter characterization data also agrees well with the simulated phase shifter in Fig. 4.5. The simulation data showed that varying the control voltage from 1V to 3V should cause a 360° phase shift. The fullscale DAC output voltage is 5V, which means that DAC values of 819 and 2457 correspond to 1V and 3V respectively. If we look at the graphs in Figures 5.2 and 5.3, we can see that the phase shifts at these DAC values are roughly 360° apart for each phase shifter.

Fig. 5.3 also shows a higher standard deviation for most samples than Fig. 5.2. This is due to the design of the DLL-HPC transmitted signal beamformer board. The unity gain buffer amplifiers that serve to isolate the DAC output from the phase shifter circuit are shared between two elements due to the LTC5562 pinout, leading to a split in the ground plane between the buffer chip and phase shifter. This introduces a small amount of noise on the buffer signal's output, which manifests itself as a higher standard deviation for the transmitted signal phase. These design choices can be seen in the schematics and PCB layout files included in Appendix A.

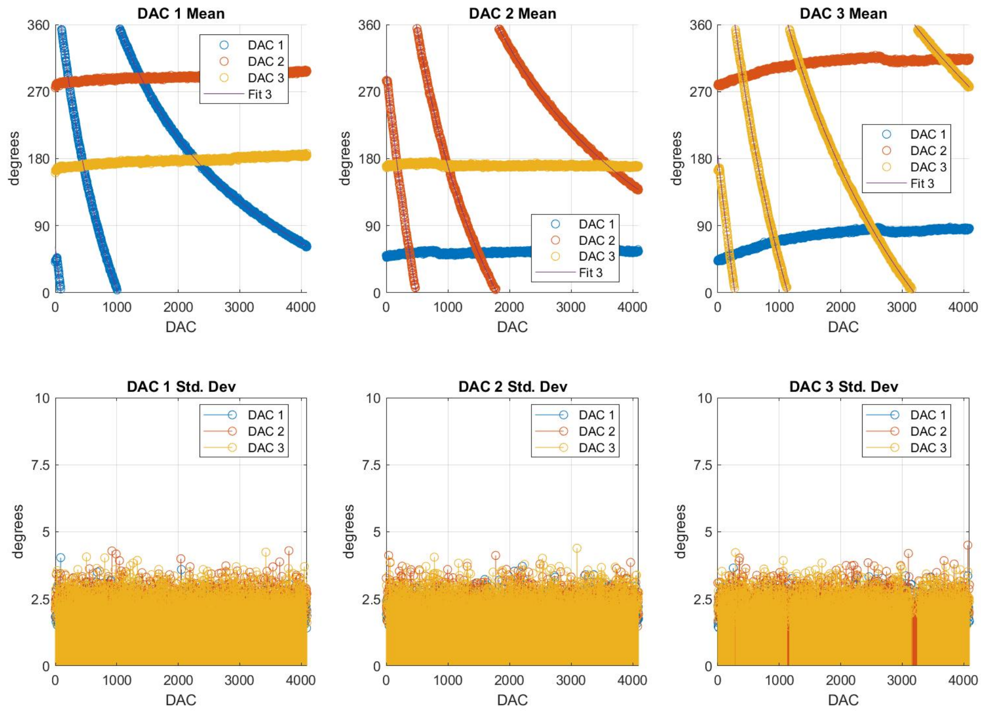


Figure 5.2: Measured phase shifter characterization data for received signal beamformer

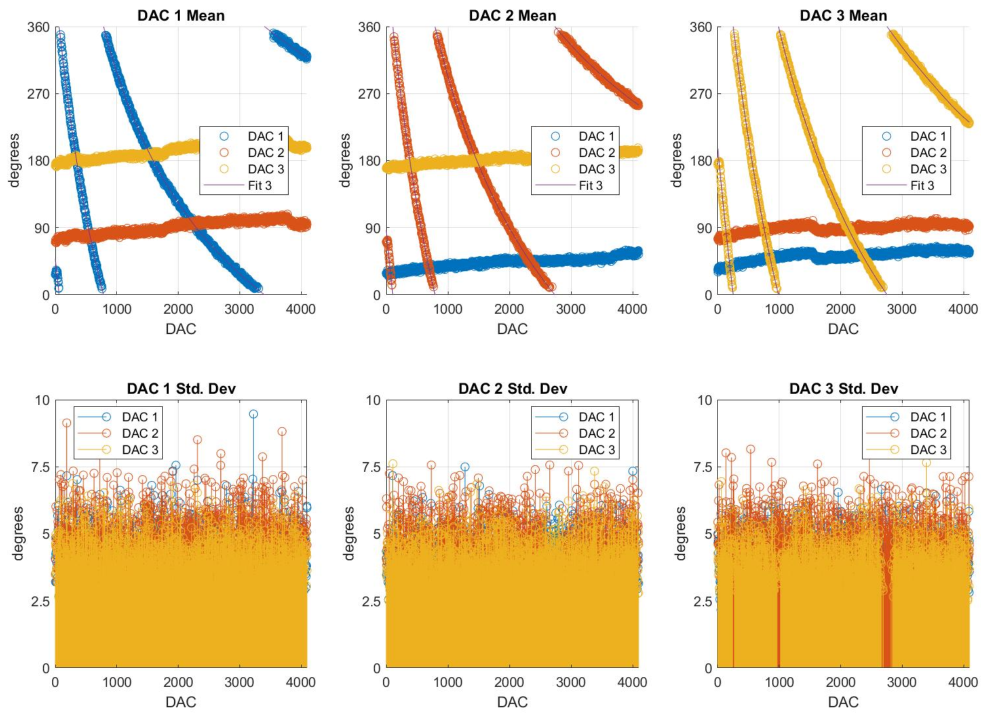


Figure 5.3: Measured phase shifter characterization data for transmitted signal beamformer

5.2 Received Signal Phase Reporting

In order for the DLL-HPC to be able to conjugate the received signal phase, it must be able to accurately determine the received signal phase using the LUTs generated in Section 5.1. In order to verify that the measured DLL-HPC's received signal phase is accurate, a Keysight M8196A 45 GHz Arbitrary Waveform Generator (AWG) was used to generate phase shifted received signals for the DLL-HPC as in Fig. 5.4. It is important to note that the AWG is only capable of generating two sources at a time, which mandated that the test be performed separately for each of the cascaded DLL-HPC received signal beamformers.

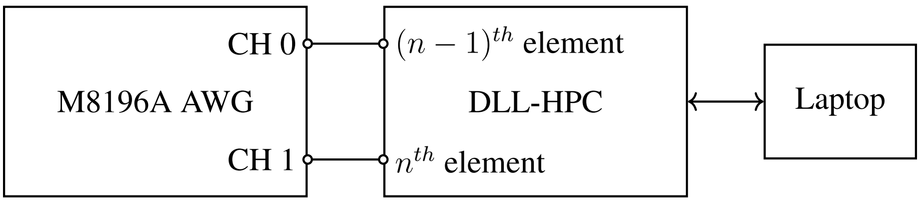


Figure 5.4: Received signal phase reporting test setup

The received signal phase was manually swept from 0° to 360° using the AWG while the DLL-HPC GUI was used to log a CSV file with several hundred samples of the reported received signal phase for each step. A MATLAB script was then used to process the data within the separate CSV files to create a single data file for each element that contained the actual AWG phase, the DLL-HPC's reported phase, and a phase value calculated by MATLAB using the coefficients and ADC value. Statistics about the DLL-HPC's ability to correctly measure the received signal phase are listed in Table 5.1, and a plot of the reported received signal phase and calculated error magnitude for each DLL-HPC is shown in Fig. 5.5.

	Element			Overall
	1	2	3	
$\overline{\lvert error \rvert}$	$+4.53°$	$+2.98°$	$+2.59°$	$+3.36°$
$\overline{error}$	$-3.30°$	$+2.03°$	$+1.04°$	$-0.08°$
std. dev.$(error)$	$+1.15°$	$+1.53°$	$+1.89°$	$+1.53°$
max$(\lvert error \rvert)$	$+13.07°$	$+7.07°$	$+6.45°$	$+13.07°$

Table 5.1: DLL-HPC received signal phase reporting statistics by element

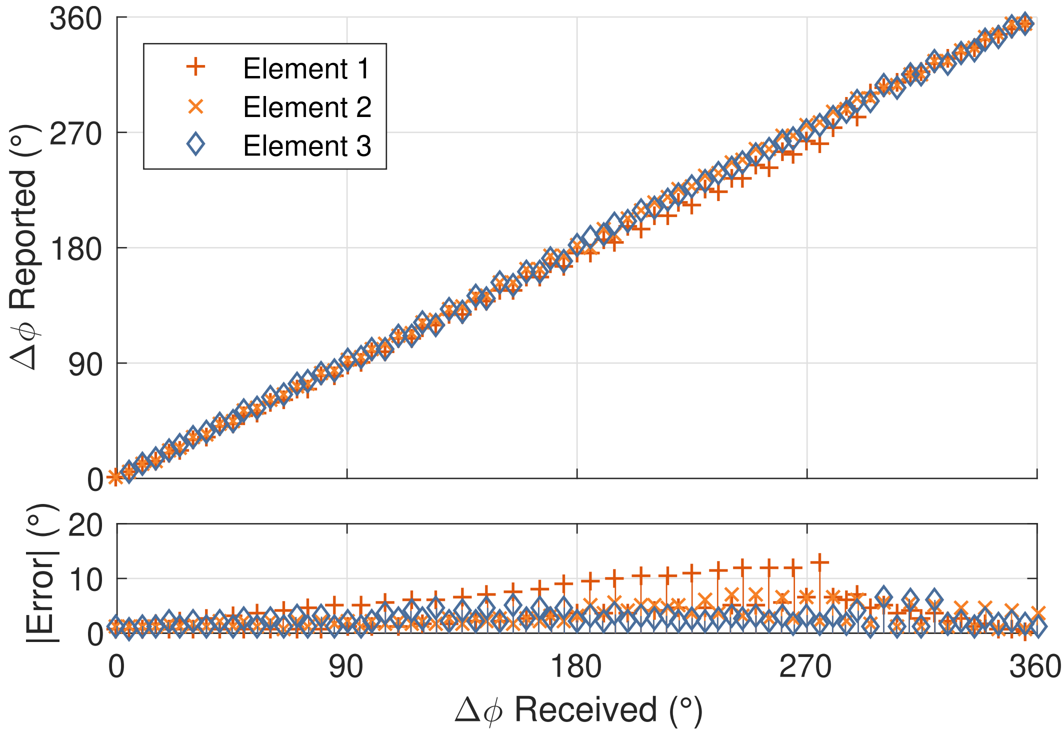


Figure 5.5: Reported received signal phase and error magnitude

Although the data in Fig. 5.5 appears to contain large errors, there are actually very few samples with error magnitudes above 7.5°; Table 5.1 is a better indicator of DLL-HPC performance. The average error magnitude of $3.36°$ indicates that the reported value will reliably be within $\pm 3.36°$, which corresponds to an effective resolution of $6.72°$. For comparison, a perfectly calibrated 6-bit digital phase shifter would be required to achieve a comparable resolution of $5.625°$. Overall, the performance of the DLL-HPC's received signal phase reporting is deemed acceptable due to the almost $0°$ average error and high effective resolution.

In addition to the received signal phase being swept, the received signal strength was swept from 0 dBm to -60 dBm: the lowest output power available to the Keysight AWG. The DLL-HPC performed the same under different input powers due to the intentional saturation of the IF signal through multiple amplifiers as discussed in Section 4.5.1. The PFD/CP phase detector works best with digital inputs, so a fully saturated IF signal allows the DLL-HPC to function with low received signal powers.

Further, these tests show that the DLL-HPC will allow for highly accurate direction of arrival calculations, as outlined in Section 3.4.2. As long as the array geometry is known, the received signal phase reported in these tests would provide an accurate calculation of the direction of arrival. This test is performed in Auburn University's anechoic chamber in Section 5.4.

5.3 Geometry Independent Received Signal Array Gain

In order to verify the operation of a DLL-HPC based HRA, a 4-element HRA prototype was tested in Auburn University's anechoic chamber. The received signal beamformer was tested first on its own to verify that the IF signals of all four elements were being driven in-phase to provide ideal beamformer gain [29]. The HRA was mounted onto a Diamond Engineering DAMS-7000 in the anechoic chamber to rotate the array, while the IF signals were summed using the MSO64 Oscilloscope's Spectrum Analyzer mode. Both the DAMS rotation and MSO64 measurements were automated by a Python script listed in Appendix C. Additionally, a 2.400 GHz source placed outside of the anechoic chamber was piped in and broadcast using a Rubber Duck antenna for the HRA to receive. The test setup is shown in Fig. 5.6.

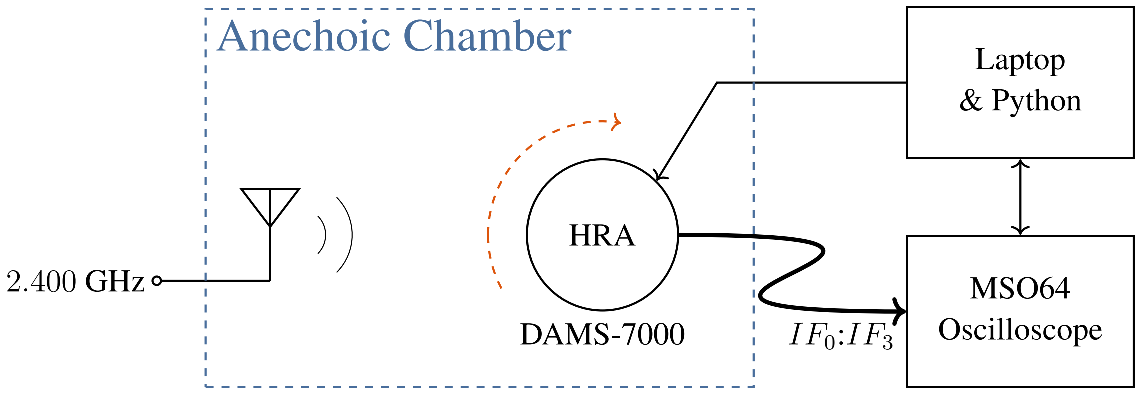


Figure 5.6: Received signal beamformer test setup

Several different linear antenna array geometries were tested in order to verify that the physical layout of a DLL-HPC based HRA does not affect performance as discussed in Section 3.2.3. The antenna arrays were implemented using Rubber Duck antennas so that the antenna patterns are isotropic across all angles of arrival; this allows us to treat the antenna factor as one when normalizing and only look at the array factor [30]. Further, the HRA was only swept over a range of view angles from $0°$ to $90°$, as the array pattern for view angles in all other quadrants is a mirrored copy of this quadrant for a linear antenna array [63]. Figure 5.7 shows the received signal gain for each linear antenna array geometry relative to a single antenna element alongside a visualization of the tested linear arrays.

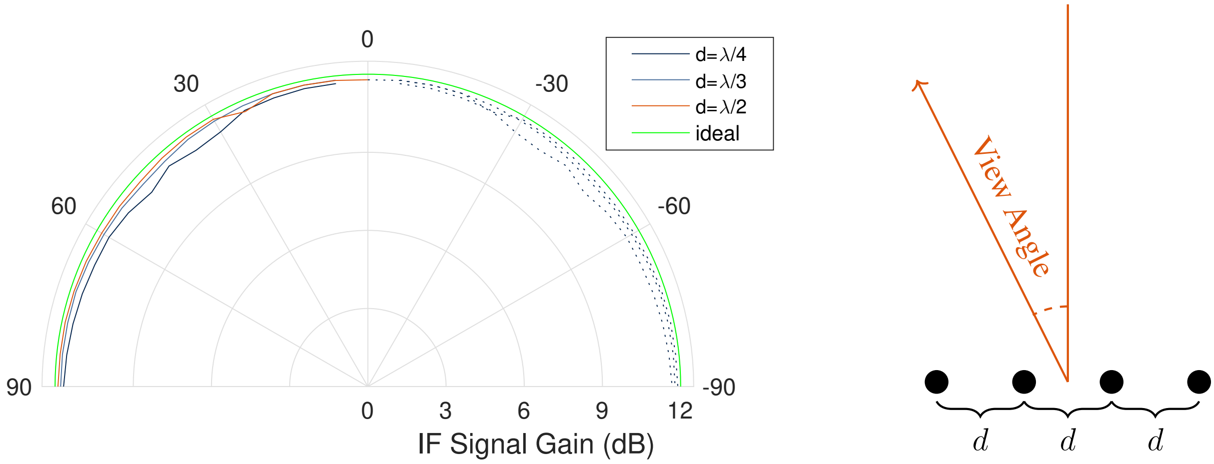


Figure 5.7: Received signal array gain for varying linear antenna array geometries

Fig. 5.7 shows the average of five collected data sets for each antenna array geometry, with a mirrored copy of the data presented as dotted lines to make the graph easier to read. Additionally, a reference plot for an ideal 4-element antenna array is included which provides a voltage gain of 4, which is a power gain 16 (12 dB). It is clear that the received signal beamformer operates in a nearly ideal fashion for all view angles, as the worst case gain of 11.16 dB corresponds to only a 18.4% reduction from ideal power gain. The average array gain and percent difference from an ideal 4-element array are given for each antenna array geometry in Table 5.2, showing that the DLL-HPC functions well as a received signal beamformer regardless of array geometry.

	Spacing		
	$\lambda/4$	$\lambda/3$	$\lambda/2$
mean	11.49 dB	11.75 dB	11.81 dB
mean % error	11.8%	6.5%	5.2%
worst case	11.16 dB	11.61 dB	11.56 dB
worst case % error	18.4%	9.5%	10.4%

Table 5.2: DLL-HPC received signal beamformer gain relative to a single antenna element for different linear array geometries.

5.4 Direction of Arrival Calculations

During the testing described in Section 5.3, the received signal phases were recorded for each antenna array view angle and geometry as well as the received signal strength. This allows for an array-level test of the DLL-HPC based HRA's direction of arrival calculation capabilities. This test used the same test setup that was shown in Fig. 5.6. The average error for each antenna array geometry is shown for each DLL-HPC element in Table 5.3.

	Element			Overall
	1	2	3	
$\lambda/4$	$-3.72°$	$-6.60°$	$-23.57°$	$-11.30°$
$\lambda/3$	$-3.70°$	$-13.59°$	$-27.58°$	$-14.96°$
$\lambda/2$	$+23.04°$	$-12.22°$	$+2.14°$	$+4.32°$

Table 5.3: Average error of reported received signal phase for each DLL-HPC received signal beamformer for various linear antenna array geometries

The values listed in Table 5.3 show significantly more error than was reported in Section 5.2. The difference can be attributed to the different test setups: the tests performed in Section 5.2 were performed using bench-top sources to directly test the DLL-HPC hardware, while the non-ideal natures of the physical antenna arrays and anechoic chamber test setup result in the higher error values listed here. It is important to note that the DLL-HPC's reported angles still allow for accurate calculation of the angle of arrival of the received signal, even with the seemingly large errors present.

Fig. 5.8 shows the calculated angle of arrival against the actual angle of arrival for the half-wavelength linearly spaced antenna array. Before the angle of arrival reaches 60° from broadside, the antenna array has an average error magnitude of only 2.3°. It is important to keep in mind that physical antenna arrays do not behave in an ideal manner, and interference among antenna elements can arise at large viewing angles [29]. In fact, phase-conjugator based RDAs in the literature are often not tested beyond ±60° from broadside, as the interference from the physical antennas begins to dominate at this angle [55]. For these reasons, the plots in Fig. 5.8 are not alarming and line up well with other reported RDAs [47, 55].

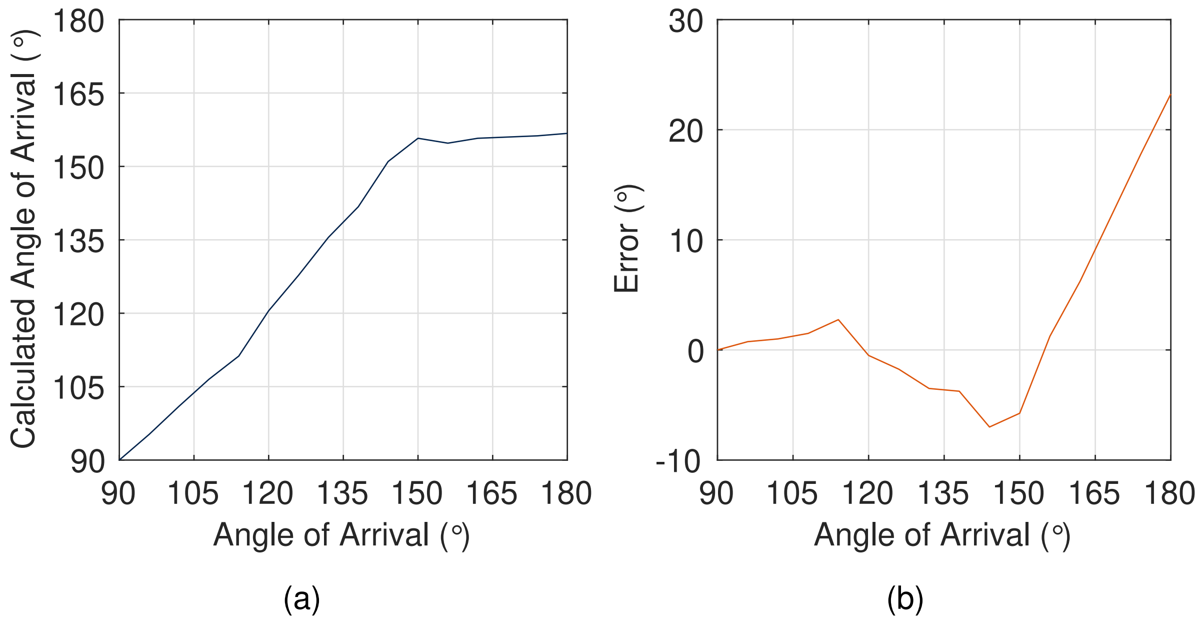


Figure 5.8: Calculated angle of arrival vs. actual angle of arrival for a half-wavelength linearly spaced antenna array (a) calculated values (b) calculated error

Regardless of the antenna-array scale errors that arise from the physical test apparatus and nonidealities of antenna arrays, the DLL-HPC's ability to perform received signal phase reporting has been verified here experimentally. What's more, the tests of Section 5.2 serve as a more pure test of this DLL-HPC functionality by removing the antenna array and directly providing sources with a known relative phase shift.

5.5 Retrodirection Tests

In addition to characterizing the received signal beamformer, the anechoic chamber was used to verify array-level retrodirection of the 4-element HRA prototype. A Tektronix RSA306B Spectrum Analyzer was connected to a Rubber Duck antenna co-located with the 2.4 GHz source antenna to measure the received signal power along the direction of arrival, as shown in Fig. 5.9. The HRA was configured to transmit the phase-conjugated signal at 2.5 GHz so that the retrodirected signal would be easily distinguishable to the Spectrum Analyzer. The Python script listed in Appendix C was used to rotate the HRA and record the Spectrum Analyzer output, while MATLAB was used to analyze and plot the results.

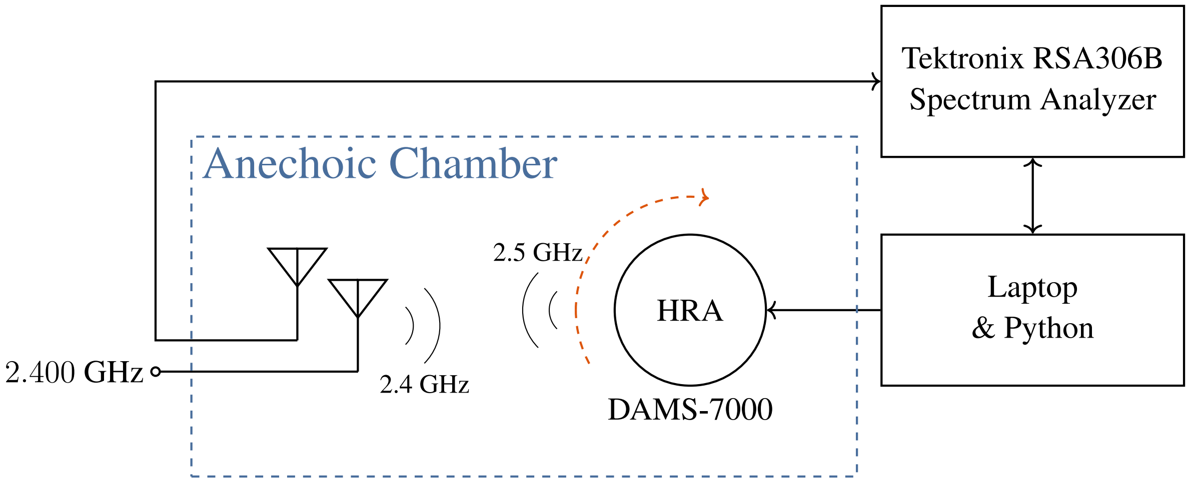


Figure 5.9: Retrodirection test setup

Fig. 5.10 shows the average of the measured results for different linear antenna array geometries. As in Sections 5.3 and 5.4, the measurements were only performed over one quadrant of the linearly spaced array due to the symmetry of the overall array pattern. In the case of an ideal retrodirective array the transmitted signal strength would resemble a perfect circle, and the target would receive the same power regardless of the view angle. The data shown in Fig. 5.10 has been normalized so that the maximum received signal strength lies at 12 dB, the maximum theoretical power gain of a 4-element antenna array. From here we can see that the retrodirective array functions well by looking at the half-power beamwidth. Both the $\lambda/4$ and $\lambda/2$ linear array geometries exhibit half-power beamwidths of $\pm 90°$, as the normalized signal strength does not drop below -3 dB at any point in the field-of view.

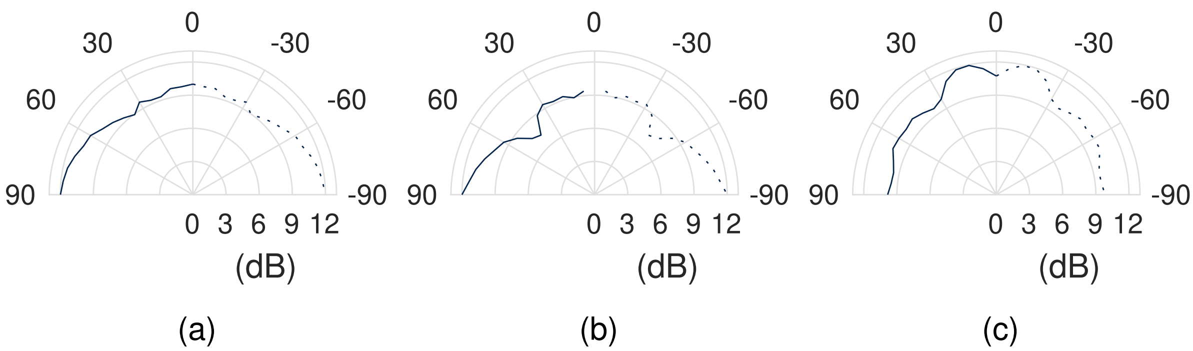


Figure 5.10: Normalized transmitted signal power vs. array view angle (a) $\lambda/4$ spacing (b) $\lambda/3$ spacing (c) $\lambda/2$ spacing

In an effort to explain the differences of Fig. 5.10 from the ideal case, the transmitted signal beamformer phase shifters were tested for accuracy. The test setup from Fig. 5.1 was used to measure the actual transmitted signal phase for a manually selected phase, which was swept from 0° to 360° in 3° steps. For each step, 10,000 phase measurements were performed with the MSO64 8 GHz oscilloscope to determine the mean, max, minimum, and standard deviation of the transmitted signal phase for a chosen phase value. The measured results are shown in Fig. 5.11 alongside relevant statistics in Table 5.4.

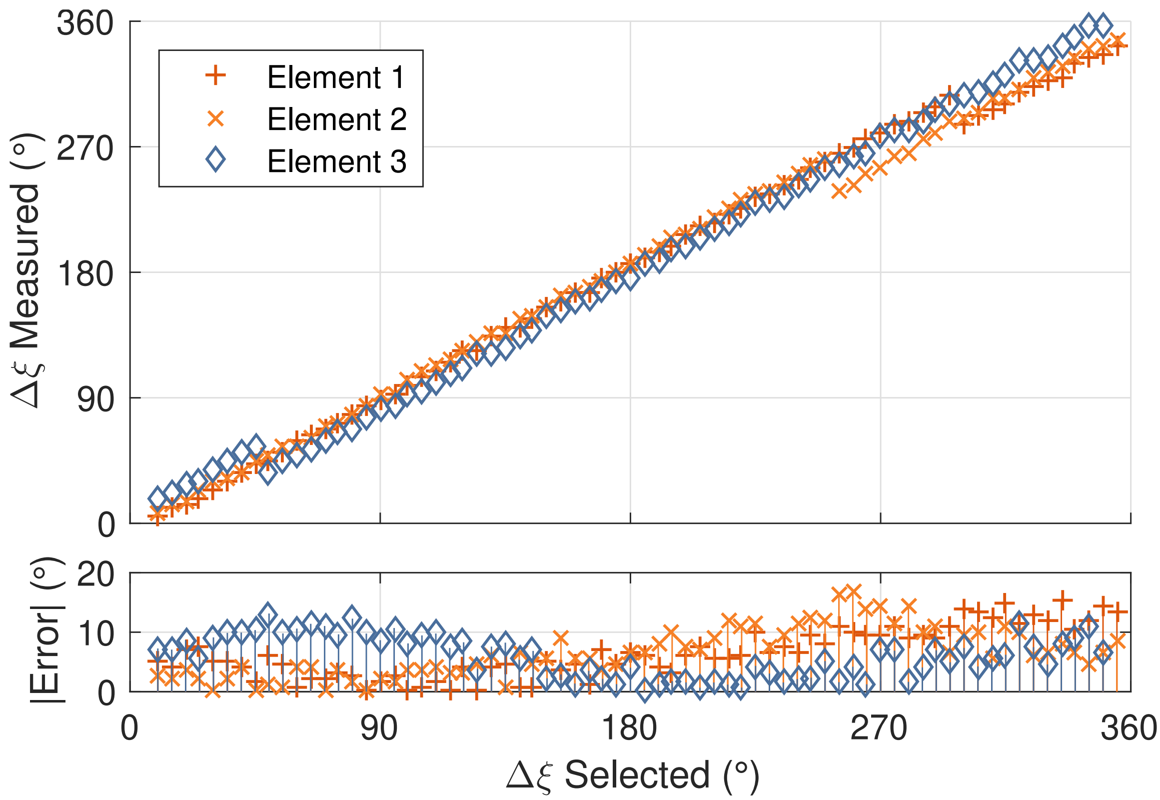


Figure 5.11: Measured transmitted signal phase and error vs. chosen transmitted signal phase

	Element			Overall
	1	2	3	
$\overline{\lvert error \rvert}$	+6.49°	+6.62°	+6.02°	+6.38°
std. dev.($error$)	+7.76°	+7.85°	+7.02°	+7.55°
max($\lvert error \rvert$)	+15.19°	+16.85°	+13.14°	+16.85°

Table 5.4: DLL-HPC transmitted signal phase accuracy statistics by element

As mentioned in Section 5.1, the transmitted signal beamformer phase shifters exhibit a higher standard deviation than the received signal beamformer phase shifters. This manifests itself in the data shown in Fig. 5.11 as larger errors. However, the average error magnitude of $+6.38°$ corresponds to an effective resolution of $12.76°$ and is therefore comparable to a 5-bit digital phase shifter with a resolution of $11.25°$. This indicates that, much like the nonidealities of the direction of arrival calculations in Section 5.4, the errors present in Fig. 5.10 arise from the nonidealities of antenna arrays and the test setup rather than from the DLL-HPC hardware itself.

5.6 Irregular and Changing Array Geometries

The final functionality tested in the anechoic chamber was the DLL-HPC's ability to adapt to irregular and changing antenna array geometries. For this test, the test setups from Fig. 5.6 and 5.9 were used to measure the received signal gain and normalized transmitted array pattern as in Sections 5.3 and 5.5. In order to simulate the possibility of an antenna array mounted to a flexible or variable fixture, the antenna array was first arranged in a half-wavelength linear pattern for initial characterization. In this configuration the Δi and Δj terms were calibrated out, and the HRA was set to its retrodirection mode. Next, the antenna elements were re-positioned according to the pattern shown in Fig. 5.12 and the measurements were taken using the same equipment and strategies as outlined in Sections 5.3 and 5.5 with no other modification to the system.

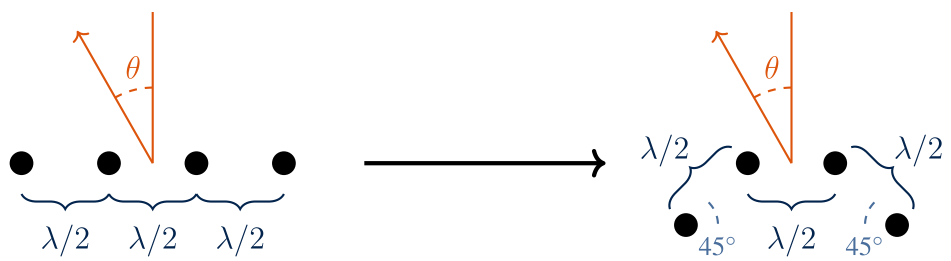


Figure 5.12: Antenna array geometry transformation used to test HRA performance in irregular and variable arrays

Fig. 5.13 shows plots of the received signal gain relative to a single element and the normalized transmitted power across one quadrant of the field of view for the irregular array. As can be seen from Fig. 5.13a, the DLL-HPC received signal beamformer still performs in a near ideal manner. The average received signal power gain across the full field of view is 11.76 dB, which is only a 6.2% error from an ideal 4-element antenna array. The transmitted signal pattern shown in Fig. 5.13b shows a half-power beamwidth of $\pm 50°$. The non-ideal shape of the transmitted array pattern is due to the orientation of the antenna elements themselves and the cascaded topology of the DLL-HPC prototype. The lower right antenna element in Fig. 5.12 was chosen as the 0^{th} element; this means that as the view angle is increased as shown, the 0^{th} and 1^{st} elements' received and transmitted signals are obstructed by the 2^{nd} and 3^{rd} antenna elements. These obstructions by a resonant antenna structure within a few wavelengths of the radiating sources account for the steep drop-off in transmitted signal power beyond view angles of $\pm 30°$.

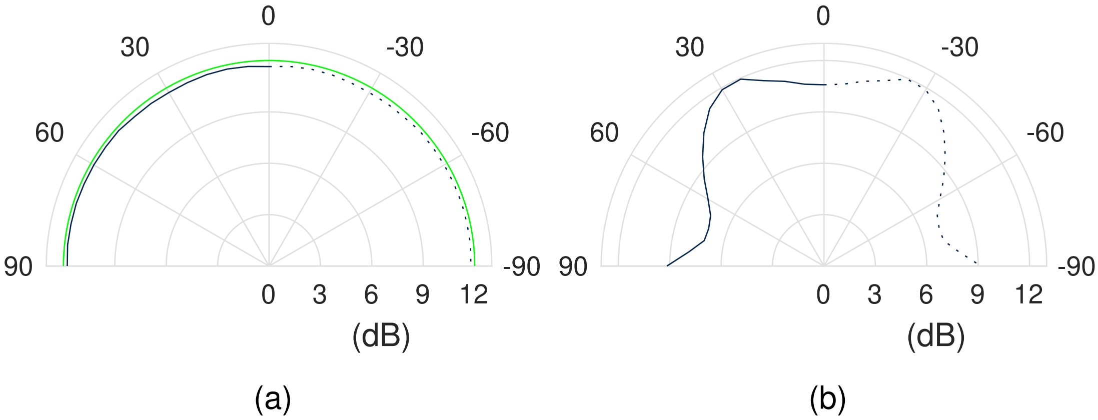


Figure 5.13: Performance of irregular and changing antenna array (a) received signal gain relative to single element (b) transmitted signal power

This test verifies that a DLL-HPC based RDA is capable of maintaining proper calibration and performance even through array geometry variations. However, as shown by the transmitted signal pattern, the physical array layout must still be taken into account and compensated in other ways.

5.7 Phase Conjugation Test

Upon verifying array-level retrodirective functionality in the anechoic chamber, benchtop testing was performed to formally verify the DLL-HPC's ability to accurately perform phase conjugation. The test setup shown in Fig. 5.14 was used to measure each DLL-HPC's ability to accurately conjugate a received signal phase difference. As before, the M8196A AWG was used to generate a known received signal phase difference, while the MSO84 Oscilloscope was used to measure the generated carrier signal phase difference. The received signal phase was swept from $0°$ to $360°$ in $1°$ steps with 10,000 measurements of the generated carrier signal phase value and standard deviation performed for each step. All test results are shown in Fig. 5.15, and relevant statistics are shown in Table 5.5

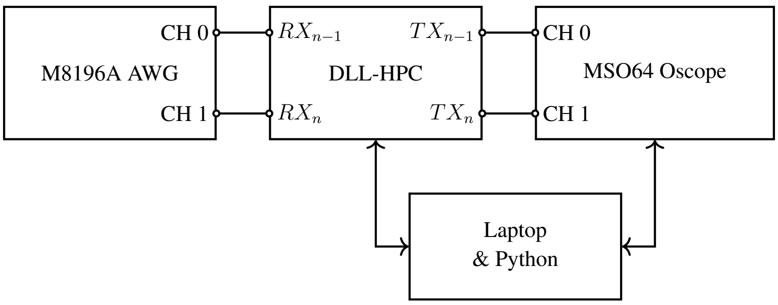


Figure 5.14: Phase conjugation test setup

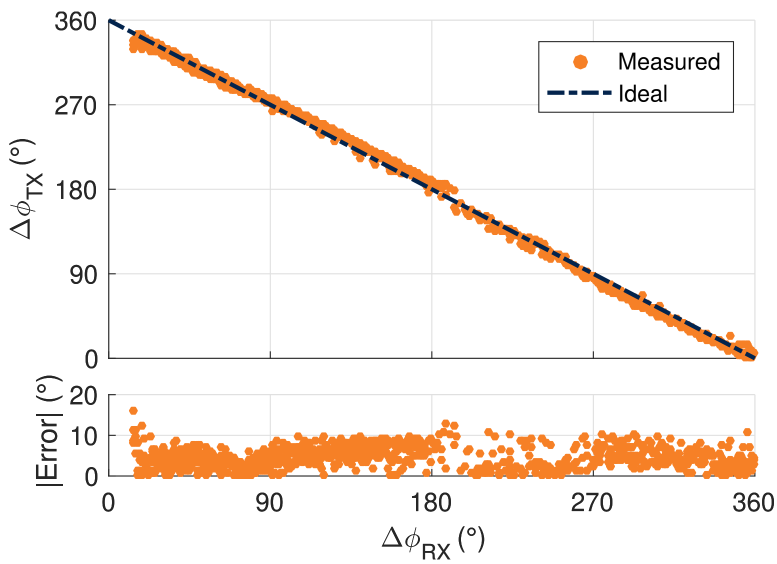


Figure 5.15: Measured phase conjugation and error magnitude vs. received signal phase

	All Elements
$\overline{\lvert error \rvert}$	$+4.65°$
std. dev.($\lvert error \rvert$)	$+2.59°$
max($\lvert error \rvert$)	$+16.17°$

Table 5.5: DLL-HPC phase conjugation test overall statistics

This test shows that the DLL-HPC design is able to accurately perform phase conjugation of received signals. Further, this indicates that the non-idealities found in Sections 5.5 and 5.6 arise from array-level issues, such as inter-element interference. By directly measuring phase conjugation of known signals, the average error magnitude of $+4.65°$ indicates reliable performance with an effective resolution of $9.3°$. This is better than a 5-bit digital phase shifter could achieve with a resolution of $11.25°$. Furthermore, the average standard deviation of $+2.59°$ indicates an extremely stable output signal phase for constant inputs. This shows that the fabricated DLL-HPC prototypes perform reliably as phase conjugators.

5.8 Automatic Mobile Target Tracking

The tests performed in the anechoic chamber so far have already demonstrated automatic mobile target tracking, as the DLL-HPC based HRA was never turned off or reset during rotations. However, this does not provide any sort of numerical validation to the DLL-HPC's ability to track a moving target. In an effort to find the limits of the DLL-HPC's received signal beamformer speed, the test setup in Fig. 5.16 was used. The transmitted signal beamformer provides a time-varying received signal phase difference while the closed-loop DLL control voltage signal is measured to find the settling time of the system.

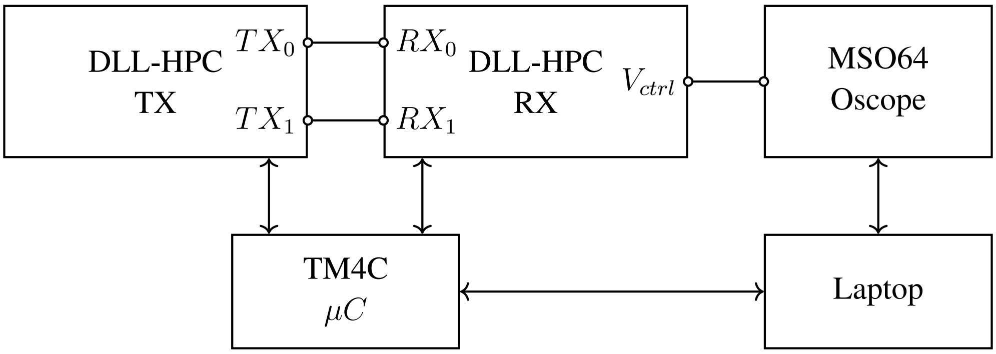


Figure 5.16: Automatic mobile target tracking test setup

To find the response time of the DLL-HPC's closed loop DLL beamformer, step changes were applied to the received signal phase and the system was modeled as a first order system. The magnitude of these step changes was varied from $\pm 15°$ to $\pm 180°$, and the settling time was calculated for each. Fig. 5.17 shows a representative test with the corresponding first-order system values denoted; more measured data sets can be found in Appendix D. The average settling time for the received signal beamformer was found to be 2.26 ms regardless of the input step size, which is 50x faster than the current IEEE standard 802.11ad/ay acquisition used for WiFi beamformers [11].

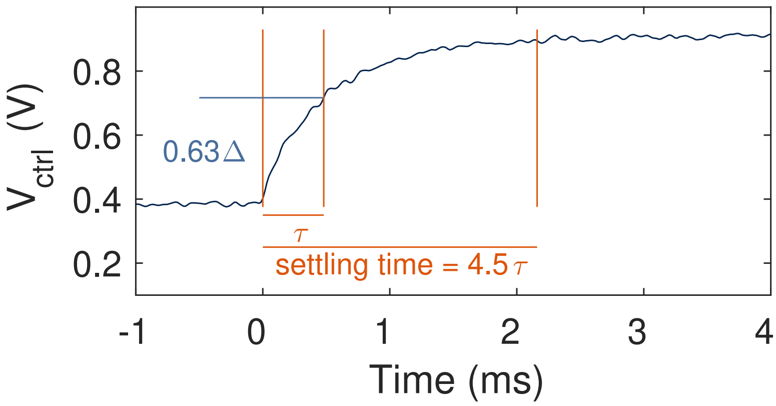


Figure 5.17: DLL-HPC received signal beamformer closed-loop settling time for 180° step input

The settling time of the transmitted signal beamformer was found to be 10.31 ms in the same tests. The difference in received and transmitted signal beamformer settling times is due to the component selection and firmware design. Because the sampling rate of the system is set to 100 Hz as described in Section 4.5.4, the transmitted signal beamformer is only updated once every sample, or every 10 ms. This agrees well with the measured results, which show the transmitted signal beamformer settling about one sample later. This number could be drastically reduced by changing the design: faster DACs and a faster sampling rate would allow the transmitted signal settling time to approach the closed-loop DLL settling time, which is the limiting factor in this situation. Further, the closed-loop DLL settling time can be reduced through PFD/CP loop filter design.

In addition to measuring the system settling time, the DLL-HPC received signal beamformer was tested under extreme variations in the received signal phase. The same test setup from Fig. 5.16 was used to provide a received signal with a constantly varying received signal phase that swept from 0° to 360° to 0° in a triangular pattern. The system was tested with rates of change from 50 °/s to 30,000 °/s and was found to track well in all tests. Fig. 5.18 shows the measured DLL control voltage for the 50 °/s and 15,000 °/s test, with more measured data sets in Appendix D

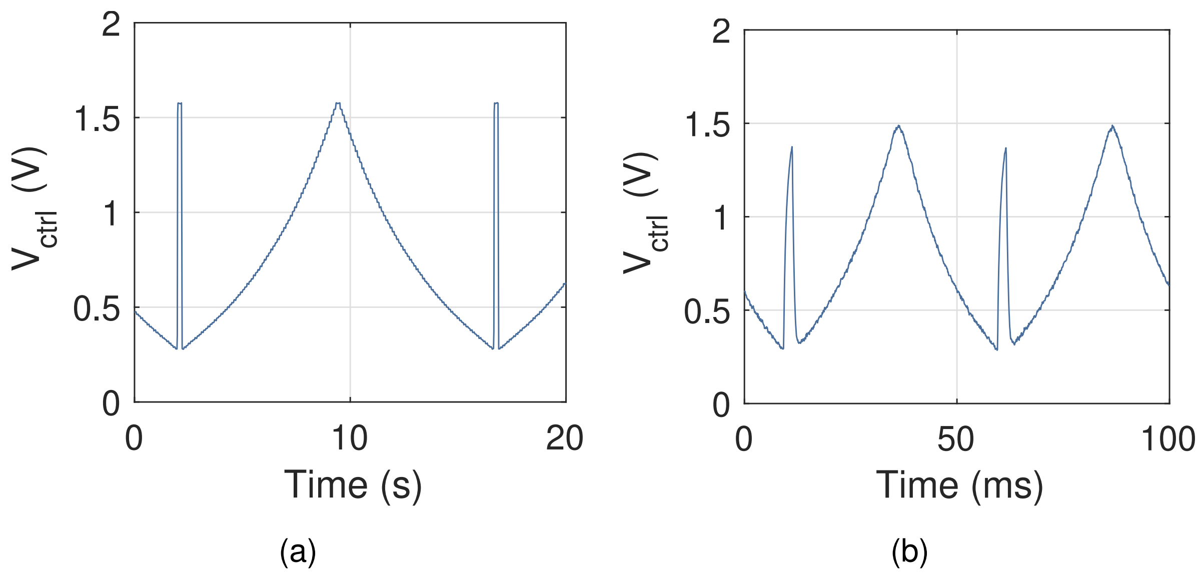


Figure 5.18: DLL-HPC received signal beamformer closed-loop control voltage tracking (a) 50 °/s received signal (b) 15,000 °/s received signal

From these tests we can conclude that a DLL-HPC based HRA will be able to accurately and automatically track any reasonable real-world user. As a point of reference, a car driving at 75 mph (120 kmph) past a cell tower 100 ft (30.5 m) from the road would have a maximum angle of arrival change of 63.1 °/s, as illustrated in Fig. 5.19. This would correspond to a maximum rate of change in the received signal phase of 31.5 °/s for a half-wavelength spaced linear antenna array. As the DLL-HPC has been shown to accurately track and provide ideal beamformer gain for signals with relative phases varying 952 times faster than this peak value, it is safe to assume that a DLL-HPC could handle any reasonable real-world application.

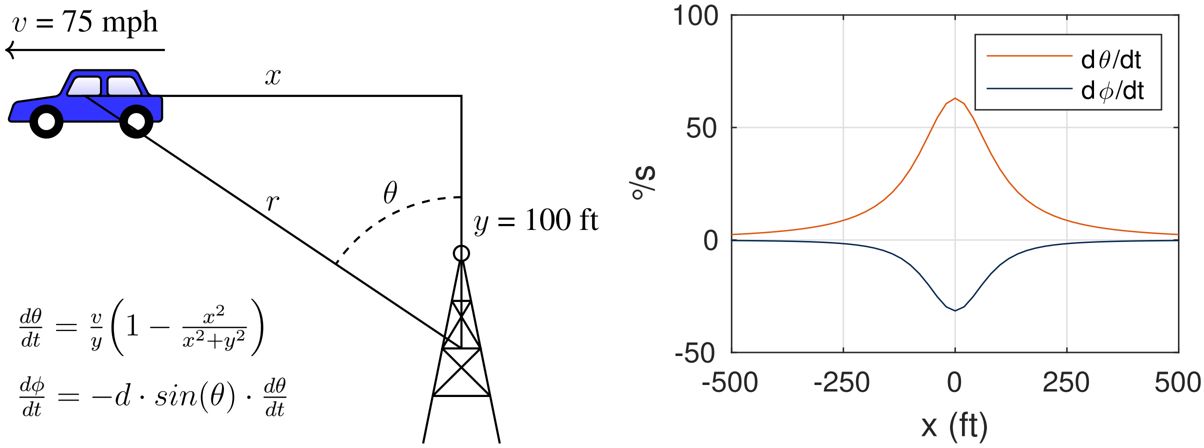


Figure 5.19: Angle of arrival and relative phase change over time for a fast moving mobile user, assuming a half-wavelength linearly spaced antenna array

Chapter 6

Integrated Circuit Designs for Unique Components

The DLL-HPC design presented in Chapter 4 has been verified in its different functionalities by the tests in Chapter 5. However, this prototype was constructed of entirely COTS components at the PCB scale and serves only as a proof-of-concept prototype. In order to provide a path forward for miniaturization of a DLL-HPC, this chapter presents the transistor-level design and simulation of a Rollover Phase-Frequency Detector (R-PFD) using the IBM 8HP PDK: a 120nm SiGe process.

Because all other system-level components of the novel DLL-HPC design are standard RF components (mixers, amplifiers, oscillators, filters, etc.), only the R-PFD is designed and simulated in this dissertation. Should an integrated-circuit level design of the DLL-HPC be carried out one day, this chapter should provide all of the supplemental information needed to complete a design. Section 6.1 provides the motivation for designing the R-PFD, Section 6.2 presents the circuit's design, Section 6.3 details the non-standard cell designs used, and Section 6.4 presents system-level simulations and verification. Additional design information and simulations for sub-system components can be found in Appendix E.

6.1 Motivation for A Rollover Phase-Frequency Detector

DLLs have many applications, including being the basis for the novel DLL-HPC presented in this dissertation. They have been used as the basis for clock signal synchronization in complex systems [64, 65] and as all-digital frequency synthesizers [66–69]. More recently, DLLs have been used as the basis for RF beamformers by providing in-phase signals to be summed [27,56], much like the DLL used as the basis of the DLL-HPC. However, DLL-based RF beamformers face a problem: a DLL is only unconditionally stable for fixed phase differences [53].

In a dynamic RF environment the angle of arrival of each communication link varies with time [15,16,21], which will be manifested at the inputs of a DLL beamformer as a change in the relative phase over time. A time varying phase difference between signals is mathematically equivalent to a frequency difference, which cannot be accommodated in a traditional PFD based DLL [53]. Fig. 6.1 shows that a frequency difference between the inputs of a PFD-based DLL will cause an unstable and inaccurate control voltage, as the loop-filter voltage will saturate as it approaches the system power rails.

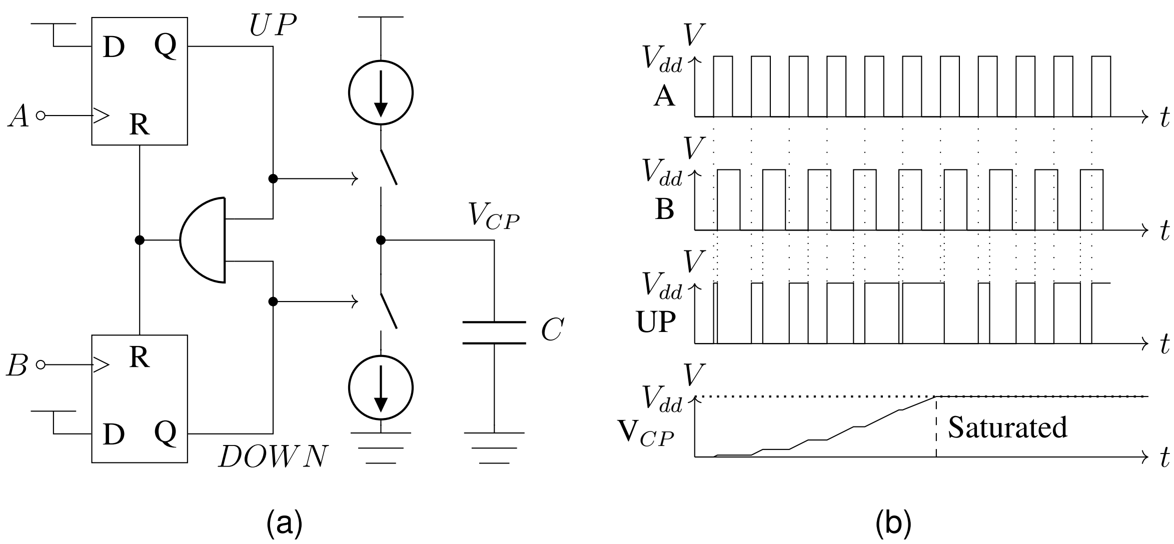


Figure 6.1: (a) Traditional PFD and charge pump system (b) DLL operation for inputs of different frequencies

Because the rapid settling time of the DLL-HPC shown in Section 5.8 is entirely dependent on a PFD-CP design, a R-PFD which will not saturate at the power rails is deemed a necessary step towards the miniaturization of a DLL-HPC based HRA. Without a R-PFD, the HRA would lose the ability to accurately perform phase conjugation or direction of arrival reporting as the phase-delay control voltages saturate and become inaccurate. All other components of the proposed DLL-HPC system are standard RF components that can be found in textbooks and current literature; however, there is currently no published design of a R-PFD.

6.2 Rollover Phase-Frequency Detector Design

The additional functionality needed to implement a R-PFD is conceptually very simple. We first select an upper and lower threshold, V_{high} and V_{low}, that we would like the charge pump output voltage V_{CP} to stay within. We then only need to check if V_{CP} is outside of these limits and if the PFD is currently driving V_{CP} farther from these limits through the UP and $DOWN$ charge pump control signals. If this is the case, then we can use separate charge pump control signals called $RISE$ and $FALL$ to raise or lower V_{CP} until it reaches the other threshold. This process is outlined in Algorithm 2.

Algorithm 2: R-PFD Theoretical Operation

```
while True
    if (V_CP > V_high) && UP
        FALL = 1;
        while V_CP > V_low
            Wait
        FALL = 0;
    else if (V_CP < V_low) && DOWN
        RISE = 1;
        while V_CP < V_high
            Wait
        RISE = 0;
```

Algorithm 2 can be easily implemented using discrete logic gates. Each comparison can be implemented with a simple analog comparator or digital logic circuit. Furthermore, the $RISE$ and $FALL$ signals can be implemented using standard flip-flops and digital memory elements, as these signals only change when another signal changes; this is analogous to a flip-flop or latch only changing on a rising or falling edge of a clock signal. If Algorithm 2 is implemented alongside the PFD-CP circuitry from Fig. 6.1, then the charge pump output voltage V_{CP} would exhibit sawtooth behavior rather than saturate, as illustrated in Fig. 6.2.

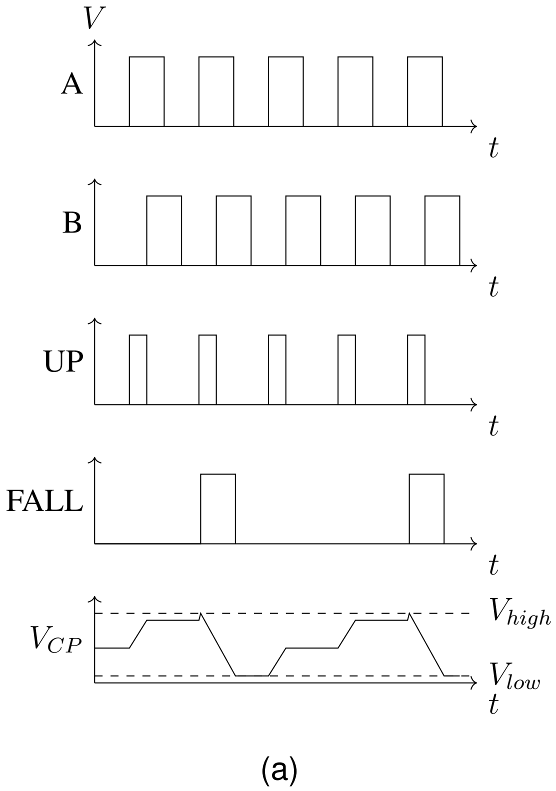


Figure 6.2: A R-PFD will exhibit sawtooth behavior rather than saturating behavior when (a) V_{CP} approaches upper threshold (b) V_{CP} approaches lower threshold

Fig. 6.3 shows a rollover feedback circuit implementation of Algorithm 2 using standard logic gates. The circuit makes use of two comparators to compute $V_{CP} > V_{high}$ and $V_{CP} < V_{low}$ at nodes A and B respectively. If an UP pulse occurs while $V_{CP} > V_{high}$, or simply $A = 1$, then the $FALL$ signal will be latched to a logical 1. This $FALL$ signal will be utilized to force the charge pump to sink current and drive V_{CP} lower until it reaches the lower threshold and $V_{CP} < V_{low}$. Once this occurs and $B = 1$, the flip-flop holding the $FALL$ signal high will be cleared and normal PFD operation will resume. A mirrored sequence of events will take place if a $DOWN$ pulse occurs while $V_{CP} < V_{low}$, as can be seen from the symmetry of the circuit design. Furthermore, the entire rollover feedback circuit can be enabled or disabled by a single logic signal as shown.

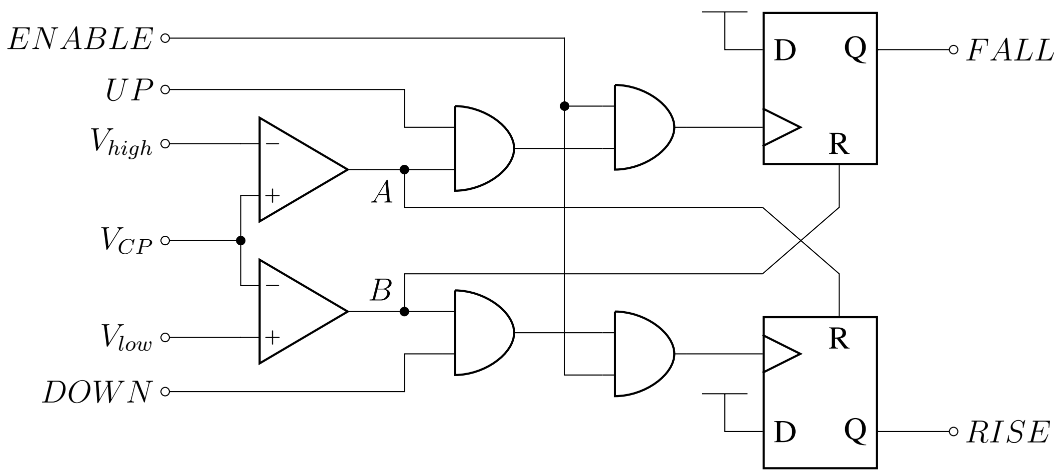


Figure 6.3: A rollover feedback circuit to implement Algorithm 2

Fig. 6.4 shows the full R-PFD system schematic. The rollover feedback circuit from Fig. 6.3 is used to augment a standard PFD-CP circuit to provide rollover functionality. The $RISE$ and $FALL$ signals generated by the rollover feedback circuit are logically OR'ed with the UP and $DOWN$ signals from a standard PFD to control the charge pump properly. Additionally, the $RISE$ and $FALL$ signals are each capable of disabling the PFD's D flip-flops when they are high to prevent inaccurate behavior. When either the $RISE$ or $FALL$ signal is high, the charge pump's output voltage V_{CP} will be varying continuously; as this is what controls the phase delay in a standard DLL, this means that the relative phase of the two signals A and B will be varying as well. If the PFD is allowed to operate during this sliding, spurious UP and $DOWN$ pulses may be generated that waste power and cause the rollover to take more time. Additionally, if the threshold voltages are chosen such that they are one full phase rotation, or 360°, apart then the polarity of the PFD output signals may become swapped. For these reasons, the $RISE$ and $FALL$ signals are logically OR'ed with a standard PFD's reset line in order to disable the PFD during rollover.

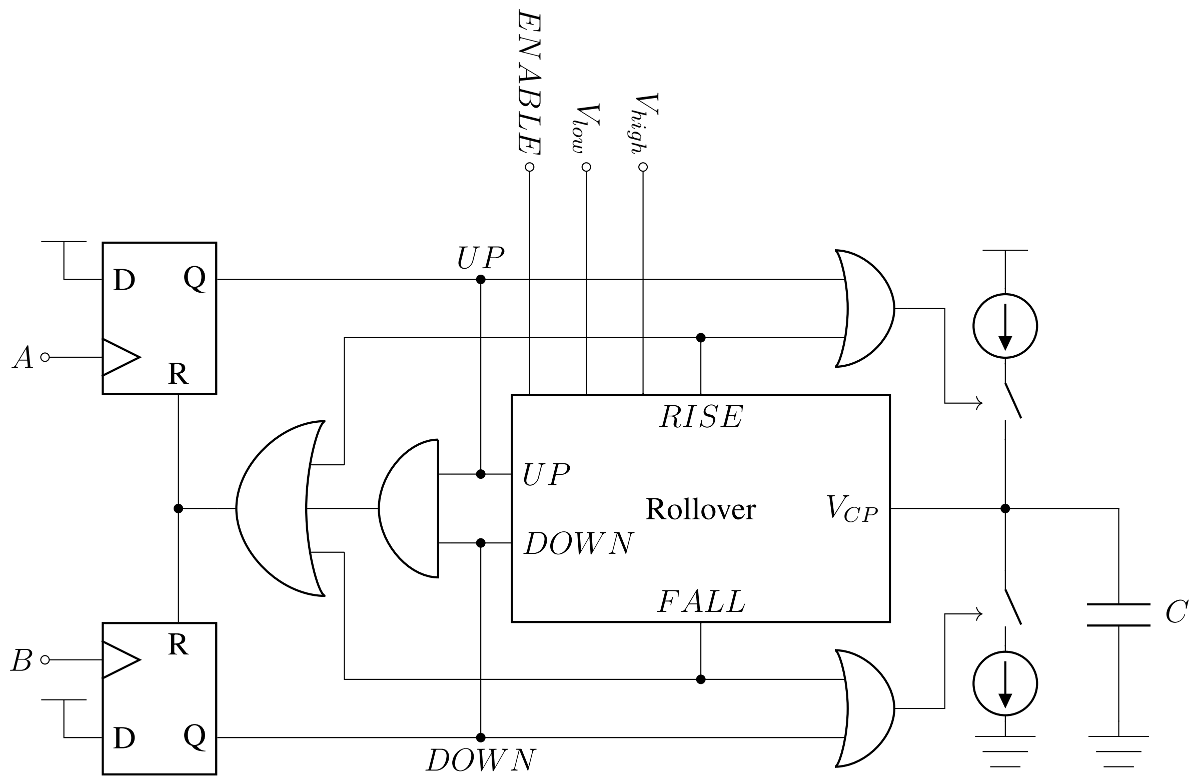


Figure 6.4: Rollover Phase-Frequency Detector (R-PFD) system schematic

6.3 Rollover Phase-Frequency Detector Implementation

A transistor-level design was made using Cadence Virtuoso and the IBM 8HP PDK to simulate a 120nm SiGe process. Each system component was designed from individual transistors, including basic logic gates. The individual logic gates were designed to have balanced rise and fall times for all input transitions and were used to create more complex digital blocks, such as flip-flops and edge detectors. As CMOS digital logic circuits are relatively simple to design and understand, the detailed schematics of the basic logic gates used in this design can be found in Appendix E.1. This section details the design of the few non-standard components included in the R-PFD design shown so far: the comparator, charge pump, and voltage-controlled delay line that will be used for system-level DLL simulations.

It is important to mention that the R-PFD design presented in Figures 6.3 and 6.4 is almost entirely composed of common standard-cell components. This means that the proposed R-PFD design could easily be adopted by any standard-cell based design, and the non-standard designs presented in this section are not required. Indeed, any future implementations of this R-PFD should be done with specific comparator, charge pump, and voltage-controlled delay designs tailored to the specific design.

6.3.1 Comparator Design

Fig. 6.5 shows the transistor-level schematic for the implemented comparators, which function as Class A operational amplifiers. Each comparator is composed of three stages: a current-mode logic (CML) inverter, a differential amplifier, and a CMOS logic inverter that buffers the output. The CML inverter is used as the first stage to provide a high input impedance, fast response time, and differential output voltage with a constant DC offset. This CML inverter output is then fed into a differential amplifier, which drives a CMOS inverter to buffer the output for use as a logic signal. As the signal is inverted twice, the polarity of the input is maintained through the circuit. Each comparator consumes approximately 3.2 μm^2 and 82 μW of power, making it the single largest and most power-hungry component in the R-PFD design. However, this large area and power consumption allow the circuit to respond quickly to changes in the two inputs. Detailed simulations of the comparator and its operating points can be found in Appendix E.2

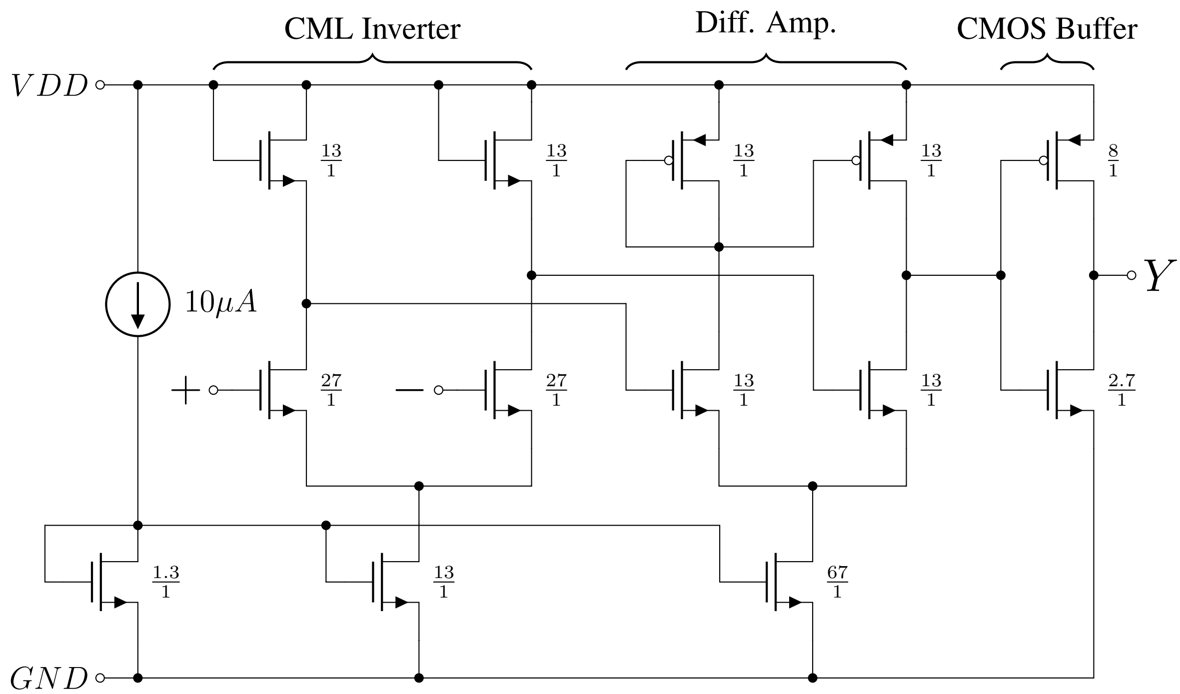


Figure 6.5: Comparator design with transistor W/L ratios listed

6.3.2 Charge Pump Design

Fig. 6.6 shows the transistor-level schematic for the implemented charge pump. The charge pump's functionality is relatively simple to understand: when either the UP or $DOWN$ input signal is high, current is allowed to flow through one leg of the initial high input impedance stage. This current is then mirrored by either a PMOS or NMOS current mirror design to sink or source current through the I_{CP} output terminal; different W/L ratios have been chosen for these output current mirrors to account for differences in electron mobility through the NMOS and PMOS devices. The charge pump consumes 12.7 μm^2 of area and 1.3 μW of power when inactive and 1.324 mW when active. The design sinks or sources 1.5 mA when either UP or $DOWN$ is enabled. Detailed simulations of the charge pump and its operating points can be found in Appendix E.3.

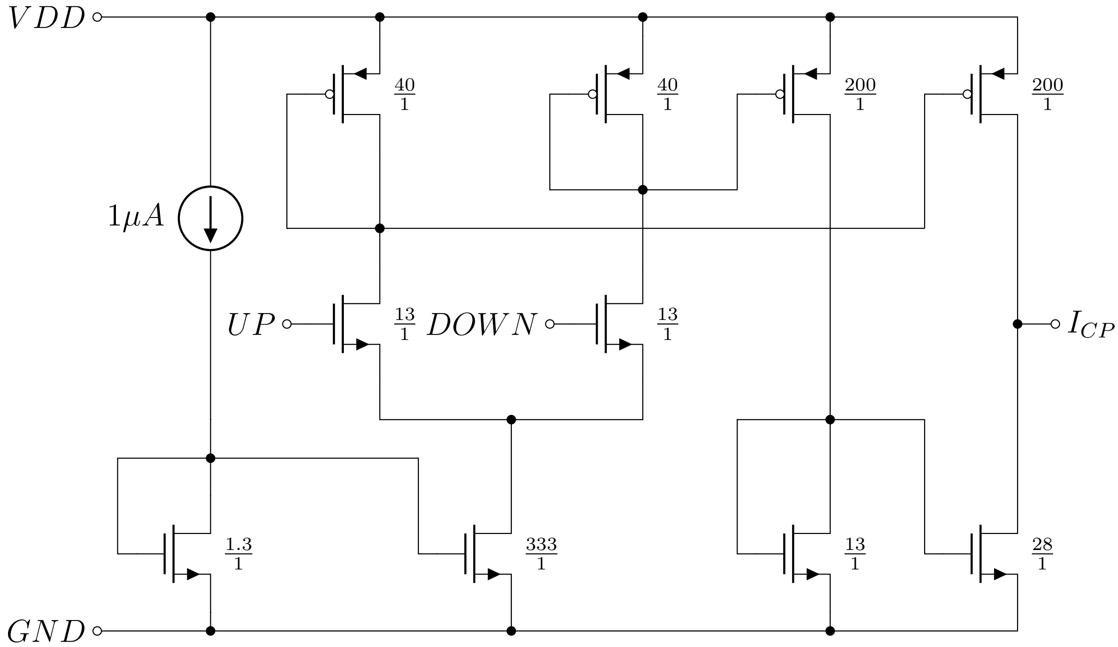


Figure 6.6: Charge pump design with transistor W/L ratios listed

6.3.3 Voltage Controlled Delay Line Design

In order to perform system-level simulations of a DLL, a voltage controlled delay line (VCDL) was needed. Fig. 6.7 shows the transistor-level schematic for the implemented VCDL: a current-starved inverter chain. This design is based on the VCDL presented in [65] and operates on the principle of controlling the amount of current allowed to a chain of CMOS inverters. As the control voltage V_{ctrl} is increased, more current is allowed to flow through each inverter and the rise/fall time of each increases accordingly. A chain of 80 inverters was chosen to provide a controllable delay of 2 ns for an input range of $0.6 - 1.0$ V, allowing system simulations to be performed with a 200 MHz signal. The design consumes 25.1 μm^2 of area and 270 μW of power on average. Detailed simulations of the VCDL and its operating points can be found in Appendix E.4.

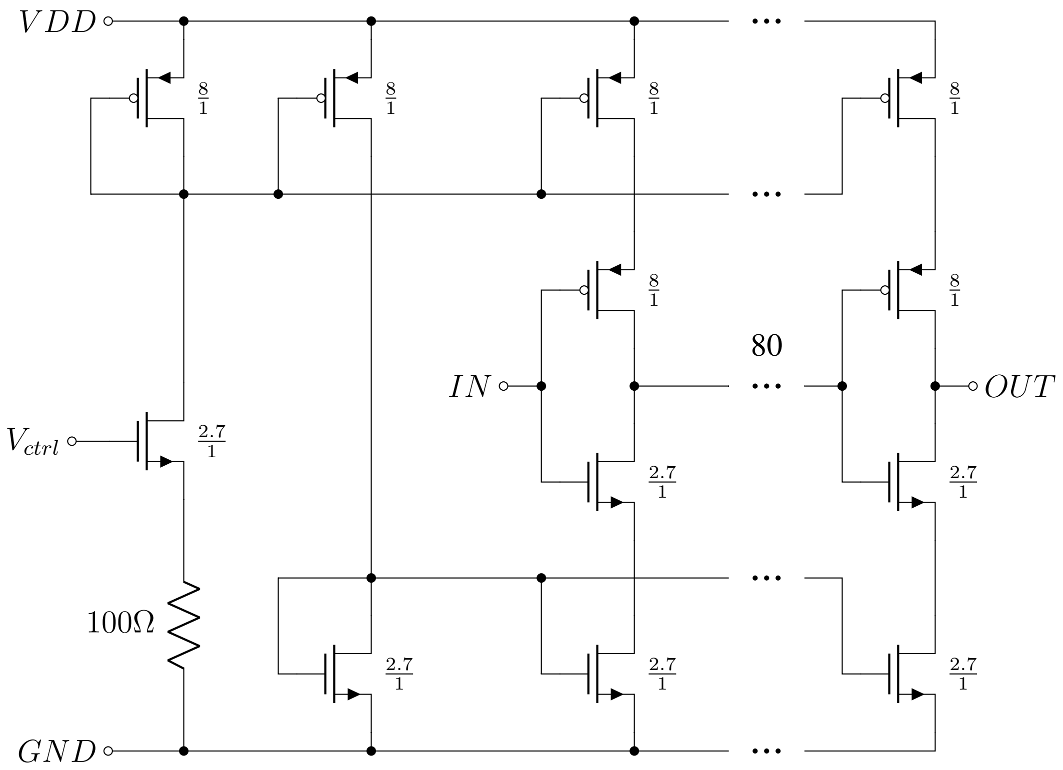


Figure 6.7: Voltage controlled delay line design with transistor W/L ratios listed

6.4 Rollover Phase-Frequency Detector Simulations

Both open-loop and closed-loop simulations of a R-PFD based DLL were performed to verify system-level operation. Fig. 6.8 show the open- and closed-loop systems simulated, while Table 6.1 shows the area and average power consumption of each system-level component. Open-loop simulations were used to verify that the R-PFD behaves as anticipated and provides accurate rollover functionality at the chosen voltage thresholds. Once open-loop R-PFD functionality was verified, tests were performed to ensure that the R-PFD allows for tracking of time-varying phase differences, or frequency differences, between inputs.

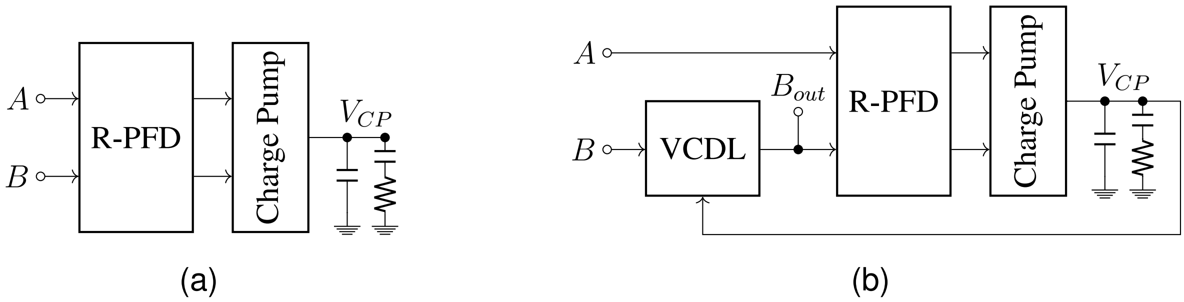


Figure 6.8: (a) open-loop R-PFD System (b) closed-loop R-PFD DLL

Component	**Area**	**Avg. Power**
R-PFD	20.0 μm^2	176.8 μW
Charge Pump	12.7 μm^2	35μW
VCDL	25.1 μm^2	270 μW

Table 6.1: Area and power consumption by system-level component

Fig. 6.9 shows simulations of the open-loop system from Fig. 6.8 with 205 MHz and 200 MHz square wave sources used for inputs A and B. Fig. 6.9a shows that when input A is set to a higher frequency than B the charge pump voltage V_{CP} will roll over from the upper threshold to the lower; Fig. 6.9b shows that the opposite is true when B has a higher frequency than A. These simulations verify that open-loop operation of the R-PFD agrees well with the desired functionality outlined in Section 6.2. However, slight non-idealities can be seen when V_{CP} is at or near the threshold voltages: these small periods of time where V_{CP} exists outside the chosen thresholds is due to the gate delays of the rollover feedback circuitry and can be avoided by a low operating frequency or faster logic gate design. The average power consumption by the R-PFD for all open-loop simulations was 176.8 μW; more simulations can be found in Appendix E.5.

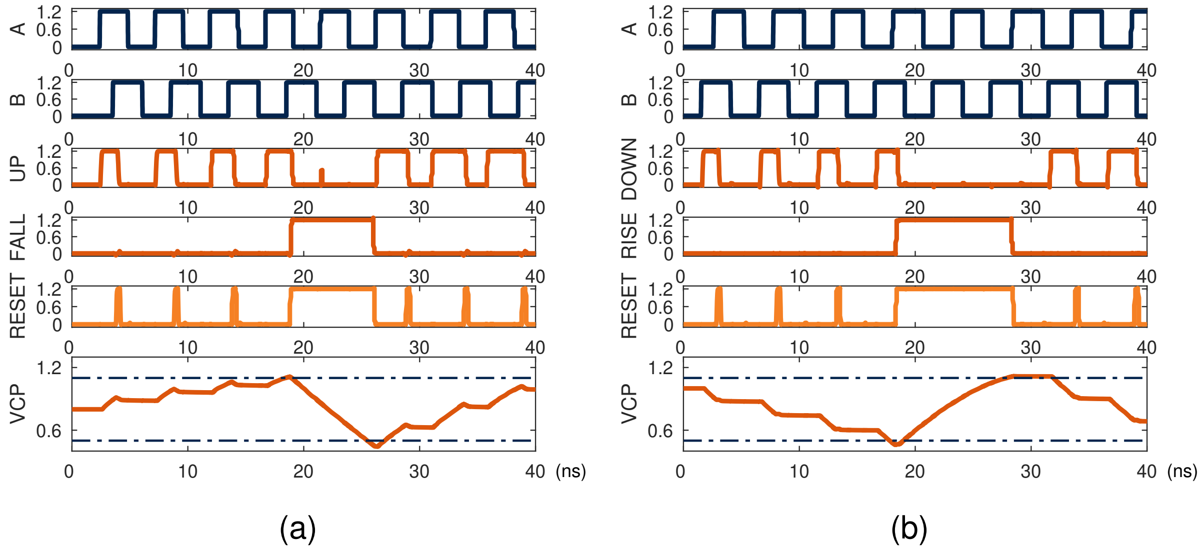


Figure 6.9: Open-loop R-PFD simulations for (a) $f_a > f_b$ (b) $f_a < f_b$

Fig. 6.10 shows a sample simulation of the closed-loop R-PFD based DLL, with B_{out} presented alongside the control voltage V_{CP}. In this large time-scale simulation, a 200 MHz signal is used as the source for B and a 205 MHz signal is used as the source for A. This simulation shows that the R-PFD enables a delay-locked loop, which is traditionally incapable of tracking a frequency difference [53], to track a signal at a different frequency. This frequency difference is mathematically equivalent to the time-varying phase difference that occurs when beamforming towards a mobile target [29]; however this simulated 5 MHz frequency difference corresponds to a relative phase difference changing at 1.8 billion $^{\circ}/s$, which far exceeds any real world system. This large frequency difference was chosen to provide a reasonable simulation time and results, but simulation time step artifacts can still be seen in Fig. 6.10 in the form of slight discontinuities in the B_{out} signal. In addition to tracking a frequency difference, the R-PFD based DLL can be used to determine the instantaneous frequency difference between the A and B input signals through measurement of the control voltage, as each point within the chosen range corresponds to a single phase difference value. The R-PFD consumed an average 176.8 μW of power during all closed-loop system simulations; more simulations can be found in Appendix E.6.

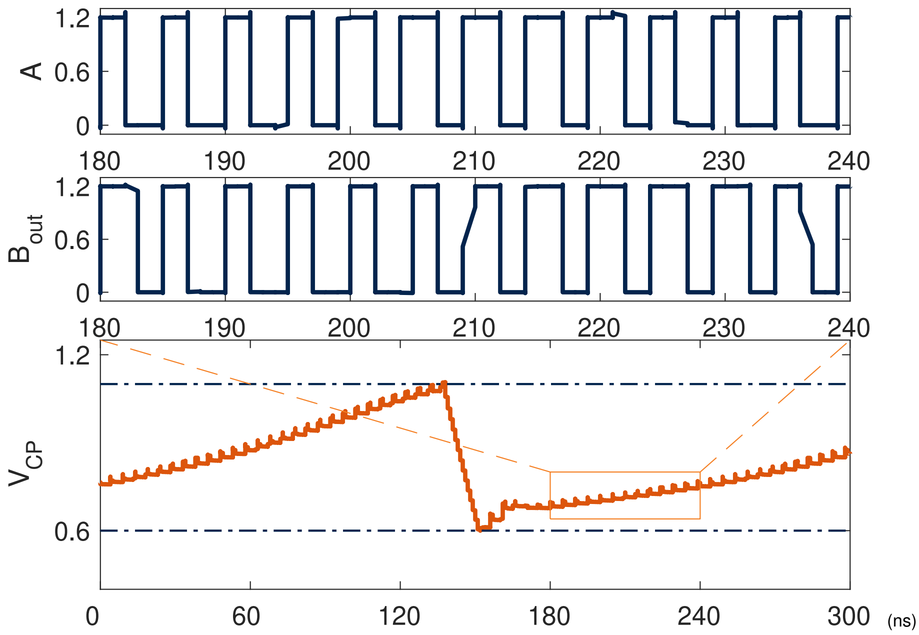


Figure 6.10: R-PFD based DLL tracking a 205 MHz signal with a 200 MHz signal

The functionality of a R-PFD based DLL has great implications for the future of DLL-based RF beamformers. A traditional DLL beamformer can provide ideal received signal gain by bringing IF signals in-phase before a summing circuit [27, 56, 70], but only in the case of fixed received signal phase differences [53]. A R-PFD based DLL will allow these beamformer techniques to be extended to continuously varying phase differences, such as those encountered in mobile wireless connections. Further, the confined control voltage range of a R-PFD based DLL means that the instantaneous phase difference between signals can be calculated from this voltage and used to determine the direction of arrival of signals, much like the RDA design presented in this dissertation. The adoption of R-PFD DLL beamformers could benefit context-aware communication schemes, such as SDMA [55].

Chapter 7

Conclusions

This dissertation has presented a novel delay locked loop based hybrid phase conjugator (DLL-HPC) and prototypical hybrid retrodirective array (HRA). Fundamental concepts for understanding the operation of a HRA were presented in Chapters 2 and 3. The system design for a single DLL-HPC and its operation were discussed in depth in Chapter 4 before a proof-of-concept HRA was constructed and fully characterized in Chapter 5. The unique functionalities of a DLL-HPC based HRA were discussed in theory and verified through laboratory testing, serving as a starting point for future research into HRA based communication systems.

Test results shown in Chapter 5 show the novelty and improvements of the HRA when compared to existing beamformer technologies. The HRA outperforms current beamforming standards, such as IEEE 802.11ad/ay, by a factor of up to 50 with regard to initial acquisition time and requires no communication overhead to maintain ideal beamformer operation. Additionally, this is the first HRA to provide full backwards compatibility with existing beamforming standards. These improvements make the proposed DLL-HPC based HRA a feasible first step to integrating retrodirective technology into mobile communication networks.

In addition to a novel DLL-HPC and HRA, a novel rollover phase-frequency detector (R-PFD) design was presented as an enabling technology for DLL based RF beamformers in Chapter 6. A standard-cell friendly R-PFD design was presented alongside a transistor-level design using the IBM 8HP 120 nm SiGe process. Simulation results for both open- and closed-loop operation were presented to demonstrate that a R-PFD based DLL is capable of tracking continuously varying relative phase differences, enabling a new class of RF beamformers.

This work towards the adoption and miniaturization of hybrid retrodirective arrays is intended to serve as a starting point for future efforts. Much work is still needed to reduce system complexity, miniaturize components to an acceptable scale for mmWave 5G systems, and develop new network protocols and strategies to allow for full HRA adoption. However, the potential benefits of HRAs make this effort worthwhile: automatic lock-on, automatic mobile-target tracking, direction of arrival reporting, and backwards compatibility make the HRA a strong candidate for improving future networks.

References

[1] P. Hindle, “History of Wireless Communications,” 2015.

[2] Samsung, “Samsung Family Hub,” 2020.

[3] “Hidrate Spark,” 2020.

[4] Amazon, “AmazonBasics Microwave,” 2020.

[5] K. Westcott, J. Loucks, D. Littmann, P. Wilson, S. Srivastava, and D. Ciampa, “Build it and they will embrace it Consumers are preparing for 5G connectivity in the home and on the go,” tech. rep., Deloitte Center for Technology, Media, & Telecommunications, 2019.

[6] D. Ernst, “Changing Dynamics of the Smart Home: Opportunities for Service Providers A Parks Associates Whitepaper Developed for Calix,” tech. rep., Parks Associates, 2019.

[7] “Video Streaming Market Worth $184.3 Billion By 2027 — CAGR: 20.4%,” 2020.

[8] L. Wood, “Global 5G Services Market Report 2019,” 2019.

[9] T. S. Rappaport, S. Sun, R. Mayzus, H. Zhao, Y. Azar, K. Wang, G. N. Wong, J. K. Schulz, M. Samimi, and F. Gutierrez, “Millimeter wave mobile communications for 5G cellular: It will work!,” *IEEE Access*, vol. 1, pp. 335–349, 2013.

[10] M. Cheffena, “Industrial wireless communications over the millimeter wave spectrum: Opportunities and challenges,” *IEEE Communications Magazine*, vol. 54, pp. 66–72, 9 2016.

[11] P. Zhou, K. Cheng, X. Han, X. Fang, Y. Fang, R. He, Y. Long, and Y. Liu, "IEEE 802.11ay-Based mmWave WLANs: Design challenges and solutions," *IEEE Communications Surveys and Tutorials*, vol. 20, no. 3, pp. 1654–1681, 2018.

[12] W. Roh, J. Y. Seol, J. H. Park, B. Lee, J. Lee, Y. Kim, J. Cho, K. Cheun, and F. Aryanfar, "Millimeter-wave beamforming as an enabling technology for 5G cellular communications: Theoretical feasibility and prototype results," *IEEE Communications Magazine*, vol. 52, pp. 106–113, 2 2014.

[13] P. Zhou, X. Fang, Y. Fang, Y. Long, R. He, and X. Han, "Enhanced random access and beam training for millimeter wave wireless local networks with high user density," *IEEE Transactions on Wireless Communications*, vol. 16, pp. 7760–7773, 12 2017.

[14] M. Giordani, M. Mezzavilla, and M. Zorzi, "Initial Access in 5G mmWave Cellular Networks," *IEEE Communications Magazine*, vol. 54, pp. 40–47, 11 2016.

[15] S. Singh, R. Mudumbai, and U. Madhow, "Interference analysis for highly directional 60-GHz mesh networks: The case for rethinking medium access control," *IEEE/ACM Transactions on Networking*, vol. 19, pp. 1513–1527, 10 2011.

[16] Y. M. Tsang and A. S. Poon, "Detecting human blockage and device movement in mmwave communication system," in *GLOBECOM - IEEE Global Telecommunications Conference*, pp. 1–6, 12 2011.

[17] R. Ludwig and P. Bretchko, {*RF*} *Circuit Design, Theory and Applications*. NJ: Prentive Hall, 2000.

[18] L. Wei, R. Hu, Y. Qian, and G. Wu, "Key elements to enable millimeter wave communications for 5G wireless systems," *IEEE Wireless Communications*, vol. 21, pp. 136–143, 12 2014.

[19] M. Jacob, C. Mbianke, and T. Kürner, "A dynamic 60 GHz radio channel model for system level simulations with MAC protocols for IEEE 802.11ad," in *Proceedings of the International Symposium on Consumer Electronics, ISCE*, pp. 1–5, 6 2010.

[20] H. Shokri-Ghadikolaei, L. Gkatzikis, and C. Fischione, "Beam-searching and transmission scheduling in millimeter wave communications," in *IEEE International Conference on Communications*, vol. 2015-Septe, pp. 1292–1297, 6 2015.

[21] B. Gao, Z. Xiao, C. Zhang, L. Su, D. Jin, and L. Zeng, "Double-link beam tracking against human blockage and device mobility for 60-GHz WLAN," in *IEEE Wireless Communications and Networking Conference, WCNC*, pp. 323–328, 4 2014.

[22] V. Atta, "Electromagnetic reflector," 10 1959.

[23] C. Y. Pon, "Retrodirective Array Using the Heterodyne Technique," *IEEE Transactions on Antennas and Propagation*, vol. 12, pp. 176–180, 3 1964.

[24] N. B. Buchanan and V. F. Fusco, "Triple mode PLL antenna array," in *IEEE MTT-S International Microwave Symposium Digest*, vol. 3, pp. 1691–1694, 6 2004.

[25] N. B. Buchanan, V. F. Fusco, and M. Van Der Vorst, "SATCOM Retrodirective Array," *IEEE Transactions on Microwave Theory and Techniques*, vol. 64, pp. 1614–1621, 5 2016.

[26] P. V. Brennan, "An Experimental and Theoretical Study of Self-Phased Arrays in Mobile Satellite Communications," *IEEE Transactions on Antennas and Propagation*, vol. 37, pp. 1370–1376, 11 1989.

[27] Y. Ding, N. B. Buchanan, V. F. Fusco, R. Baggen, M. Martínez-Vázquez, and M. Van Der Vorst, "Analog/Digital Hybrid Delay-Locked-Loop for K/Ka Band Satellite Retrodirective Arrays," *IEEE Transactions on Microwave Theory and Techniques*, vol. 66, pp. 3323–3331, 7 2018.

[28] L. Sevgi and C. Ulusk, "A MATLAB-based visualization package for planar arrays of isotropic radiators," *IEEE Antennas and Propagation Magazine*, vol. 47, pp. 156–163, 2 2005.

[29] W. L. Stutzman and G. A. Thiele, “Antenna Theory and Design,” in *IEEE Antennas and Propagation Society Newsletter*, vol. 23, ch. 8, pp. 40–41, 111 River Street, Hoboken, NJ 07030-5774: John Wiley & Sons Inc., 3 ed., 1981.

[30] S. Wentworth, *Applied Electromagnetics: Early Transmission Lines Approach*. Hoboken, NJ: John Wiley & Sons Inc., 2007.

[31] C. Jeong, J. Park, and H. Yu, “Random access in millimeter-wave beamforming cellular networks: Issues and approaches,” *IEEE Communications Magazine*, vol. 53, pp. 180–185, 1 2015.

[32] V. Desai, L. Krzymien, P. Sartori, W. Xiao, A. Soong, and A. Alkhateeb, “Initial beamforming for mmWave communications,” in *Conference Record - Asilomar Conference on Signals, Systems and Computers*, vol. 2015-April, pp. 1926–1930, IEEE Computer Society, 4 2015.

[33] A. Capone, I. Filippini, and V. Sciancalepore, “Context Information for Fast Cell Discovery in mm-wave 5G Networks - VDE Conference Publication,” in *Proceedings of European Wireless 2015; 21th European Wireless Conference*, 2015.

[34] S. Perkins, “Teen’s invention could help light up bikes at night,” 5 2017.

[35] A. Jane, “Material Matters: 3m Scotchlite,” 9 2016.

[36] Biswapathik, “Reflection of light from cat’s eye,” 2014.

[37] K. Daly, “Angle of Reflection,” 2007.

[38] Chetvorno, “Corner reflector,” 3 2012.

[39] N. Buchanan and V. Fusco, “Pilot Tone Reference-Less Phase Conjugator for Phase-Modulated Retrodirective Antenna Applications,” *IEEE Antennas and Wireless Propagation Letters*, vol. 15, pp. 298–300, 2016.

[40] E. D. Sharp and M. A. Diab, “Van Atta Reflector Array,” *IRE Transactions on Antennas and Propagation*, vol. 8, no. 4, pp. 436–438, 1960.

[41] R. C. Hansen, "Communications Satellites Using Arrays," *Proceedings of the IRE*, vol. 49, no. 6, pp. 1066–1074, 1961.

[42] D. Davies, "Some properties of Van Atta arrays and the use of 2-way amplification in the delay paths," *Proceedings of the Institution of Electrical Engineers*, vol. 110, no. 3, p. 507, 1963.

[43] P. Chan and V. Fusco, "Bi-static 5.8GHz RFID range enhancement using retrodirective techniques," in *2011 41st European Microwave Conference*, pp. 976–979, 10 2011.

[44] H. I. El-Sawaf, A. M. El-Tager, and A. M. Ghuneim, "A proposed 2-D active Van Atta retrodirective array using dual-polarized microstrip antenna," in *Asia-Pacific Microwave Conference Proceedings, APMC*, pp. 1103–1105, 2012.

[45] K. S. B. Yau, "Development of a passive retrodirective Van Atta array reflector at X-band," in *2013 International Conference on Radar - Beyond Orthodoxy: New Paradigms in Radar, RADAR 2013*, pp. 398–402, 2013.

[46] A. V. Gevorkyan, "The Van Atta Array Based on Low-Profile Antennas with E-and U-Strip and with Reactive Element between Antennas," in *Conference Proceedings - 2019 Radiation and Scattering of Electromagnetic Waves, RSEMW 2019*, pp. 184–187, Institute of Electrical and Electronics Engineers Inc., 6 2019.

[47] Y. V. Yukhanov, I. V. Merglodov, E. V. Kriuk, and I. V. Ilyin, "Experimental Studies of UWB Multimode Waveguide Van Atta Array," in *Conference Proceedings - 2019 Radiation and Scattering of Electromagnetic Waves, RSEMW 2019*, pp. 244–247, Institute of Electrical and Electronics Engineers Inc., 6 2019.

[48] V. Fusco and N. Buchanan, "Developments in retrodirective array technology," *IET Microwaves, Antennas and Propagation*, vol. 7, pp. 131–140, 1 2013.

[49] T. Brabetz, V. F. Fusco, and S. Karode, "Balanced subharmonic mixers for retrodirective-array applications," *IEEE Transactions on Microwave Theory and Techniques*, vol. 49, pp. 465–469, 3 2001.

[50] B. Y. Toh, V. F. Fusco, and N. B. Buchanan, "Assessment of performance limitations of PON retrodirective arrays," *IEEE Transactions on Antennas and Propagation*, vol. 50, pp. 1425–1432, 10 2002.

[51] V. Fusco and N. B. Buchanan, "High-performance IQ modulator-based phase conjugator for modular retrodirective antenna array implementation," *IEEE Transactions on Microwave Theory and Techniques*, vol. 57, pp. 2301–2306, 10 2009.

[52] N. B. Buchanan, V. F. Fusco, M. V. D. Vorst, N. Williams, C. Winter, N. Ireland, S. Park, E. S. Agency, C. T. Services, C. Road, and U. Kingdom, "New Retrodirective Antenna Techniques for Mobile Terminal Applications," in *Proc. 32nd Antenna Workshop, ESA/ESTEX*, pp. 1–5, 1 2010.

[53] C.-K. K. Yang and B. Razavi, "Delay-locked loops-an overview," in *Phase-Locking in High Performance Systems: From Devices to Architectures*, pp. 13–22, New York, NY, USA: Wiley, 2003.

[54] A. Paraboni and C. Riva, "A new method for the prediction of fade duration statistics in satellite links above 10 GHz," *International Journal of Satellite Communications*, vol. 12, pp. 387–394, 7 1994.

[55] V. F. Fusco and S. L. Karode, "Self-phasing antenna array techniques for mobile communications applications," *Electronics and Communication Engineering Journal*, vol. 11, pp. 279–286, 12 1999.

[56] M. Y. Huang, T. Chi, F. Wang, and H. Wang, "An All-Passive Negative Feedback Network for Broadband and Wide Field-of-View Self-Steering Beam-Forming with Zero DC Power Consumption," *IEEE Journal of Solid-State Circuits*, vol. 52, pp. 1260–1273, 5 2017.

[57] P. McNamee, "Automated Memoization in C++," 1998.

[58] R. Jaeger and T. Blalock, *Microelectronic Circuit Design*. New York, NY, USA: McGraw-Hill Education, 4 ed., 2010.

[59] T. Instruments, “Build Your Own LaunchPad/BoosterPack,” 2013.

[60] K. P. Thakore, Kehul Shah, and N. M. Devashrey, “Design and implementation of low power phase frequency detector for phase lock loop,” *Proceedings of the 3rd International Conference on Computing Methodologies and Communication, ICCMC 2019*, no. Iccmc, pp. 644–647, 2019.

[61] T. Johnson, A. Fard, and D. Åberg, “An improved low voltage phase-frequency detector with extended frequency capability,” in *Midwest Symposium on Circuits and Systems*, vol. 1, 2004.

[62] “Tiva TM4C1294NCPDT Microcontroller,” 2014.

[63] S. Das, S. Bhattacherjee, D. Mandal, and A. K. Bhattacharjee, “Optimal sidelobe reduction of symmetric linear antenna array using Genetic Algorithm,” in *Proceedings of the 2010 Annual IEEE India Conference: Green Energy, Computing and Communication, INDICON 2010*, 2010.

[64] C. C. Chen and S. I. Liu, “An infinite phase shift delay-locked loop with voltage-controlled sawtooth delay line,” in *2007 IEEE Asian Solid-State Circuits Conference, A-SSCC*, pp. 448–451, 2007.

[65] J. Jasielski, S. Kuta, W. Machowski, and W. Kołodziejski, “An analog dual delay locked loop using coarse and fine programmable delay elements,” in *Proceedings of the 20th International Conference on Mixed Design of Integrated Circuits and Systems, MIXDES 2013*, pp. 185–190, 2013.

[66] M. Stoopman, K. Philips, and W. A. Serdijn, “An RF-Powered DLL-Based 2.4-GHz Transmitter for Autonomous Wireless Sensor Nodes,” *IEEE Transactions on Microwave Theory and Techniques*, vol. 65, pp. 2399–2408, 7 2017.

[67] H. Babazadeh, A. Esmaili, and K. Hadidi, “A high-speed and wide detectable frequency range phase detector for DLLs,” in *2009 NORCHIP*, 2009.

[68] Y. L. Lo, P. Y. Chou, H. H. Cheng, S. F. Tsai, and W. B. Yang, "An all-digital DLL with dual-loop control for multiphase clock generator," in *2011 International Symposium on Integrated Circuits, ISIC 2011*, pp. 388–391, 2011.

[69] G. Luo and X. Zeng, "An improved voltage-controlled delay line for delay locked loops," in *ICCRD2011 - 2011 3rd International Conference on Computer Research and Development*, vol. 2, pp. 237–240, 2011.

[70] A. Fathi, M. Mousazadeh, and A. Khoei, "High-speed, low power, and dead zone improved phase frequency detector," *IET Circuits, Devices and Systems*, vol. 13, pp. 1056–1062, 10 2019.

Appendices

Appendix A

Final Prototype Design Documents

A.1 Received Signal Beamformer Design Files

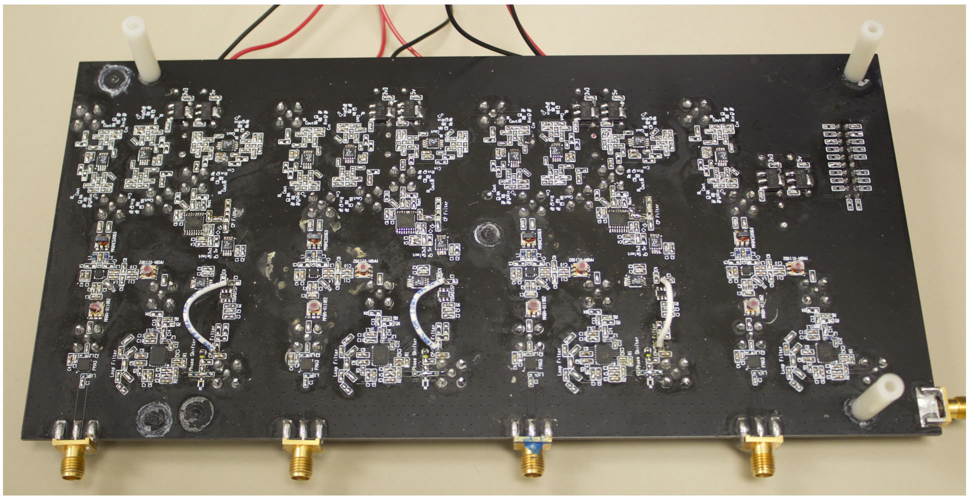


Figure A.1: Assembled DLL-HPC received signal beamformer PCB - component side

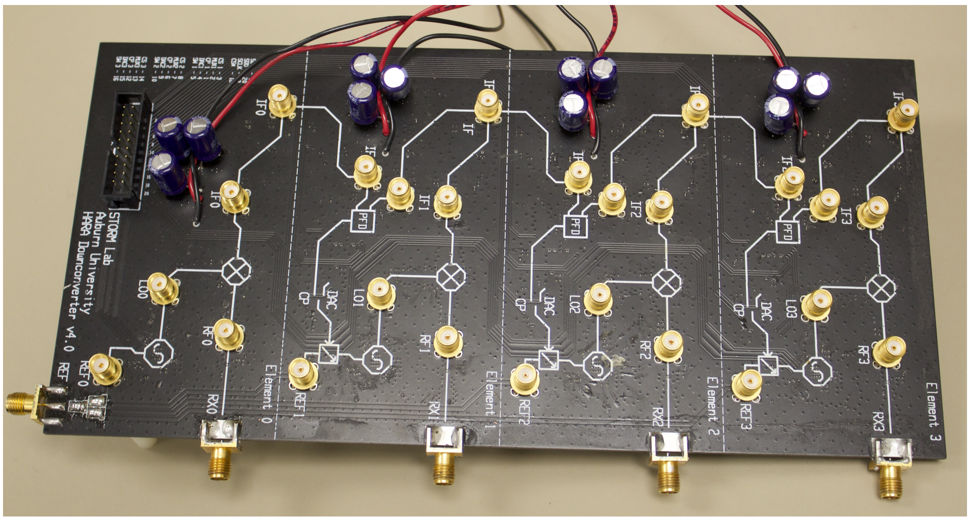


Figure A.2: Assembled DLL-HPC received signal beamformer PCB - labeled side

A.1.1 PCB Renders

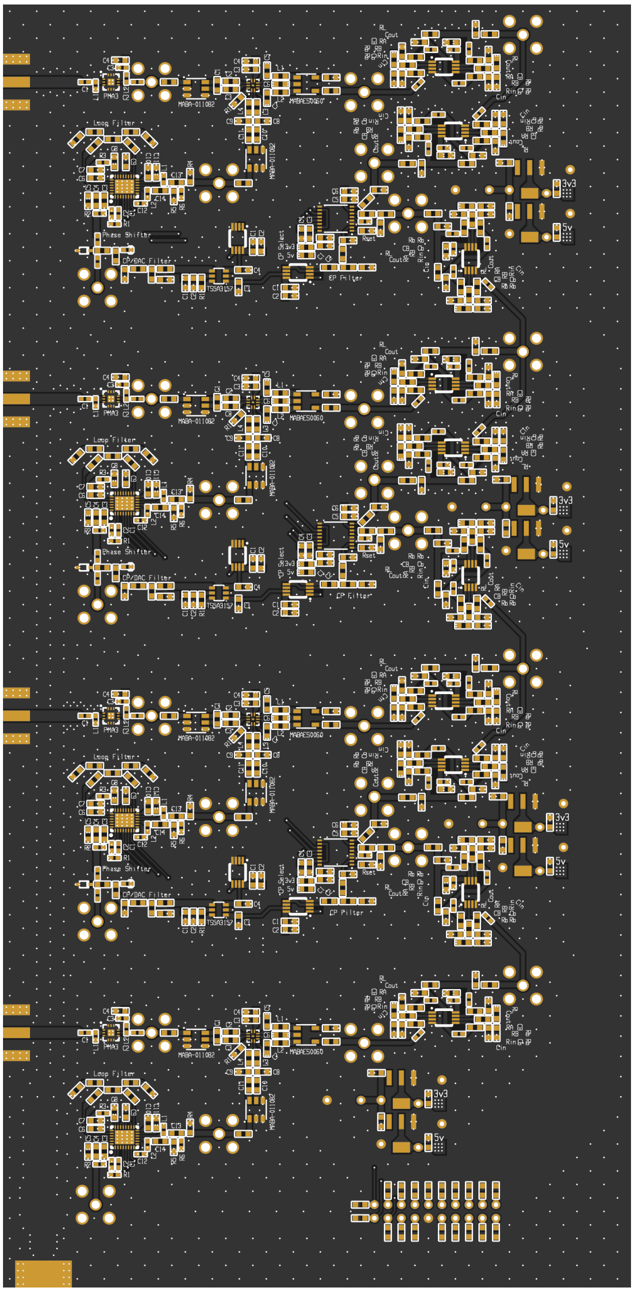


Figure A.3: DLL-HPC received signal beamformer PCB front-side render

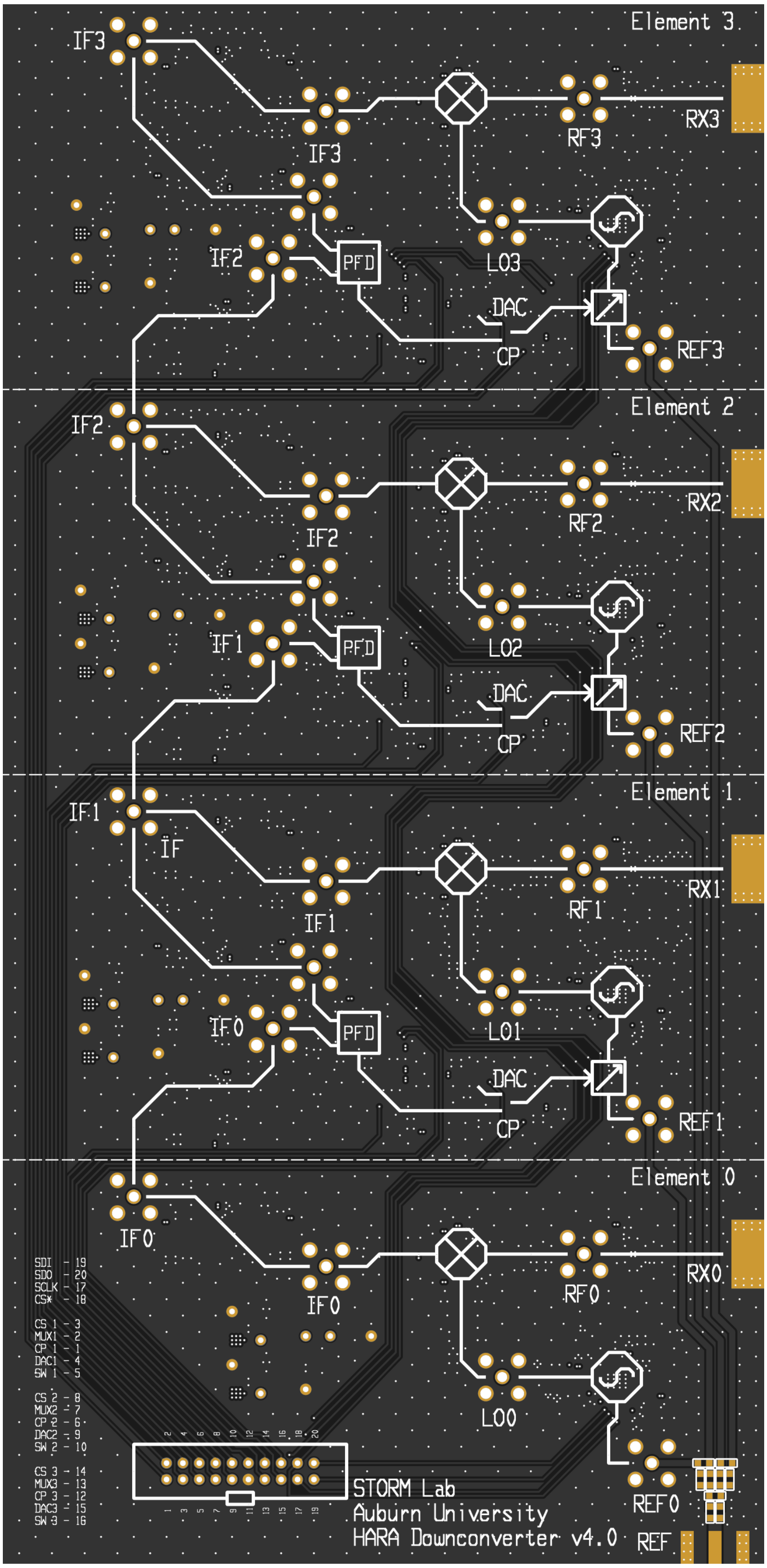


Figure A.4: DLL-HPC received signal beamformer PCB back-size render

A.1.2 PCB Schematics: Element 0

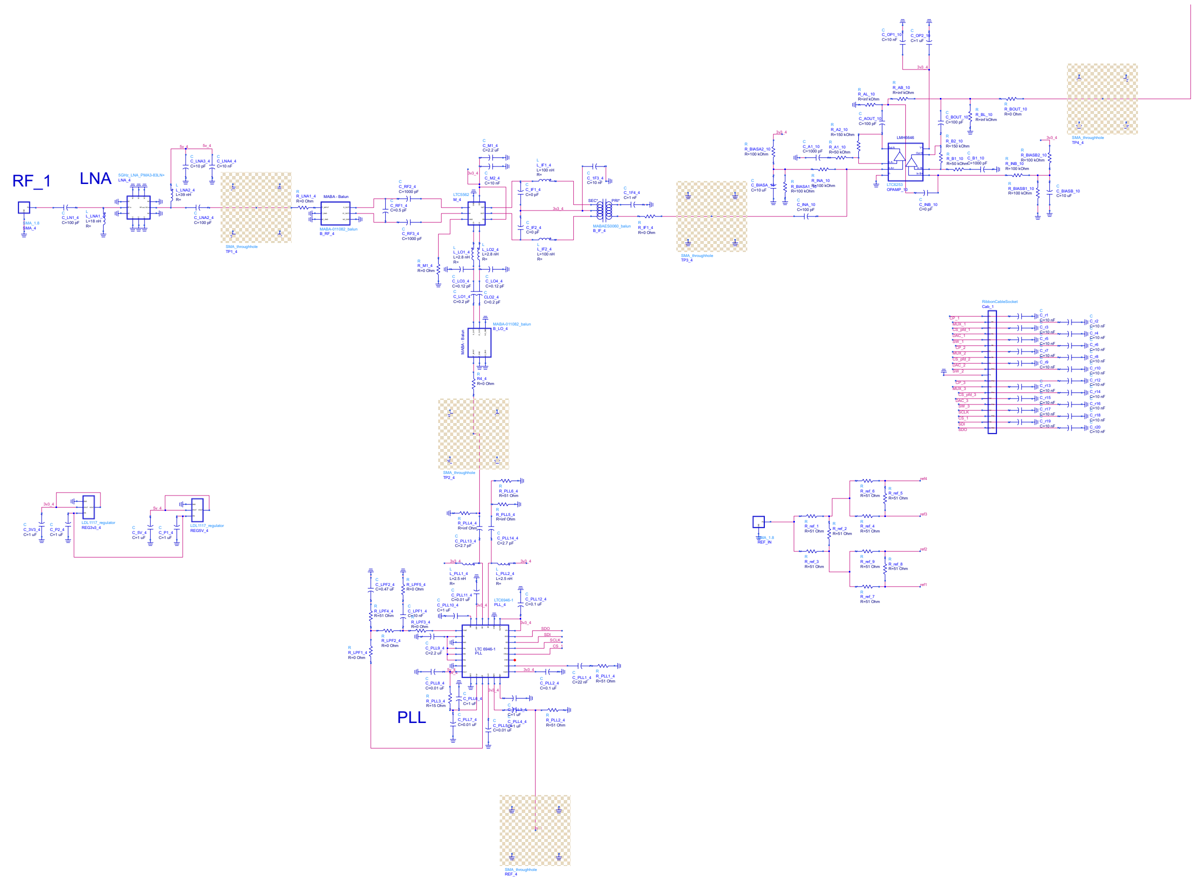


Figure A.5: DLL-HPC received signal beamformer full schematic for element 0

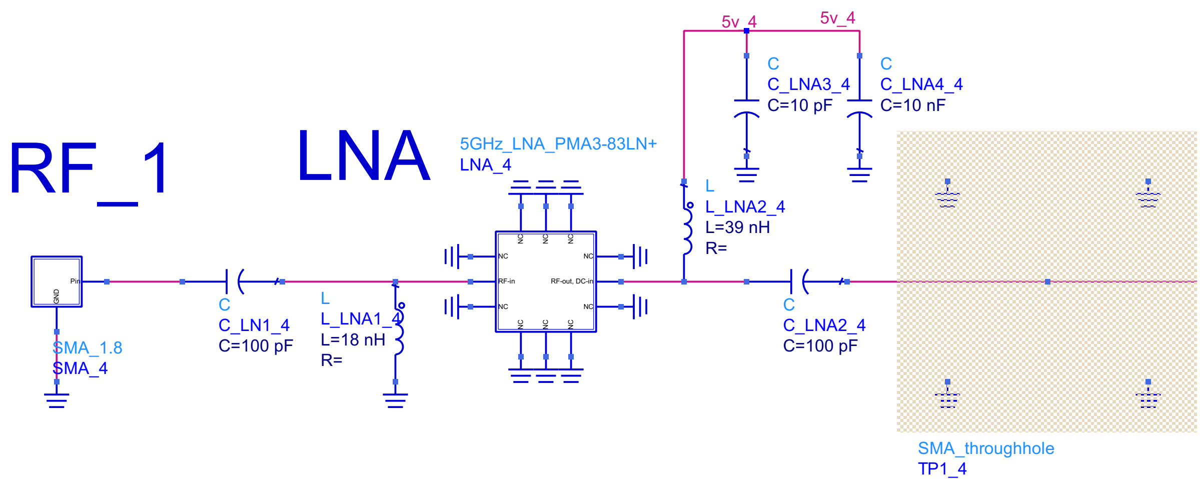


Figure A.6: DLL-HPC received signal beamformer partial schematic for element 0 - RF input and low-noise amplifier

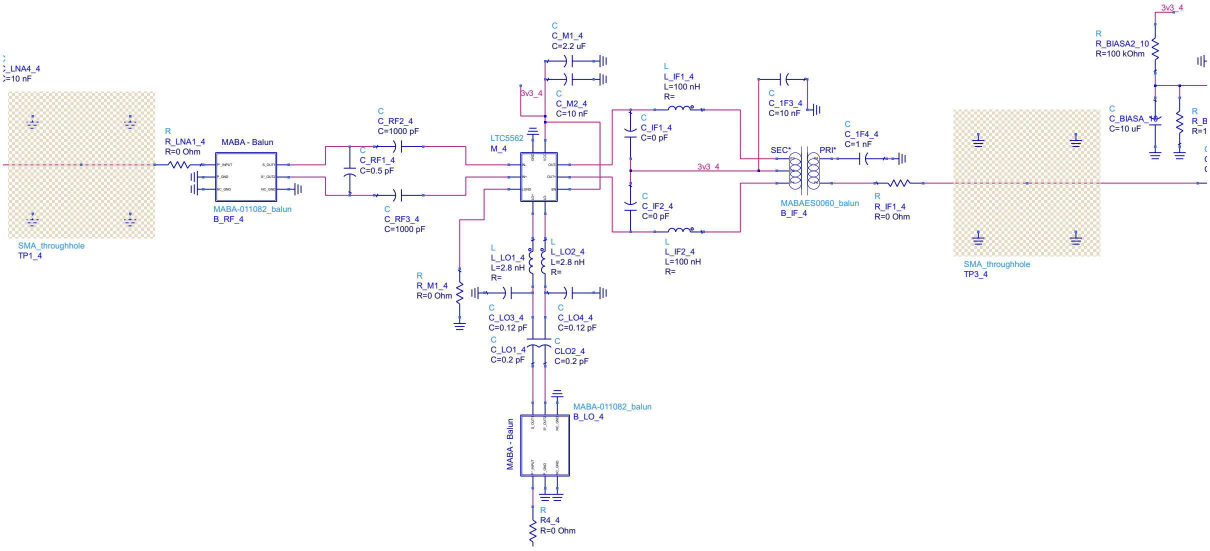


Figure A.7: DLL-HPC received signal beamformer partial schematic for element 0 - downconverting mixer

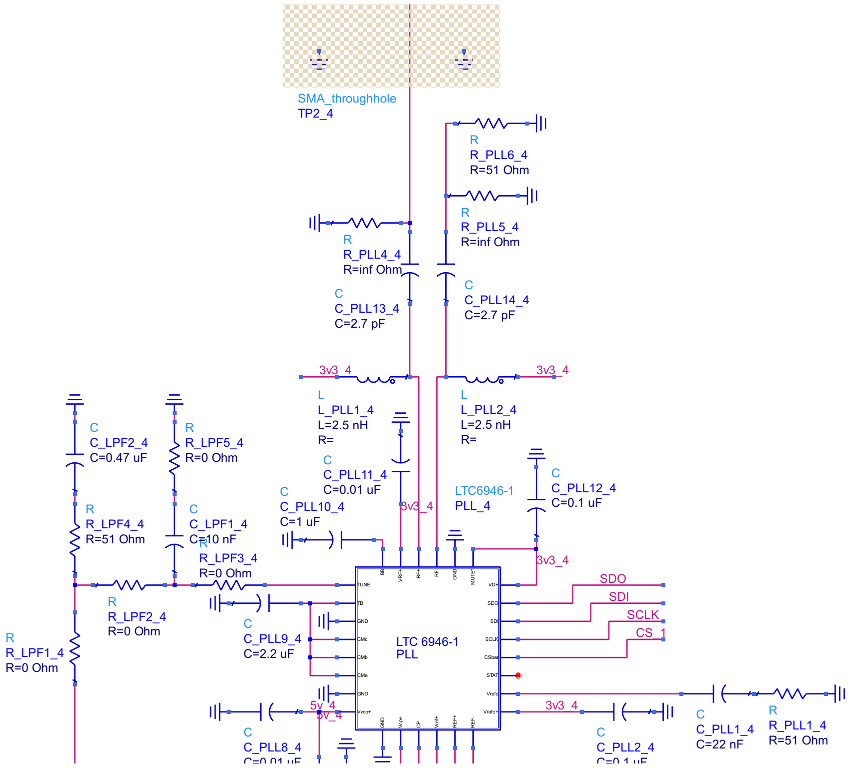


Figure A.8: DLL-HPC received signal beamformer partial schematic for element 0 - PLL LO part 1

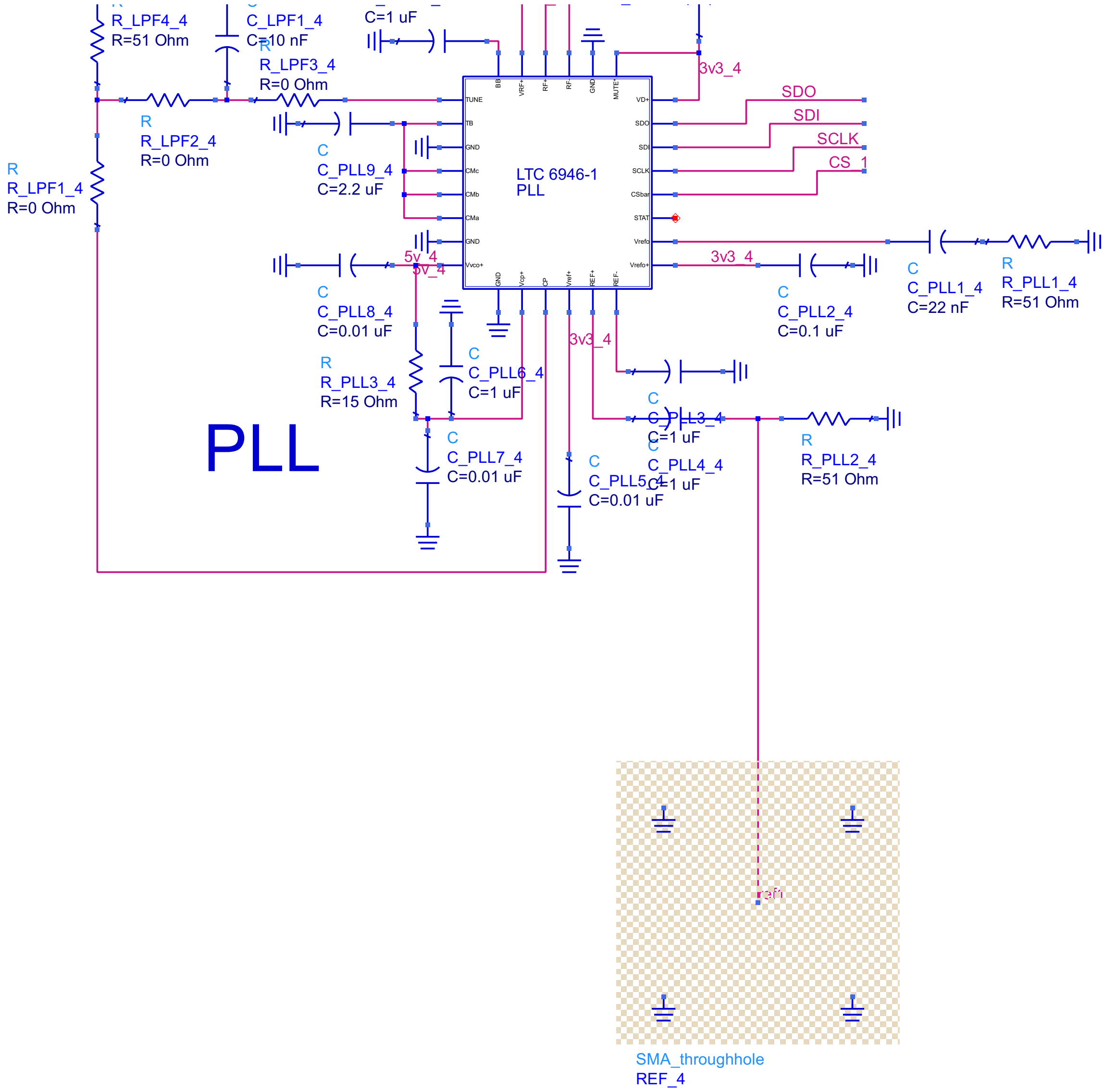


Figure A.9: DLL-HPC received signal beamformer partial schematic for element 0 - PLL LO part 2

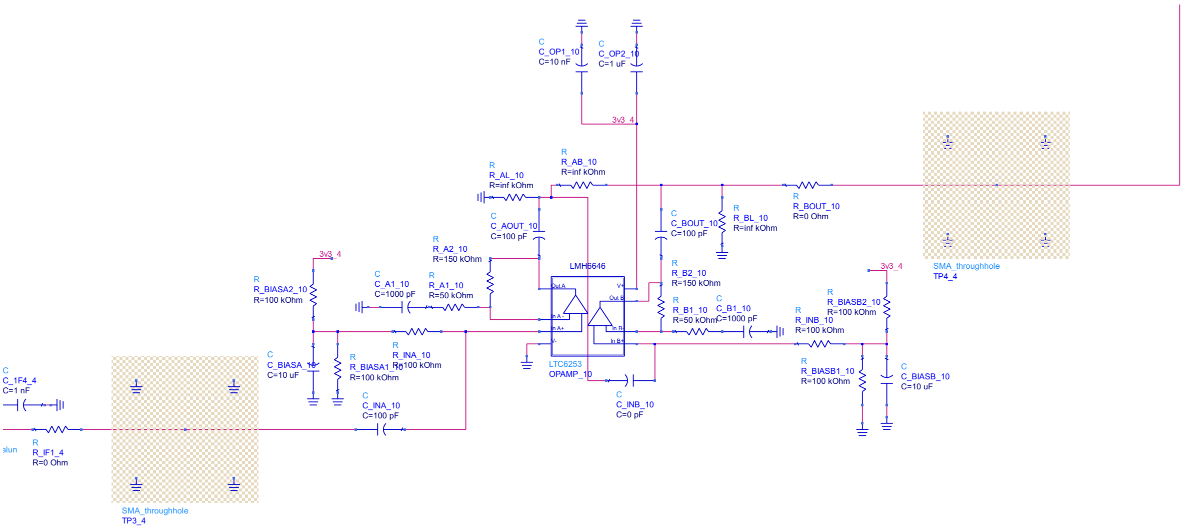


Figure A.10: DLL-HPC received signal beamformer partial schematic for element 0 - IF amplifier and signal to element 1's phase detector

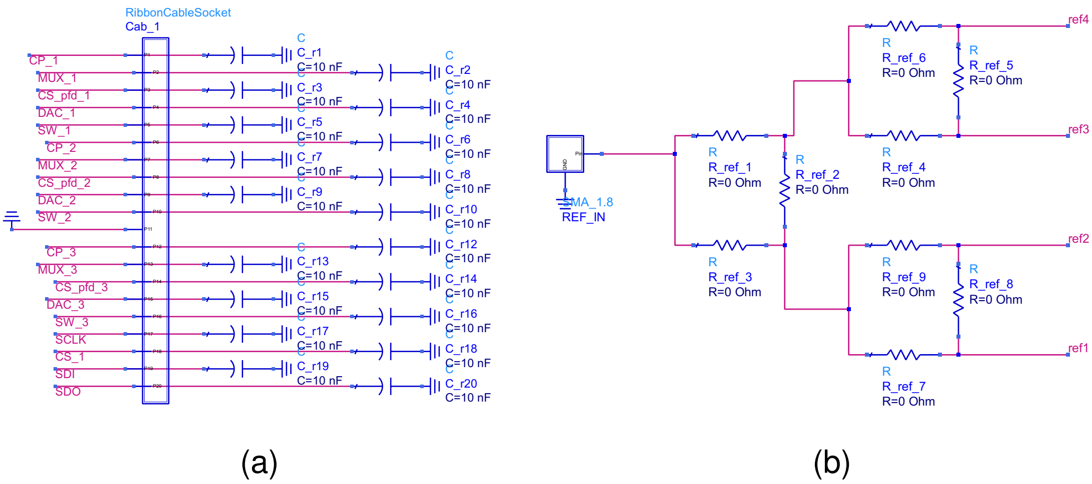


Figure A.11: DLL-HPC received signal beamformer schematic for (a) ribbon cable (b) reference signal splitter

A.1.3 PCB Schematics: Element n

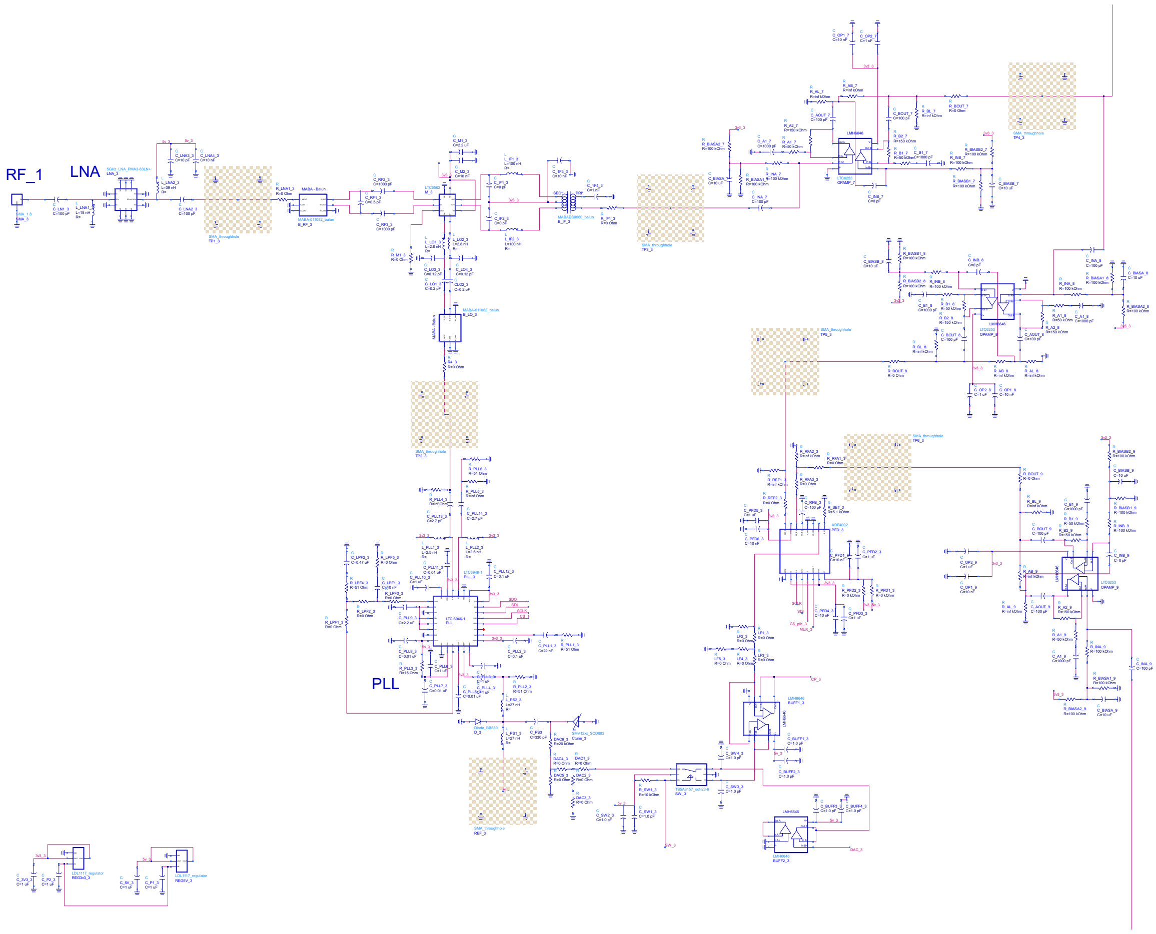


Figure A.12: DLL-HPC received signal beamformer full schematic for element 1; elements 2 and 3 are a copy of element 1

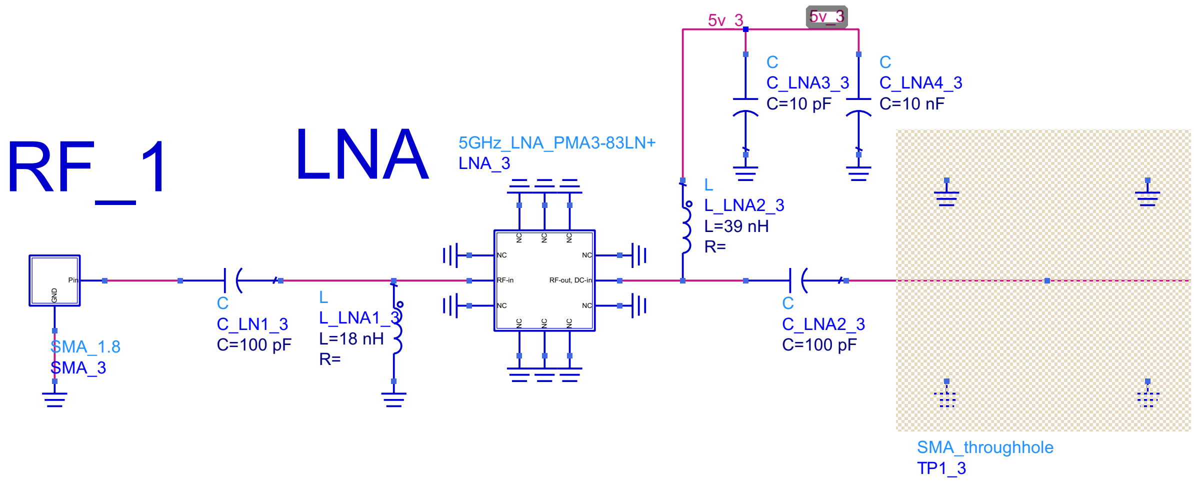


Figure A.13: DLL-HPC received signal beamformer partial schematic for element 1 - RF input and low-noise amplifier

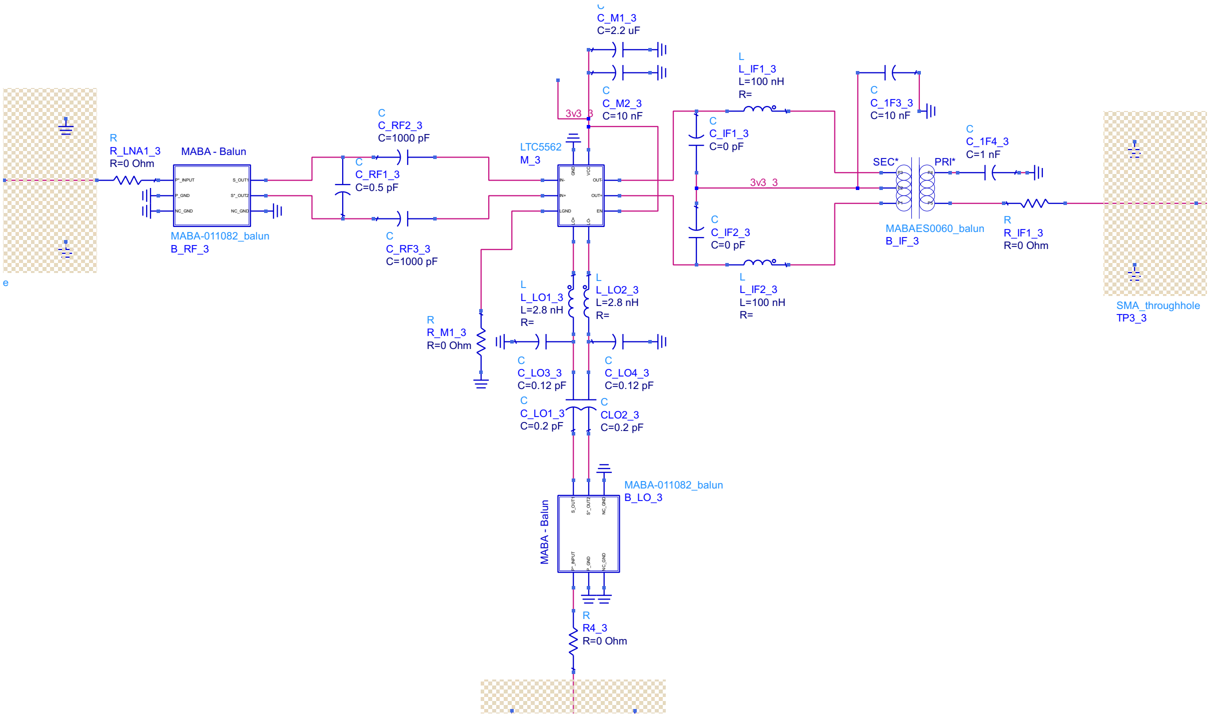


Figure A.14: DLL-HPC received signal beamformer partial schematic for element 1 - down-converting mixer

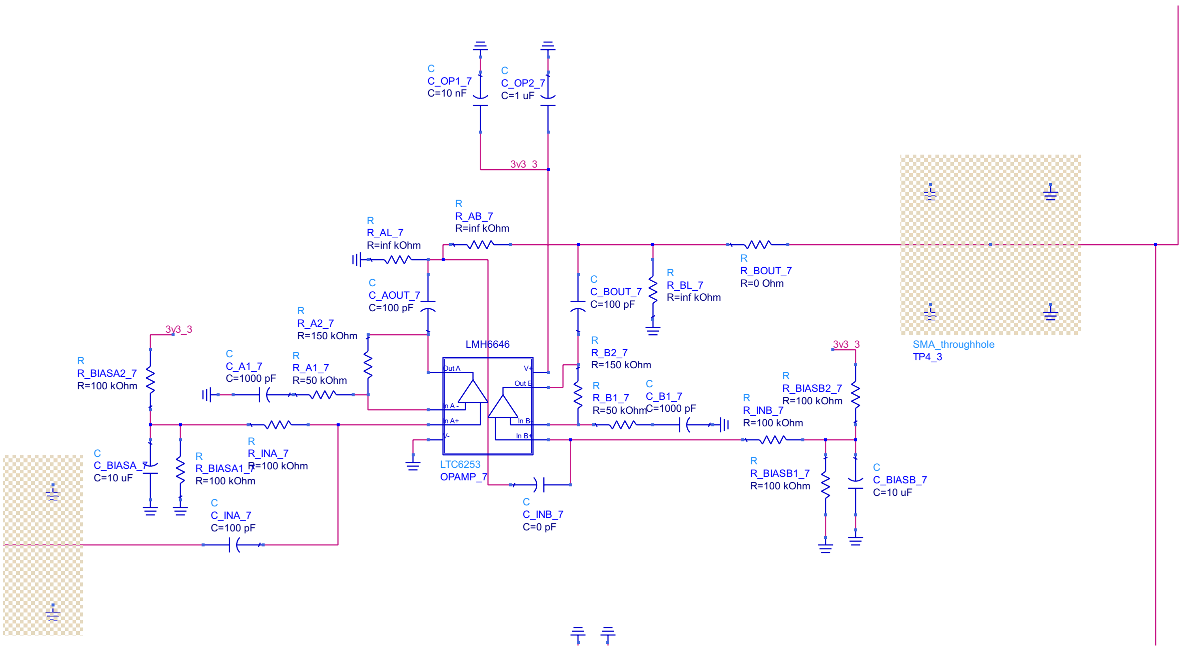


Figure A.15: DLL-HPC received signal beamformer partial schematic for element 1 - IF amplifier and signal leading to element 2's phase detector

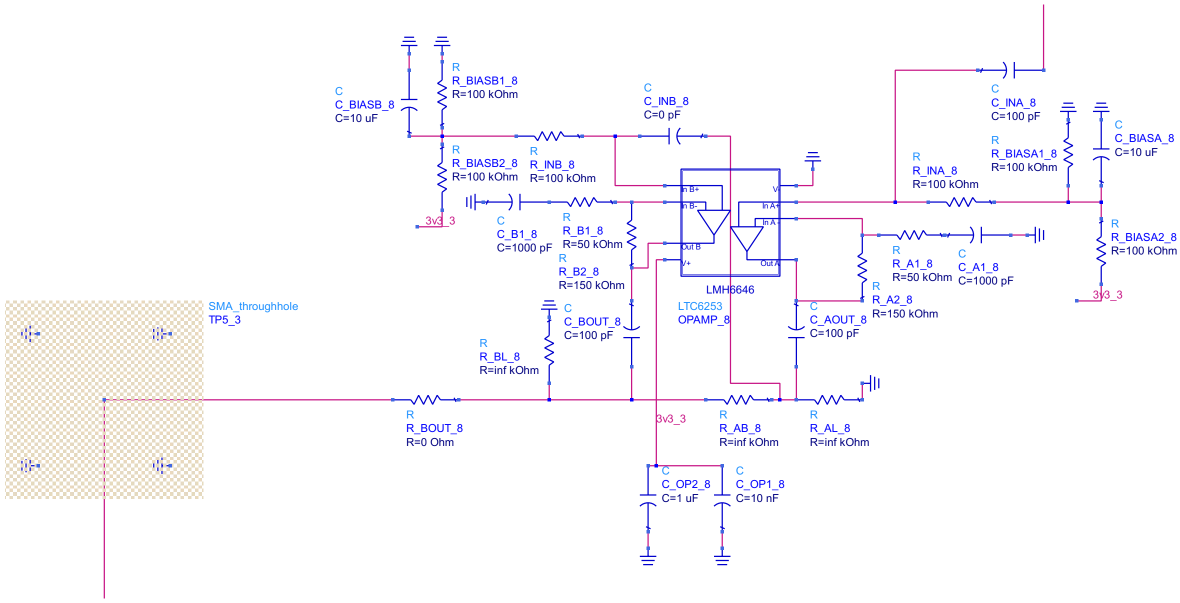


Figure A.16: DLL-HPC received signal beamformer partial schematic for element 1 - secondary IF amplifier before phase detector

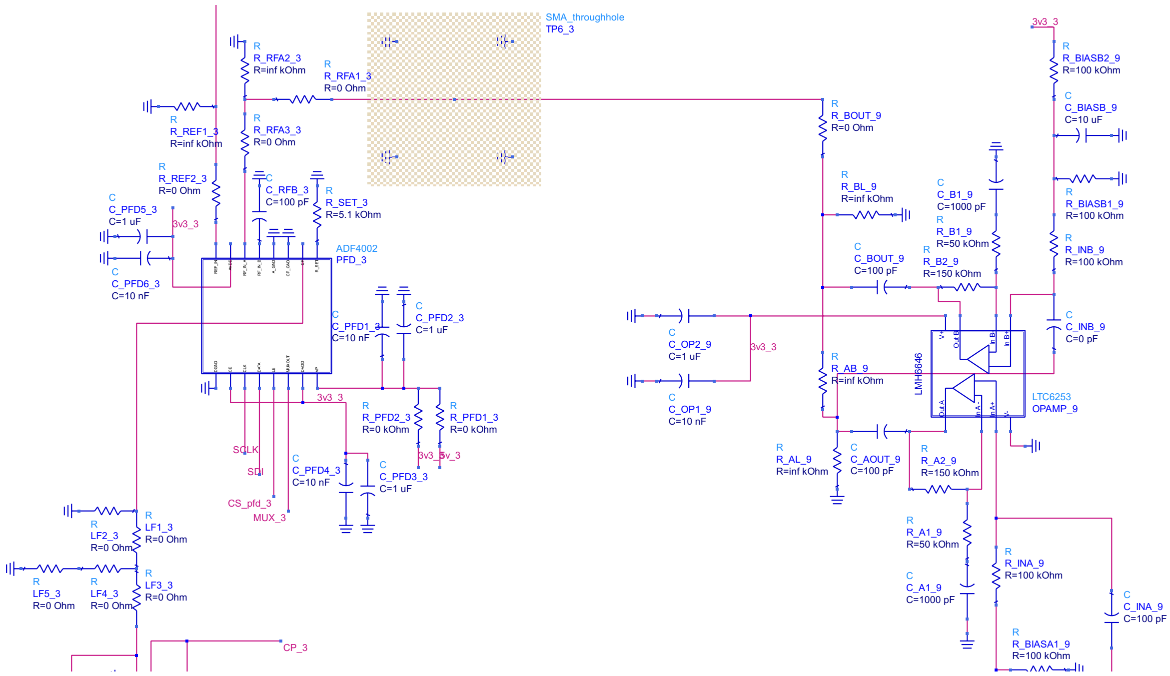


Figure A.17: DLL-HPC received signal beamformer partial schematic for element 1 - PFD/CP and secondary IF amplifier for element 0's signal

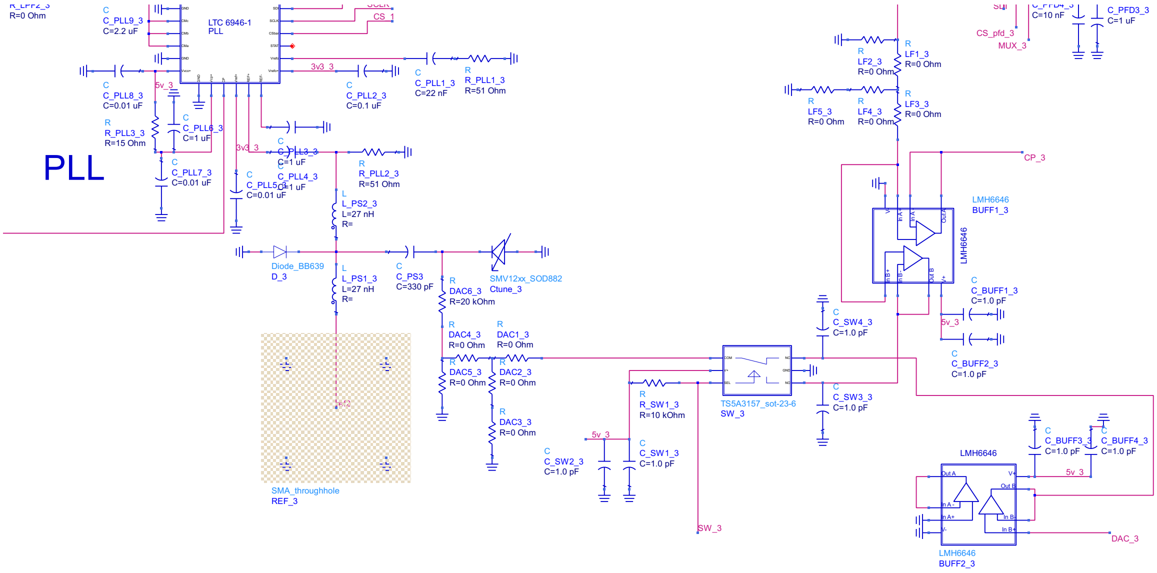


Figure A.18: DLL-HPC received signal beamformer partial schematic for element 1 - charge pump loop filter buffer, switch, and phase shifter circuitry

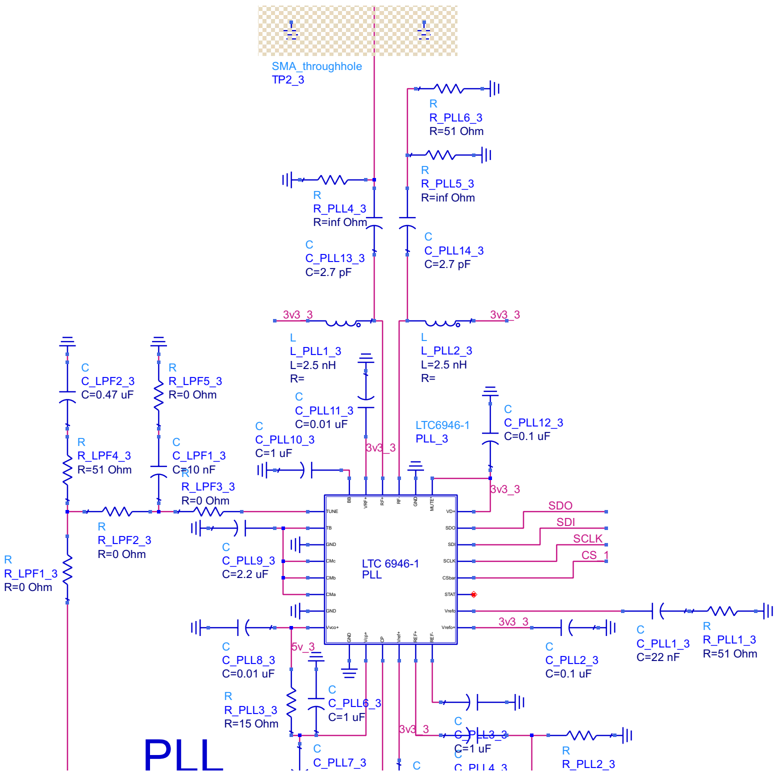


Figure A.19: DLL-HPC received signal beamformer partial schematic for element 1 - PLL LO leading back to downconverting mixer

A.1.4 PCB Layout: Whole Board

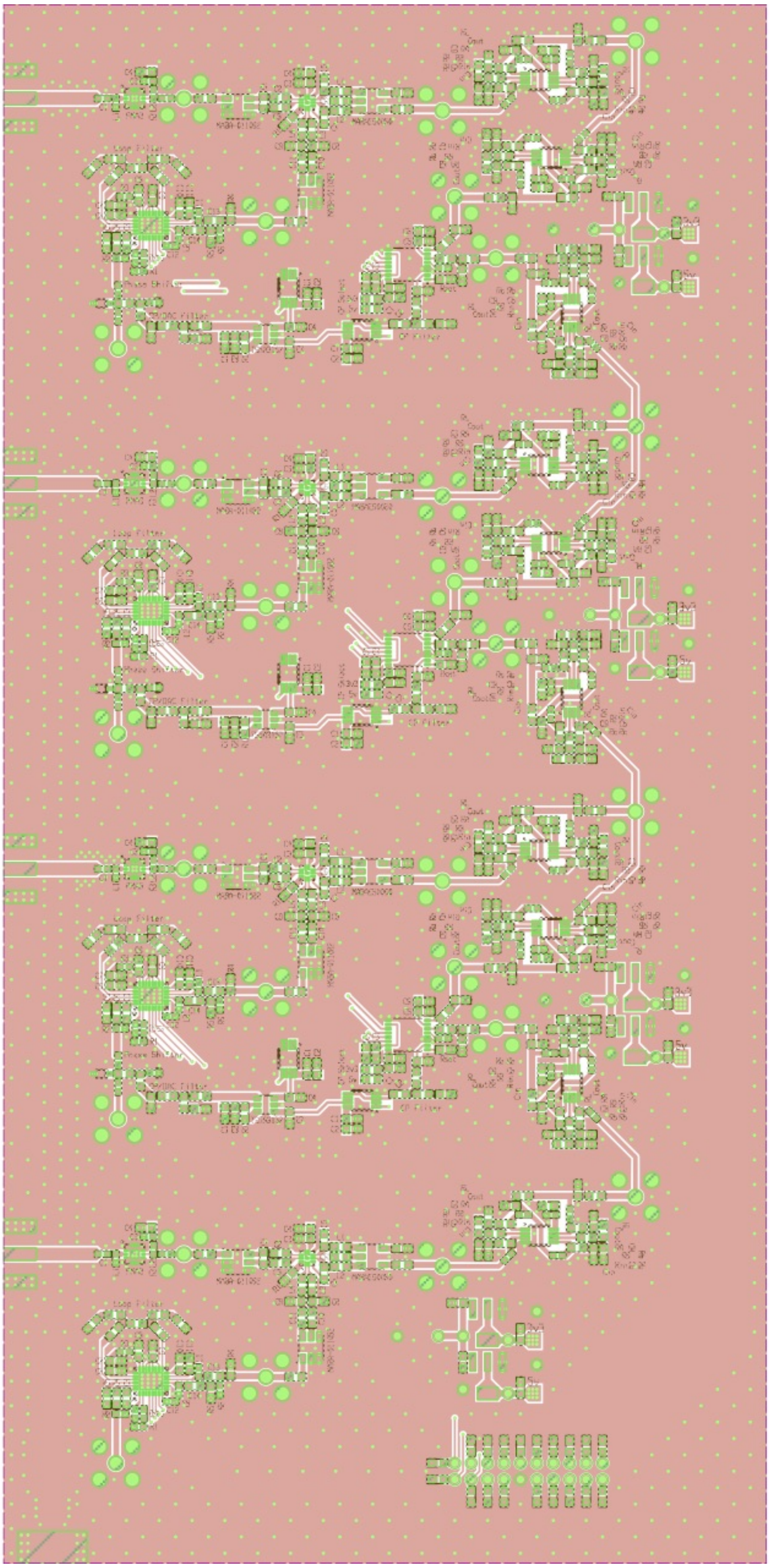


Figure A.20: DLL-HPC received signal beamformer top-side PCB layout

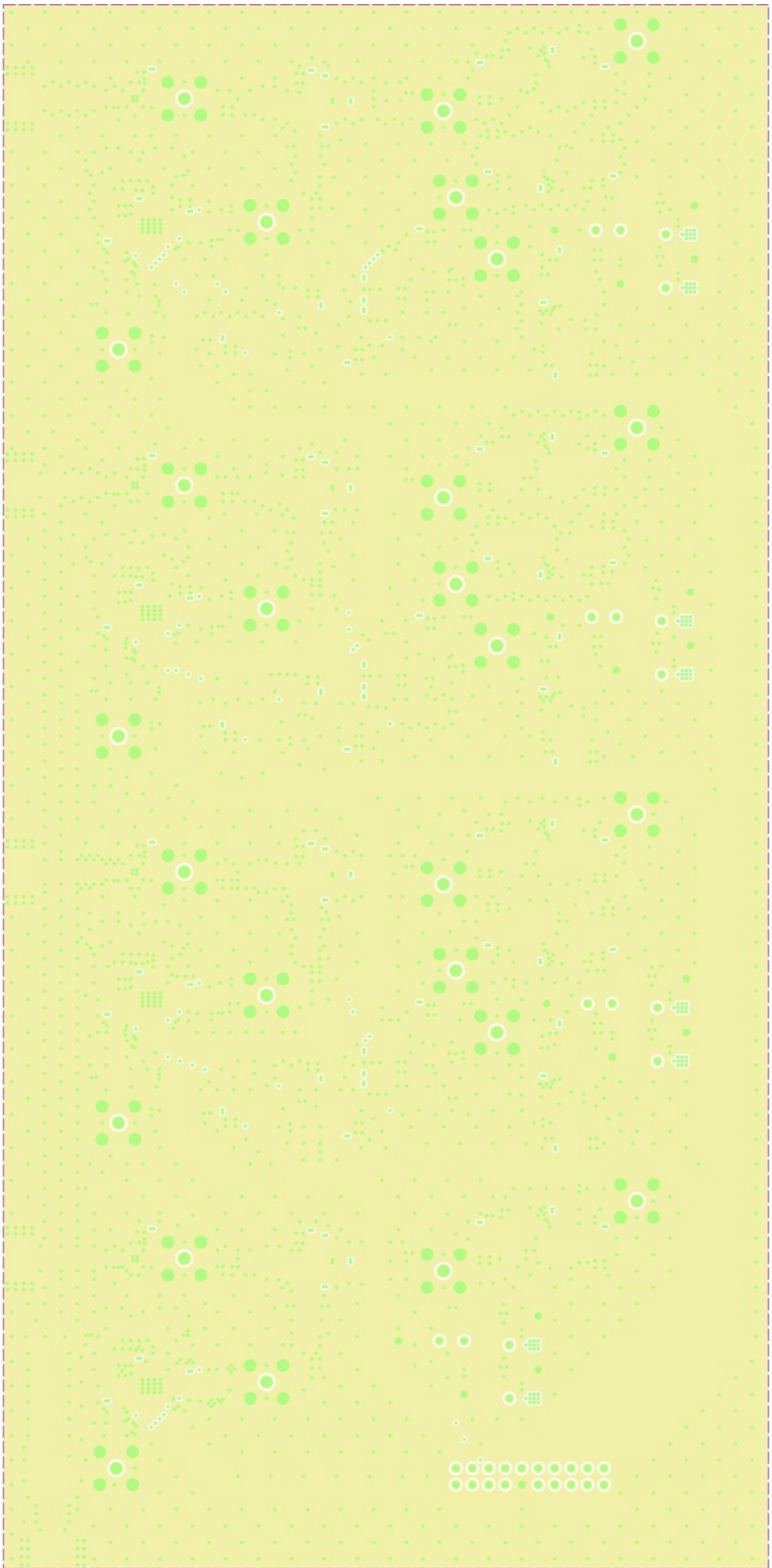


Figure A.21: DLL-HPC received signal beamformer PCB inner layer 1

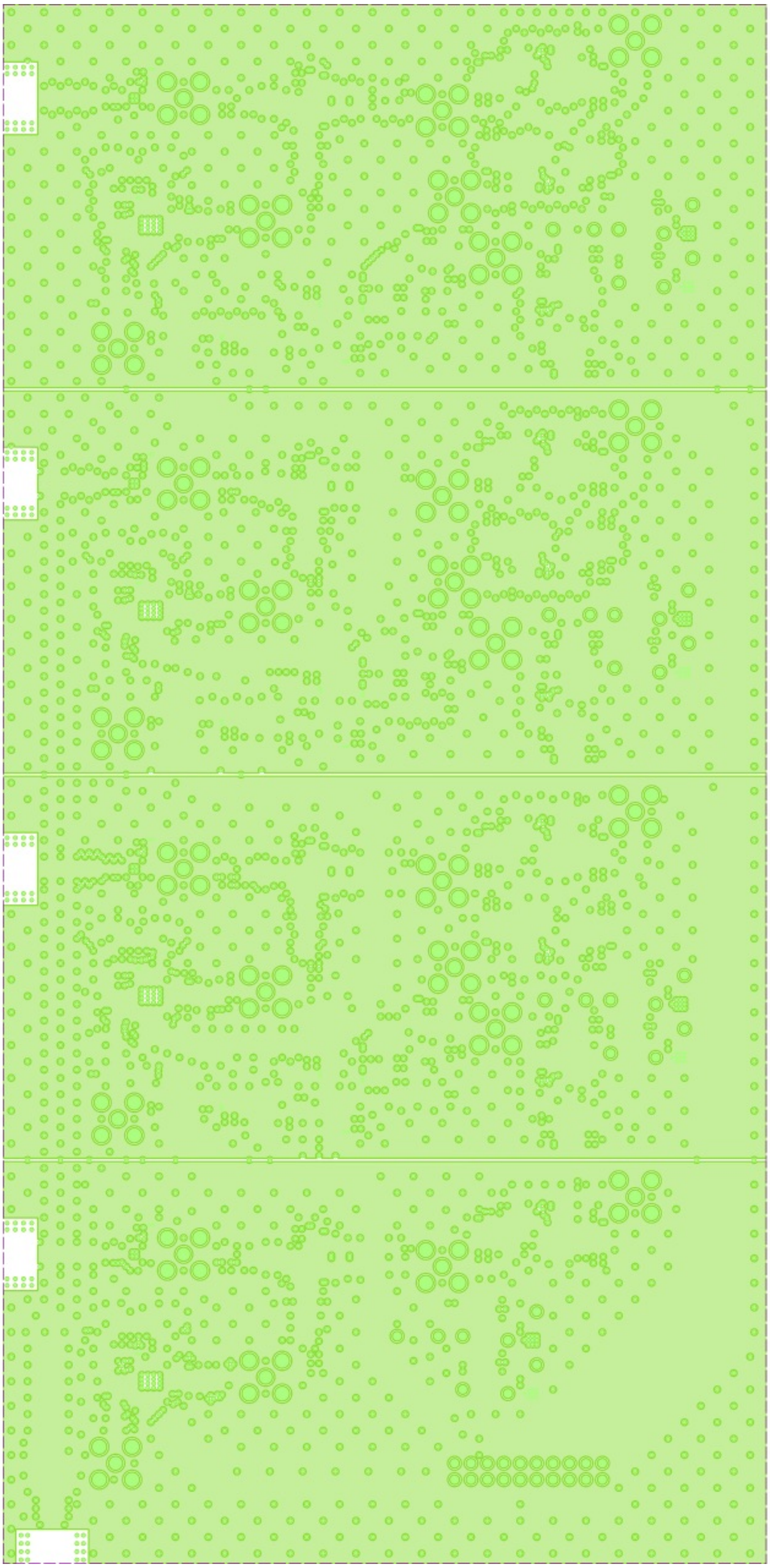


Figure A.22: DLL-HPC received signal beamformer PCB inner layer 2

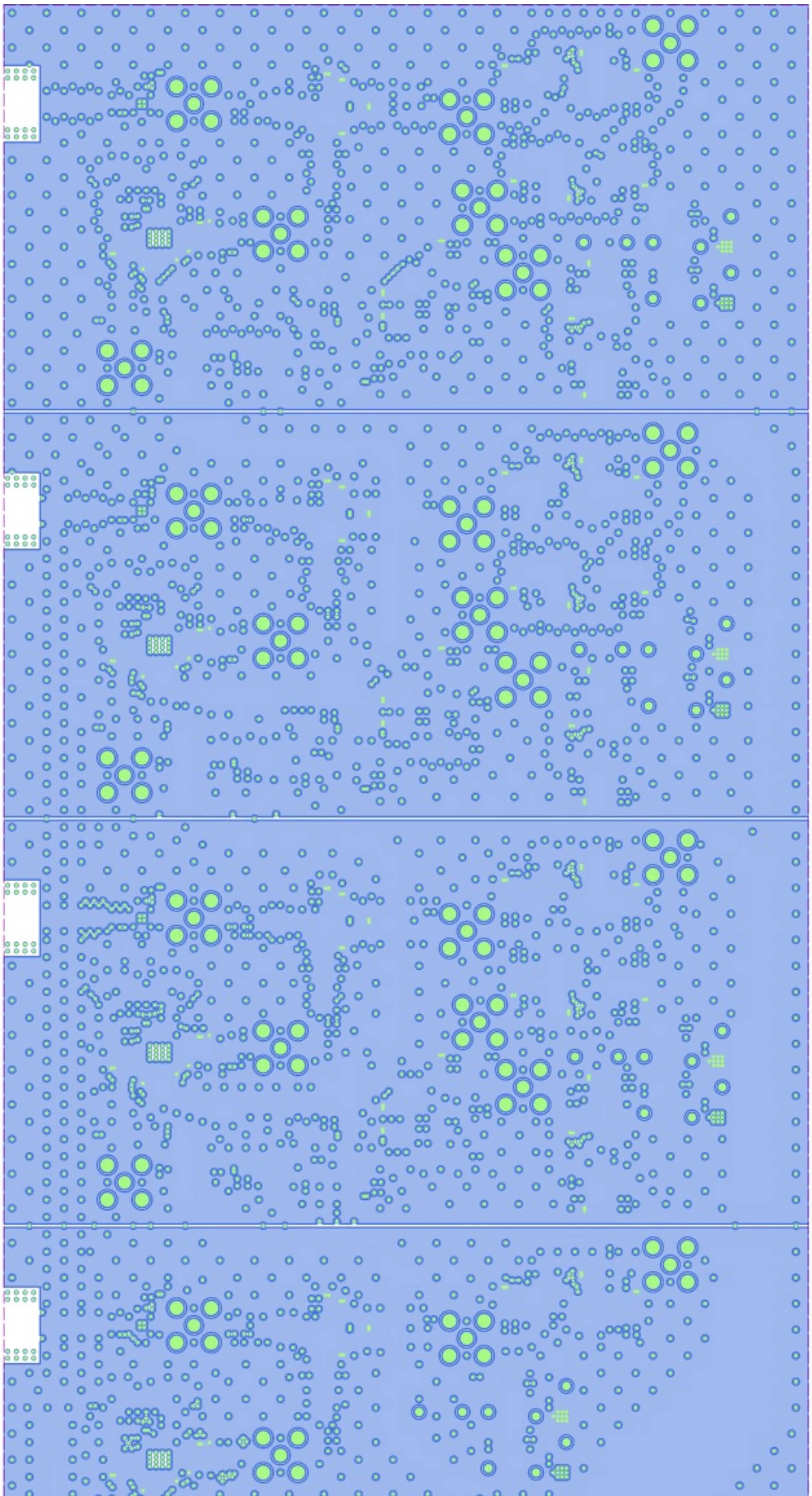


Figure A.23: DLL-HPC received signal beamformer PCB inner layer 3

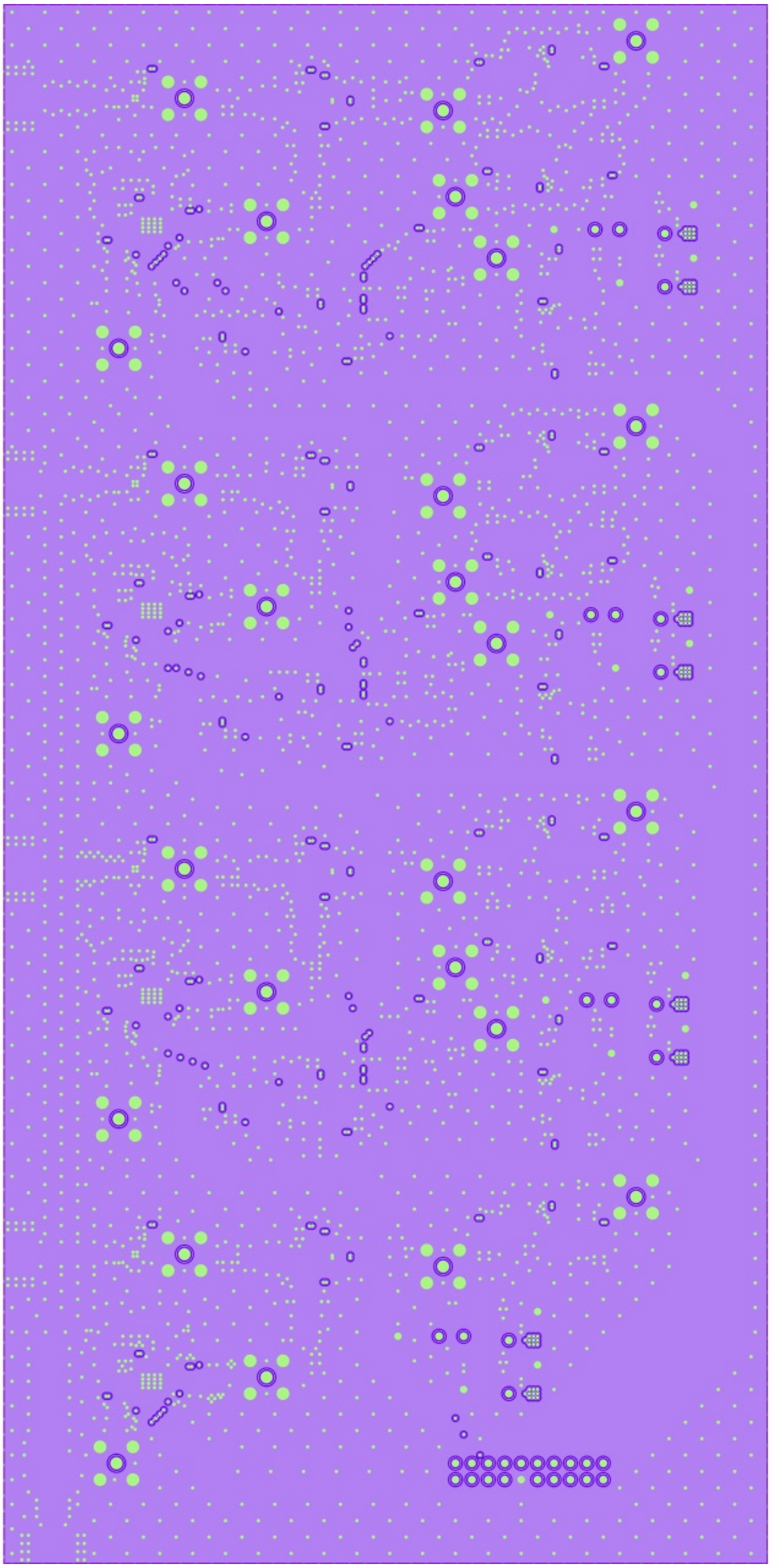


Figure A.24: DLL-HPC received signal beamformer PCB inner layer 4

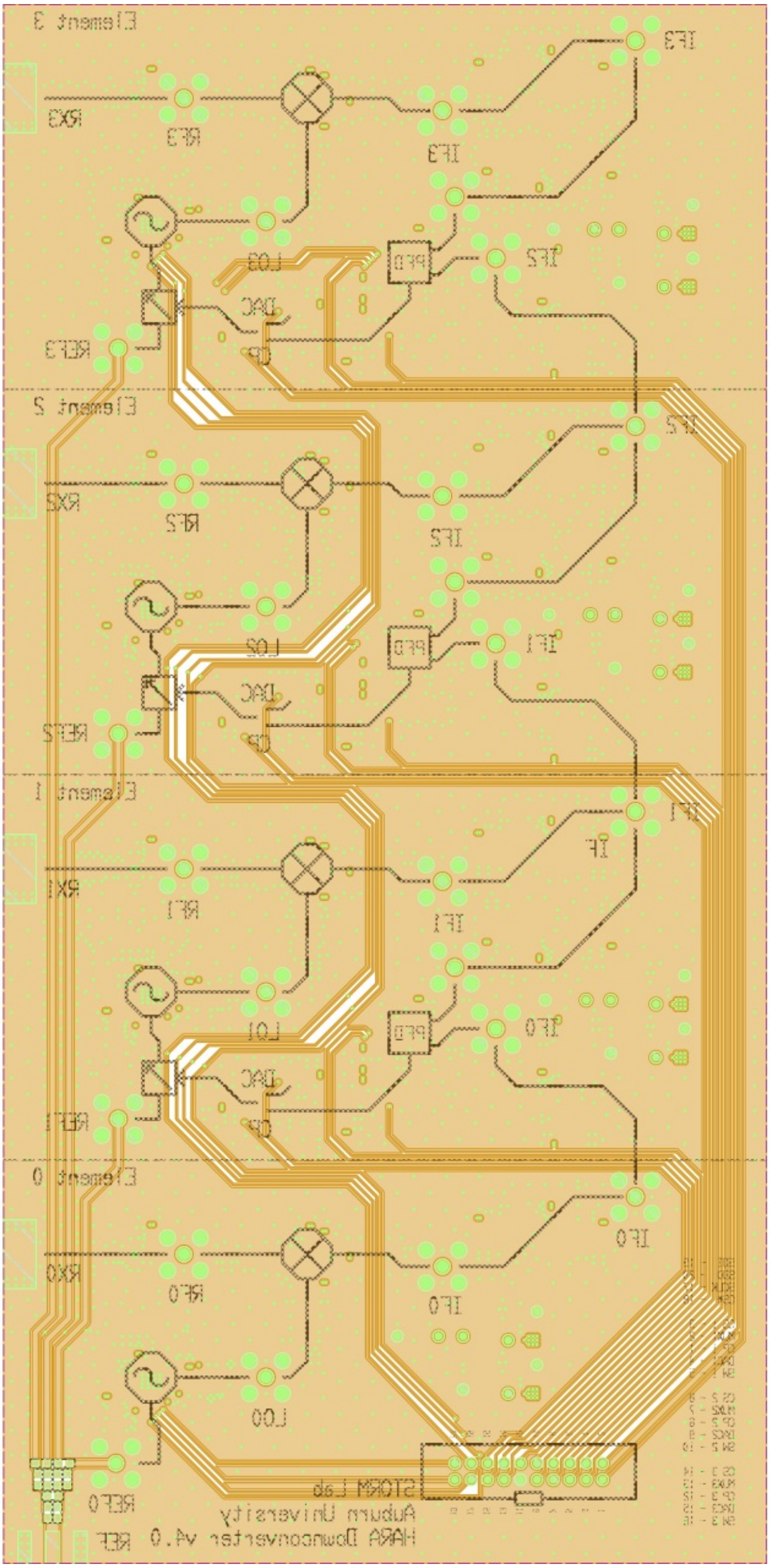


Figure A.25: DLL-HPC received signal beamformer back-side PCB layout

A.1.5 PCB Layout: Element 0

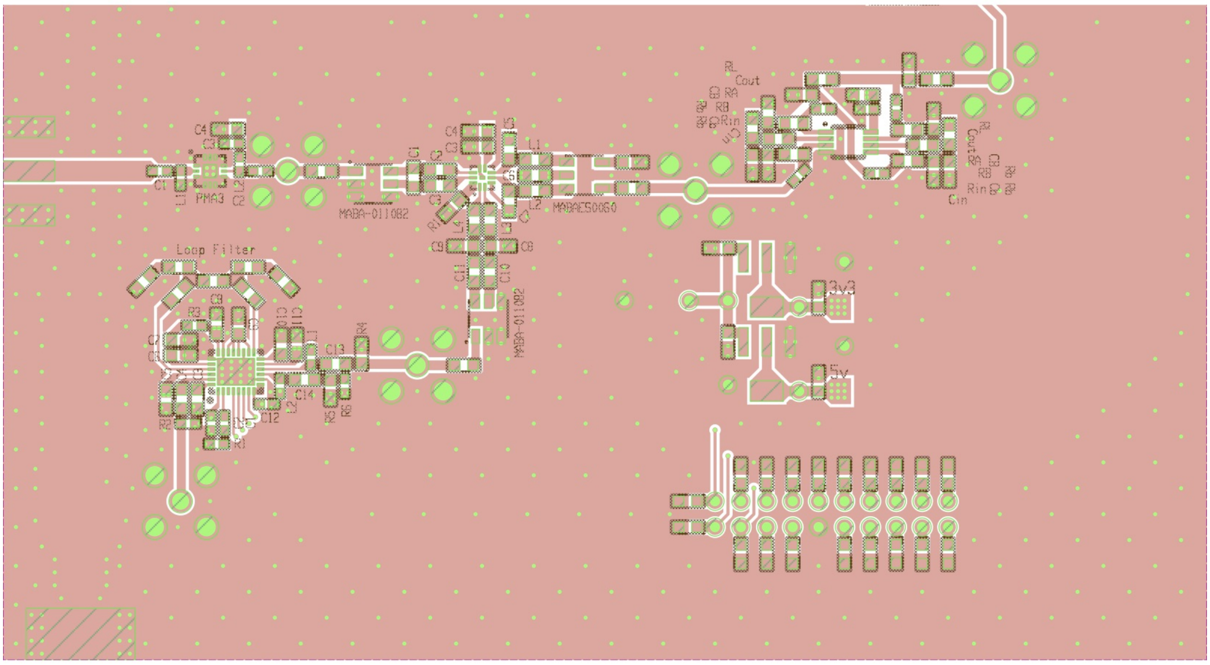


Figure A.26: DLL-HPC received signal beamformer front-side PCB layout for element 0

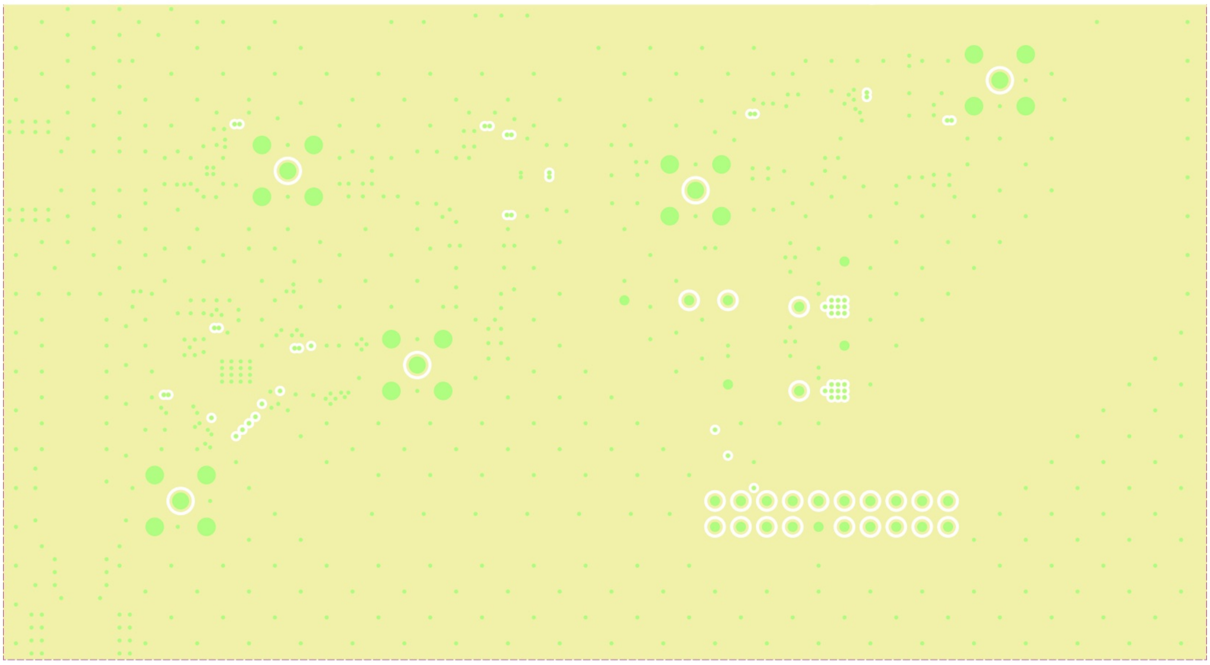


Figure A.27: DLL-HPC received signal beamformer PCB layout for element 0 inner layer 1

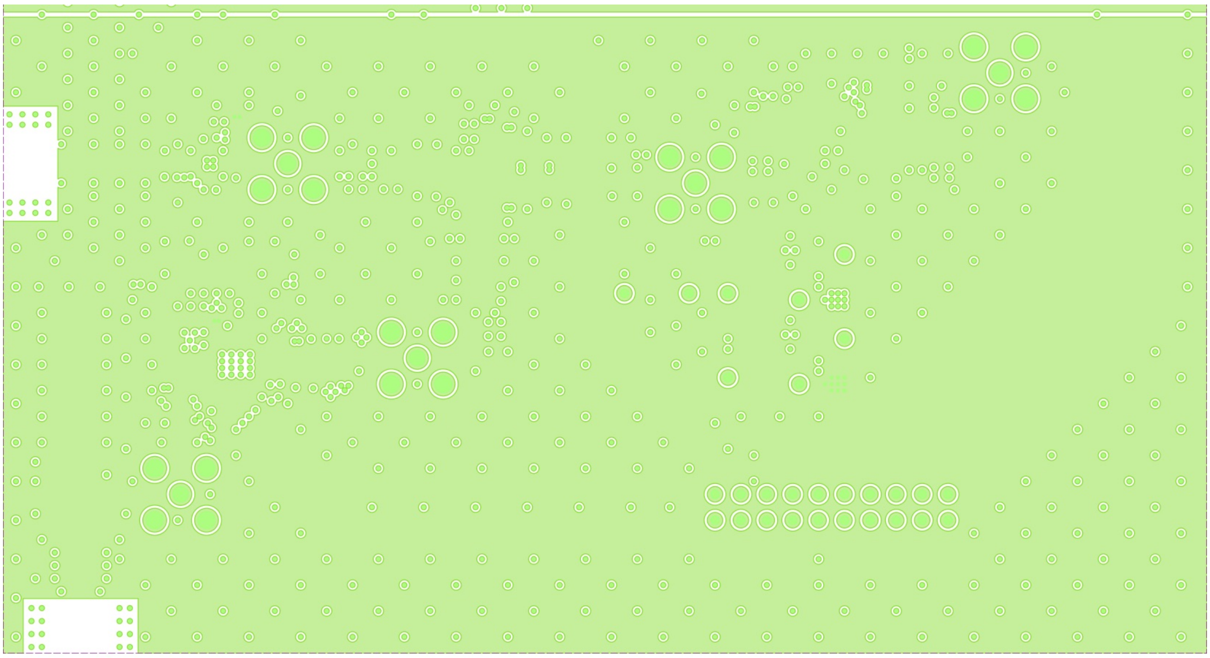


Figure A.28: DLL-HPC received signal beamformer PCB layout for element 0 inner layer 2

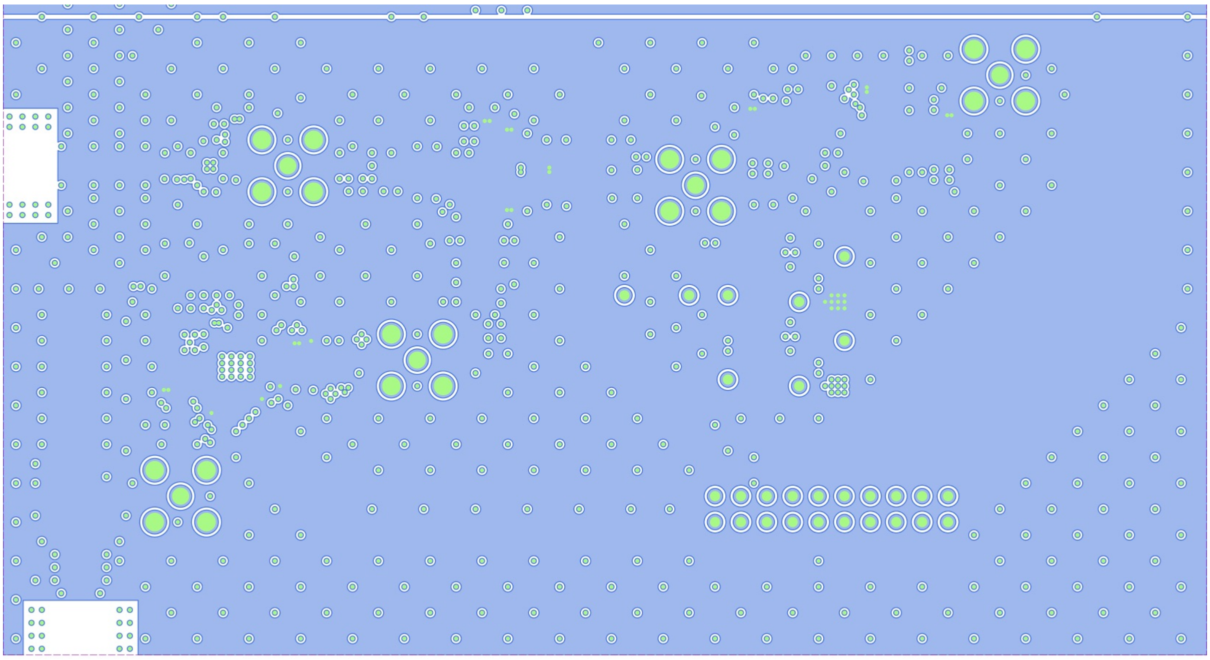


Figure A.29: DLL-HPC received signal beamformer PCB layout for element 0 inner layer 3

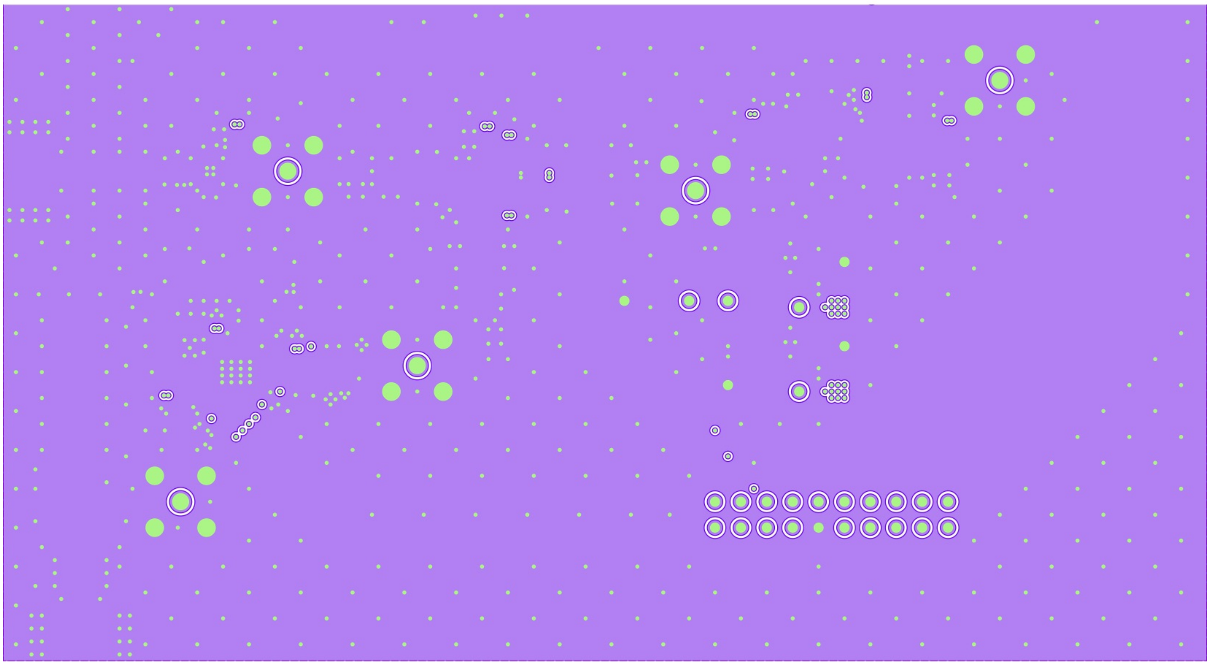


Figure A.30: DLL-HPC received signal beamformer PCB layout for element 0 inner layer 4

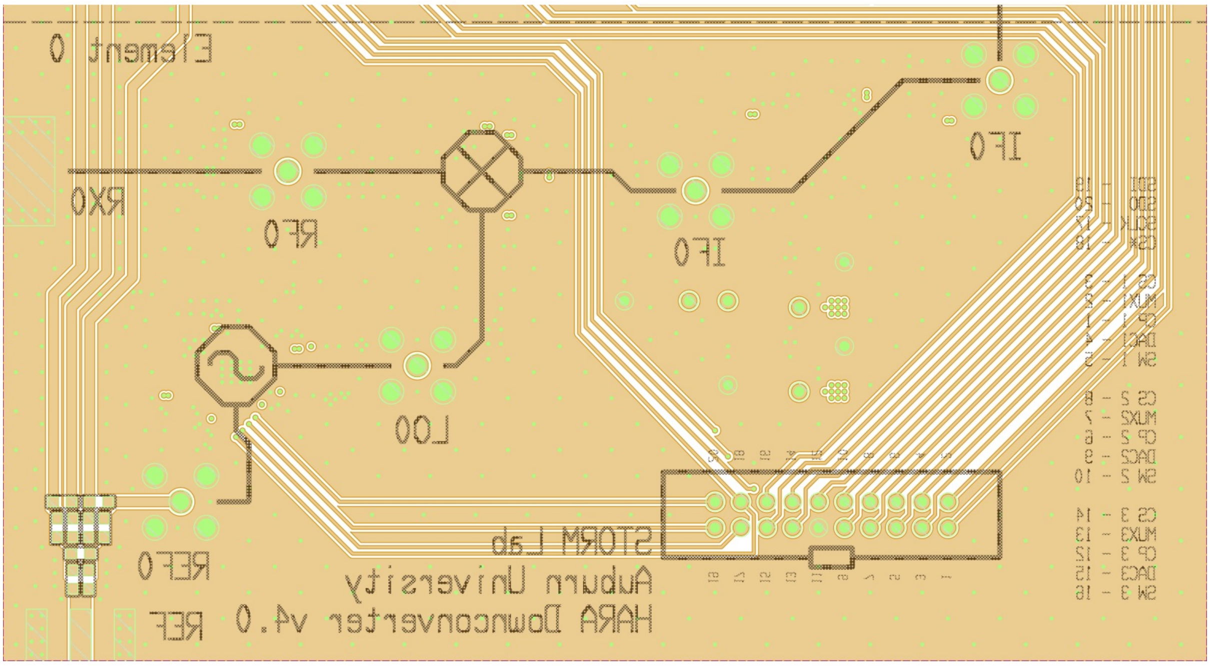


Figure A.31: DLL-HPC received signal beamformer back-side PCB layout for element 0

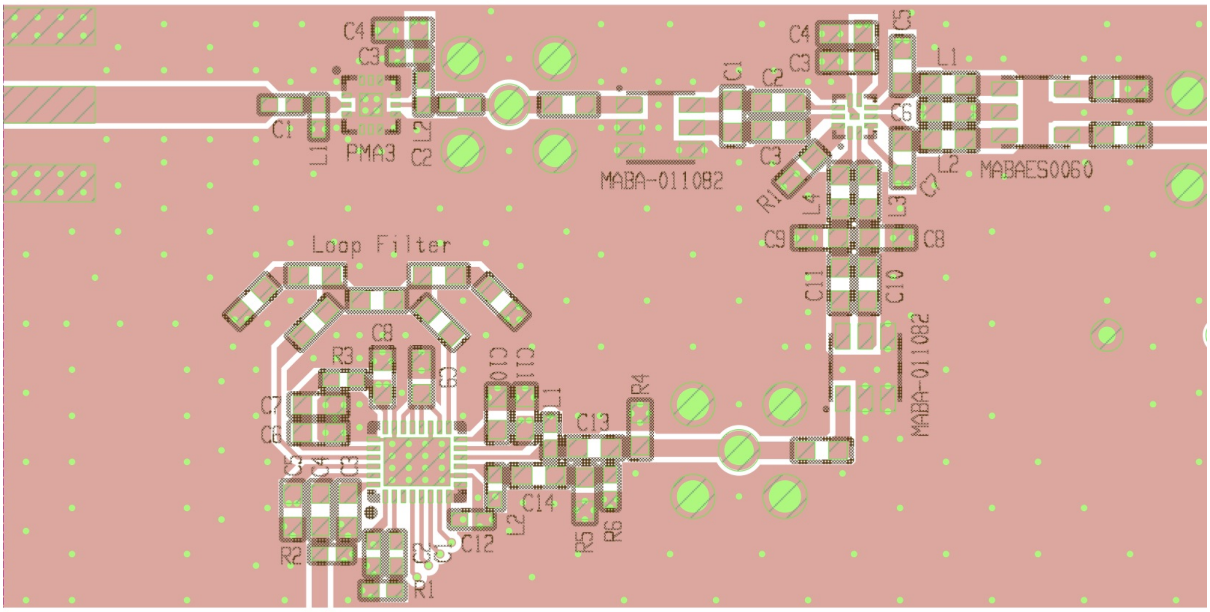


Figure A.32: DLL-HPC received signal beamformer front-side PCB layout for element 0 - RF input, low-noise amplifier, and PLL LO

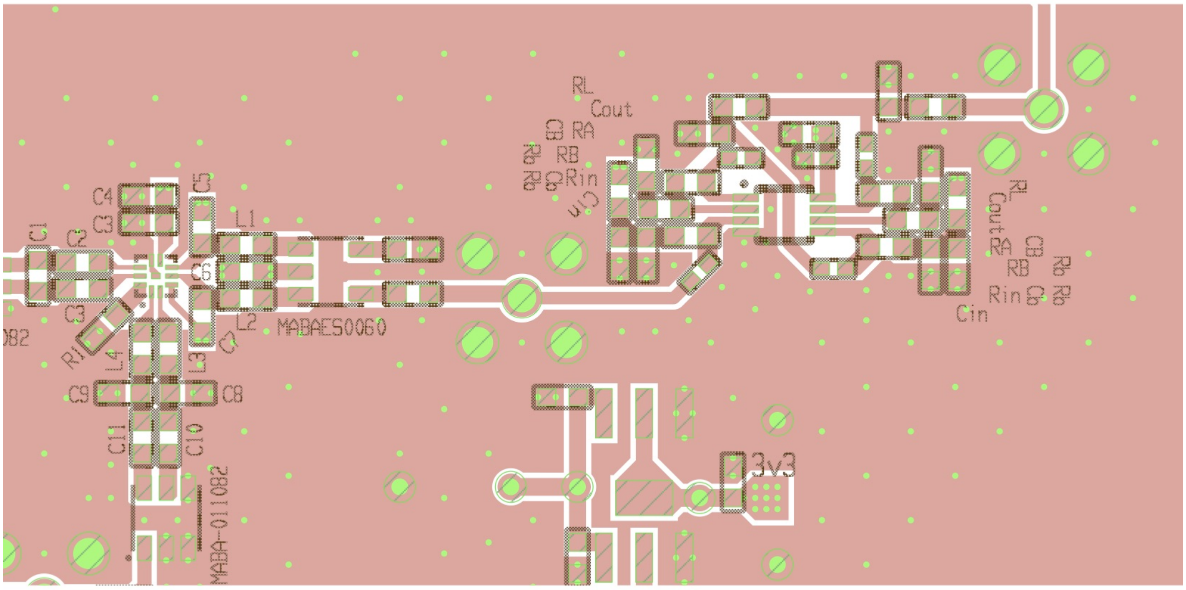


Figure A.33: DLL-HPC received signal beamformer front-side PCB layout for element 0 - downconverting mixer and IF amplifier with signal travelling to element 1's phase shifter

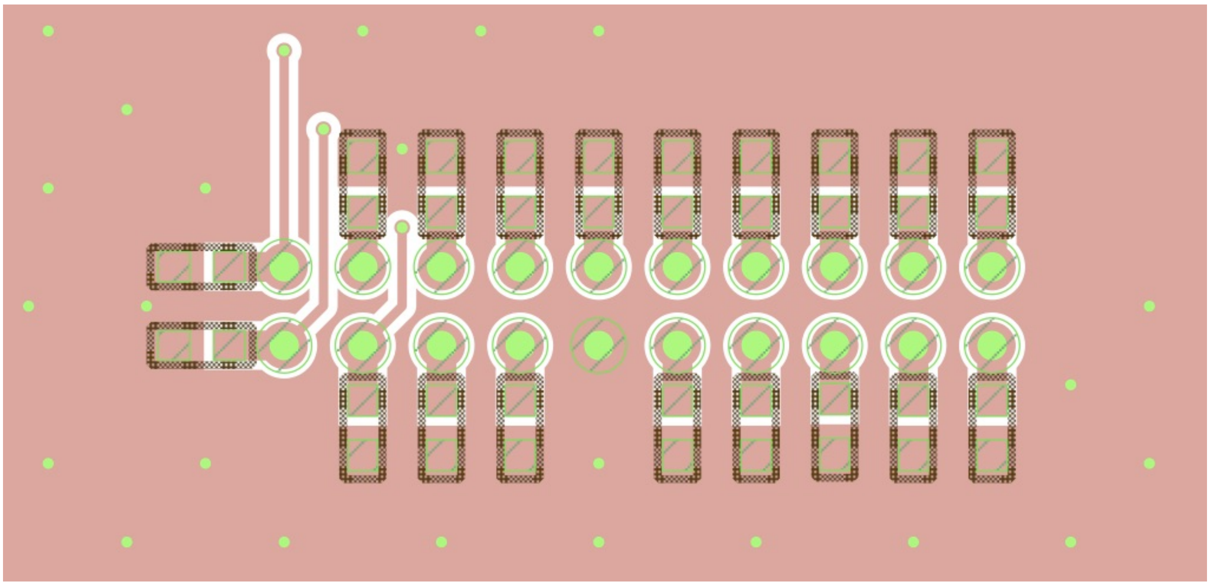


Figure A.34: DLL-HPC received signal beamformer front-side PCB layout for element 0 - ribbon cable passive connections

A.1.6 PCB Layout: Element n

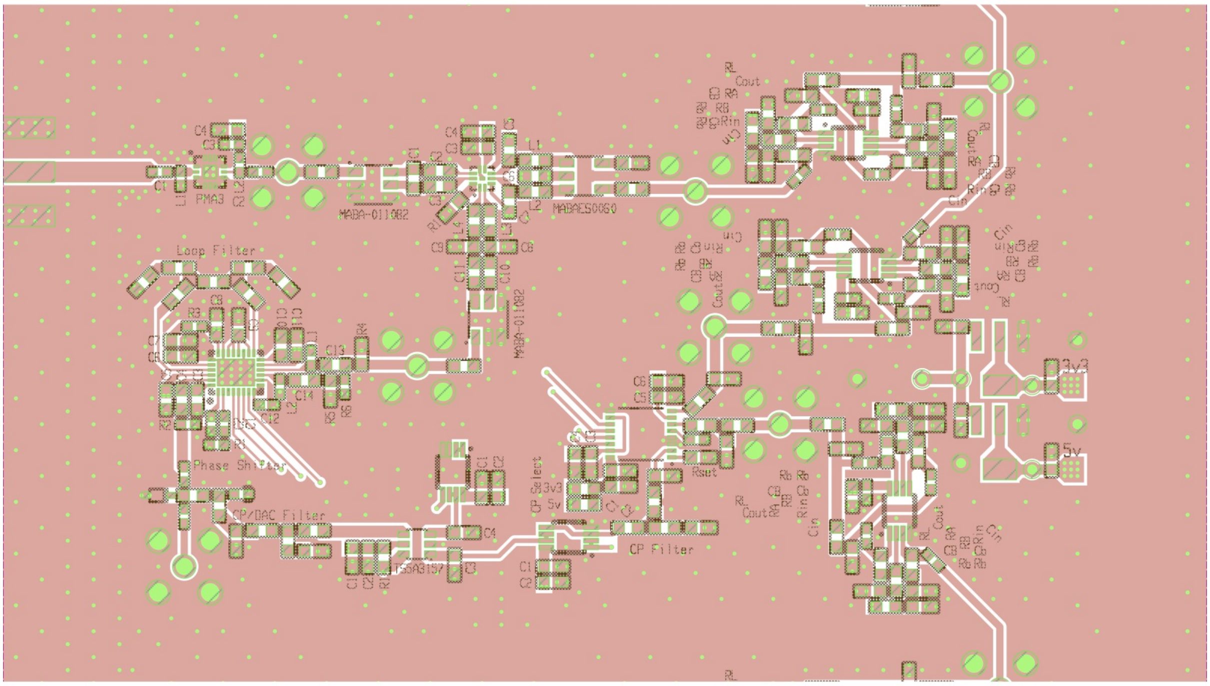


Figure A.35: DLL-HPC received signal beamformer front-side PCB layout for element 1; elements 2 and 3 are copies of element 1

Figure A.36: DLL-HPC received signal beamformer PCB layout for element 1 inner layer 1

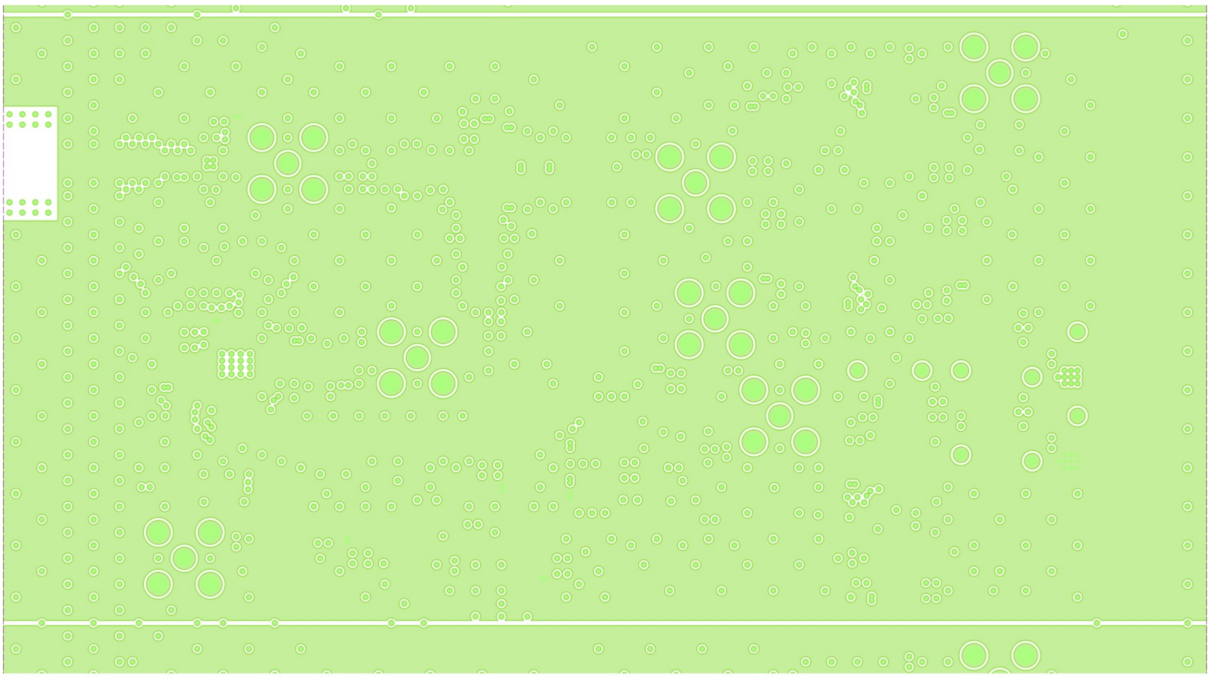


Figure A.37: DLL-HPC received signal beamformer PCB layout for element 1 inner layer 2

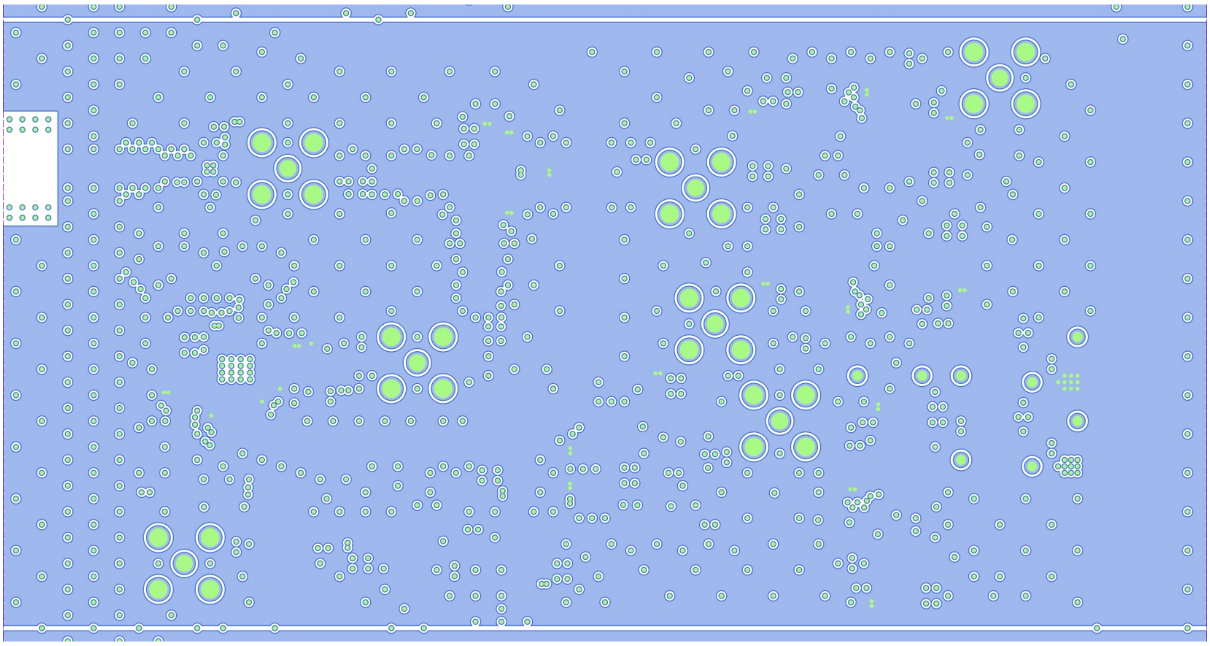


Figure A.38: DLL-HPC received signal beamformer PCB layout for element 1 inner layer 3

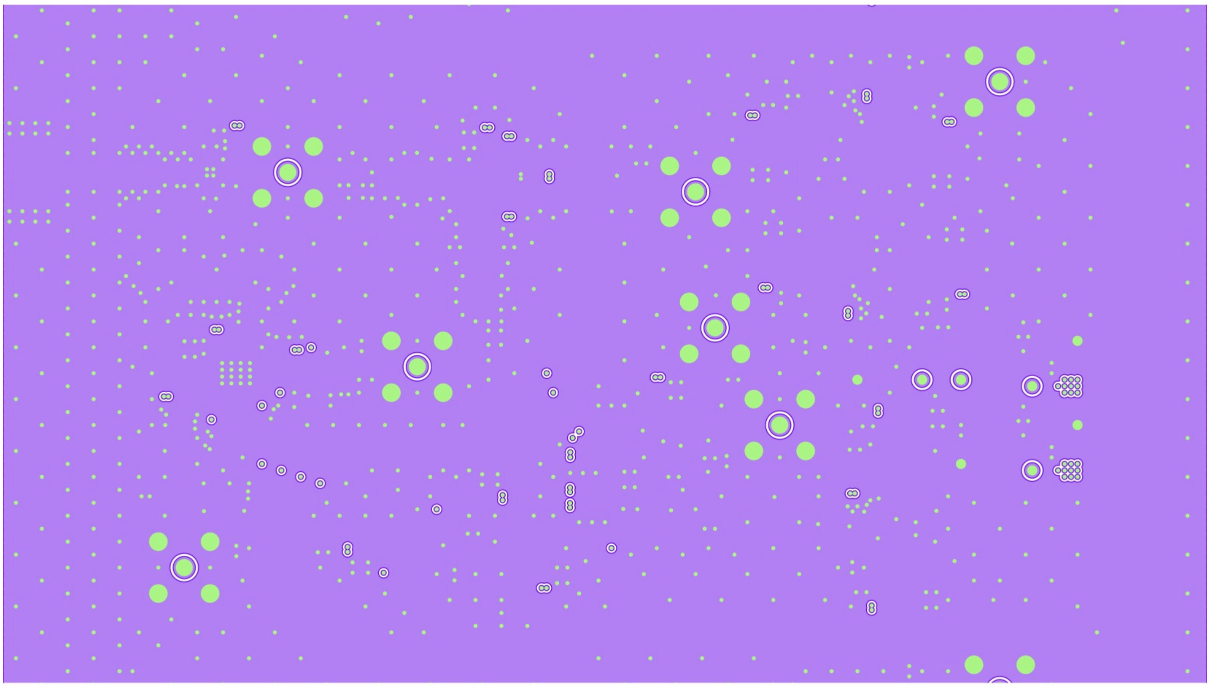


Figure A.39: DLL-HPC received signal beamformer PCB layout for element 1 inner layer 4

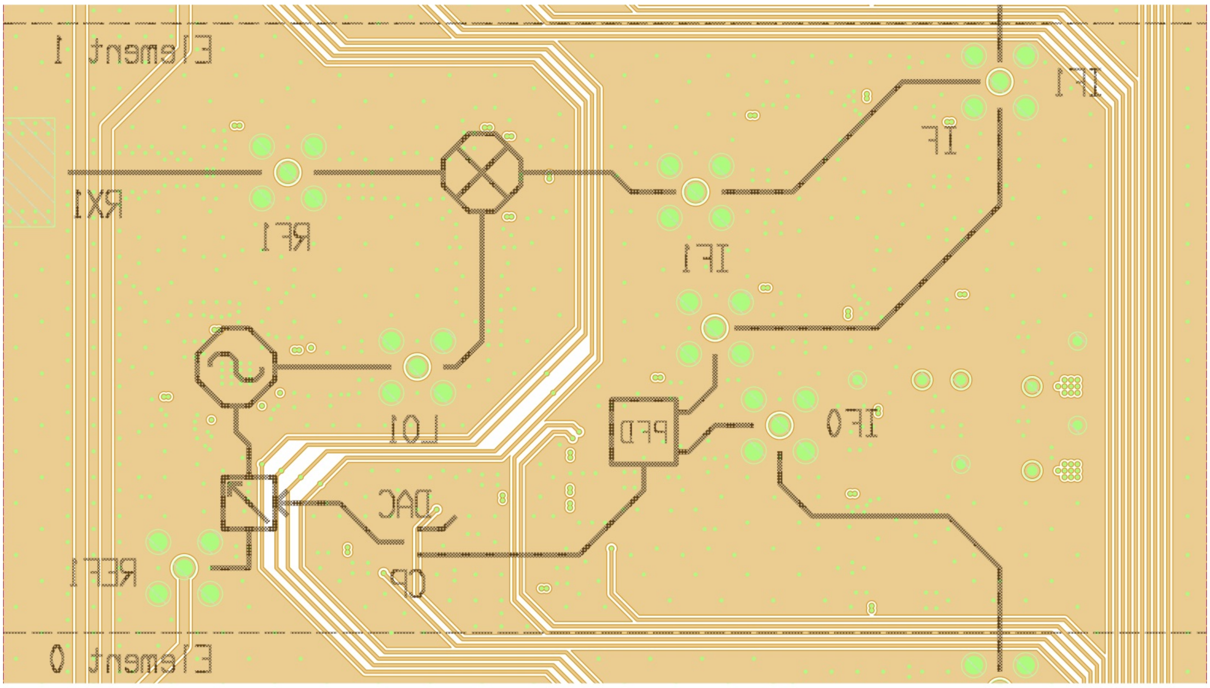


Figure A.40: DLL-HPC received signal beamformer back-side PCB layout for element 1

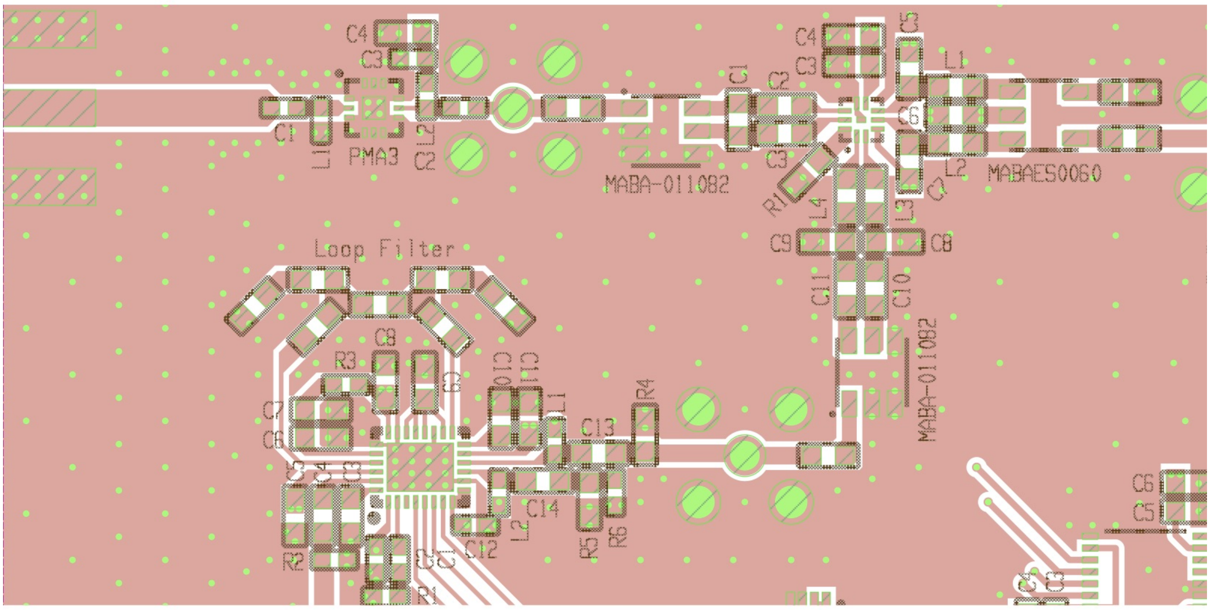


Figure A.41: DLL-HPC received signal beamformer front-side PCB layout for element 1 - RF input, low-noise amplifier, and PLL LO

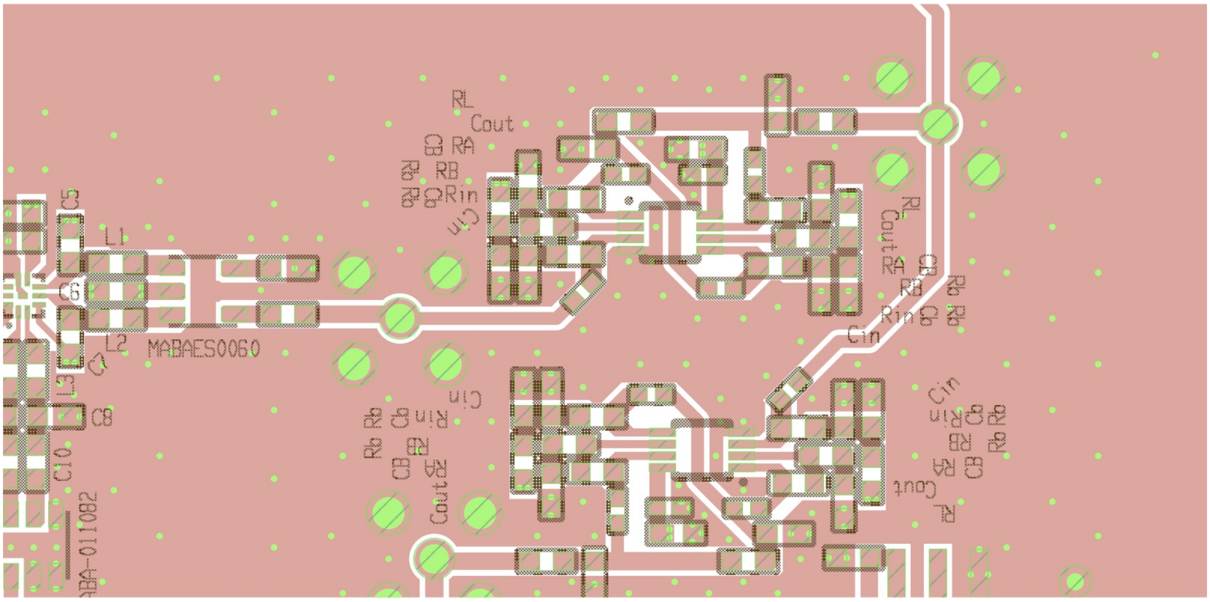


Figure A.42: DLL-HPC received signal beamformer front-side PCB layout for element 1 - downconverting mixer and IF amplifiers with signal travelling to element 2's phase shifter

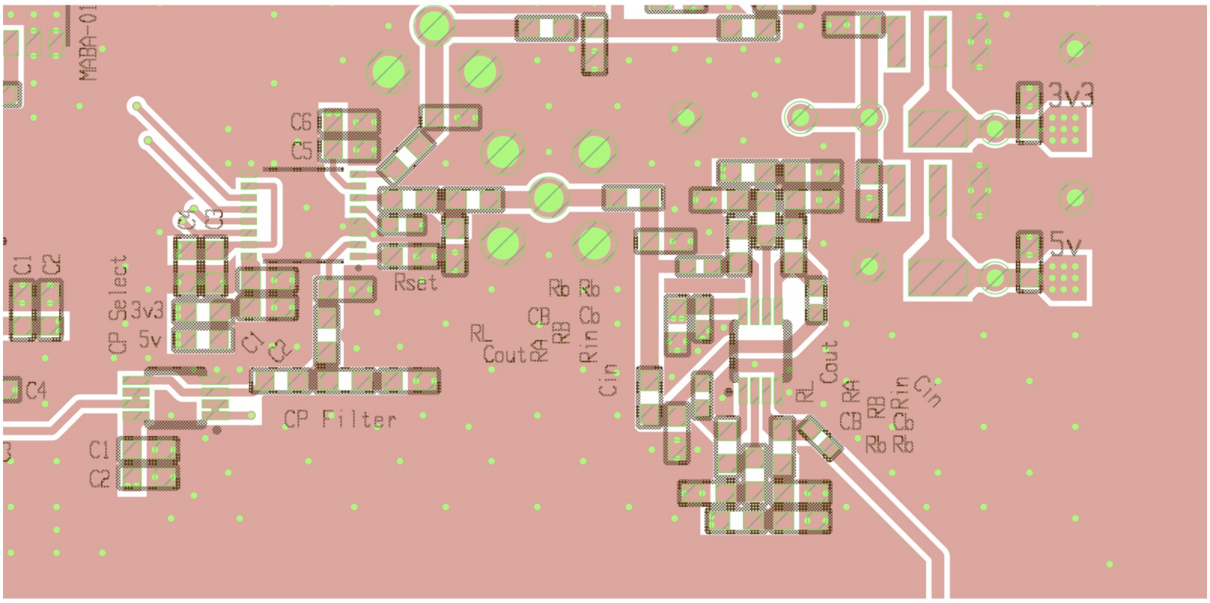


Figure A.43: DLL-HPC received signal beamformer front-side PCB layout for element 1 - IF amplifier for element 0's IF signal, PFD/CP phase detector, loop filter, and buffer circuitry

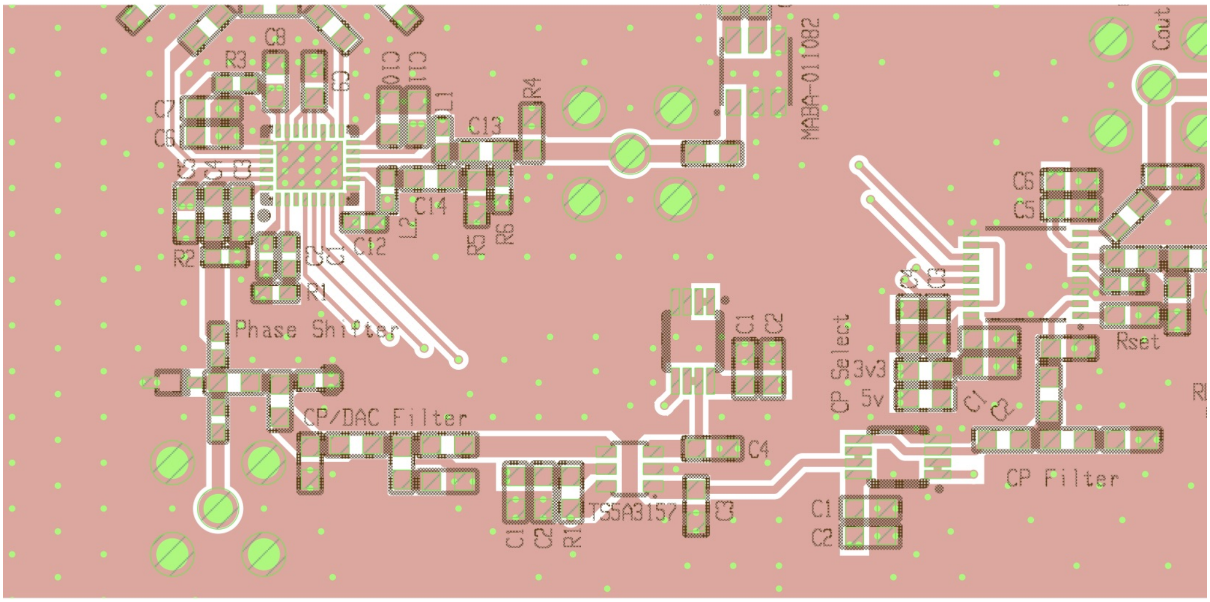


Figure A.44: DLL-HPC received signal beamformer front-side PCB layout for element 1 - PFD/CP, loop filter, switch, buffers, phase shifter, and PLL LO circuitry

A.2 Transmitted Signal Beamformer Design Files

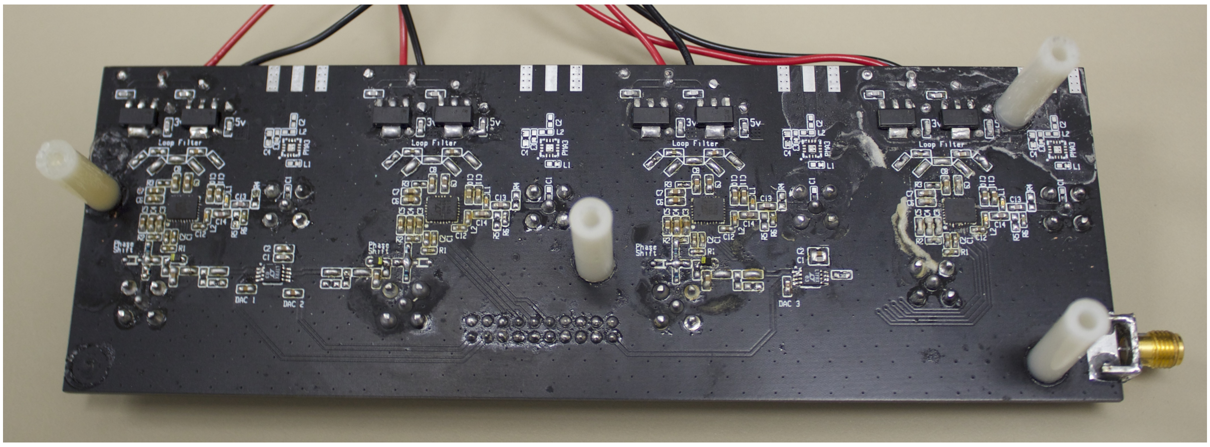


Figure A.45: Assembled DLL-HPC transmitted signal beamformer PCB - component side

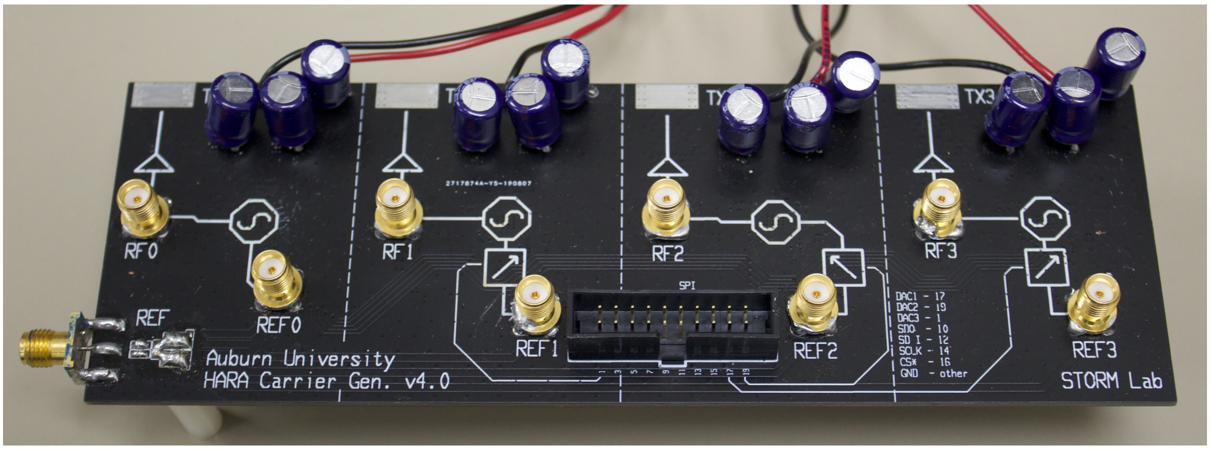


Figure A.46: Assembled DLL-HPC transmitted signal beamformer PCB - labeled side

A.2.1 PCB Renders

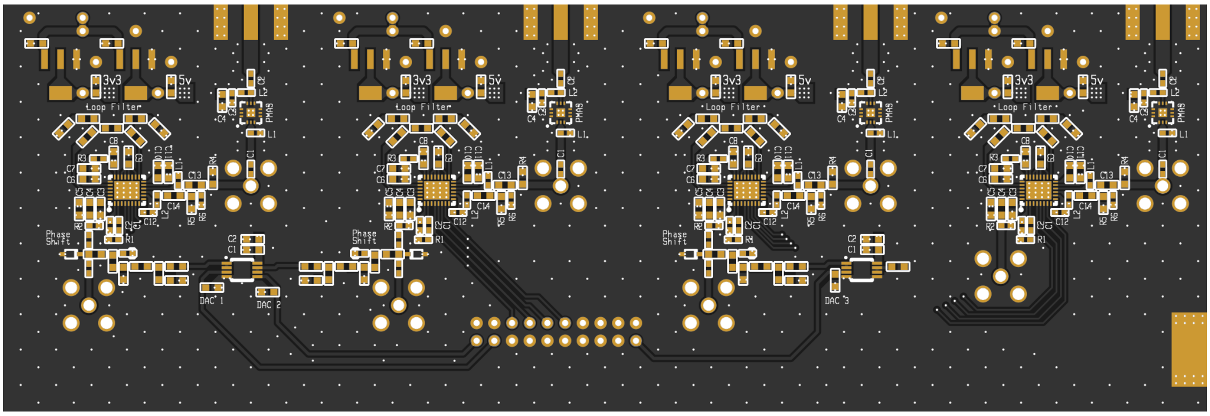


Figure A.47: DLL-HPC transmitted signal beamformer PCB front-side render

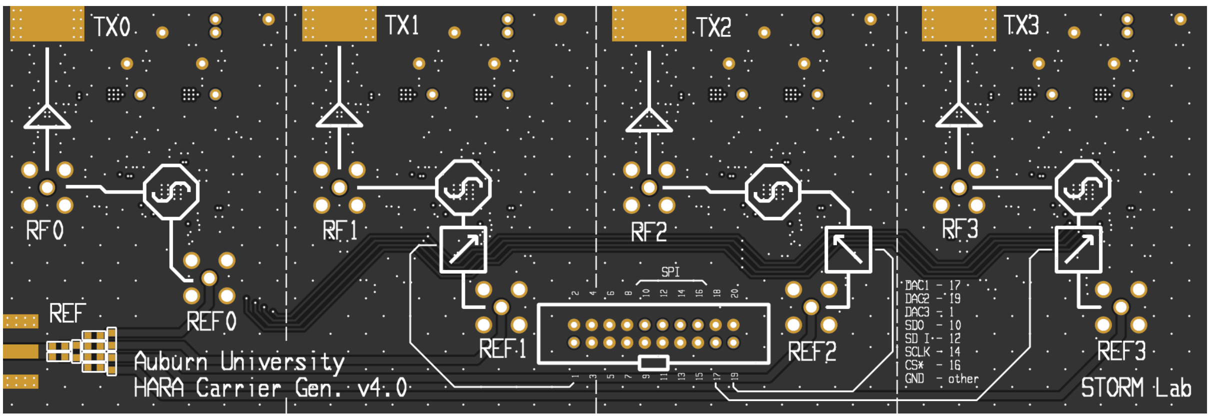


Figure A.48: DLL-HPC transmitted signal beamformer PCB back-side render

A.2.2 PCB Schematics

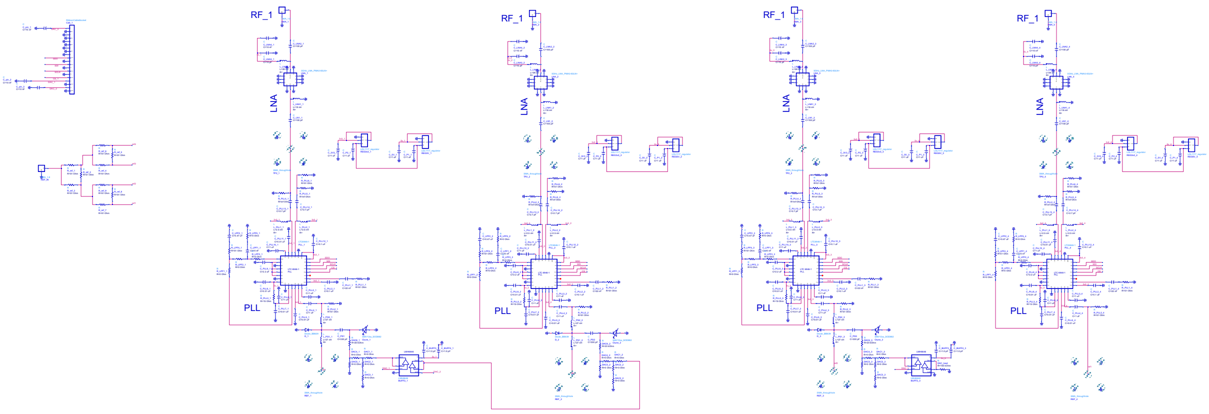
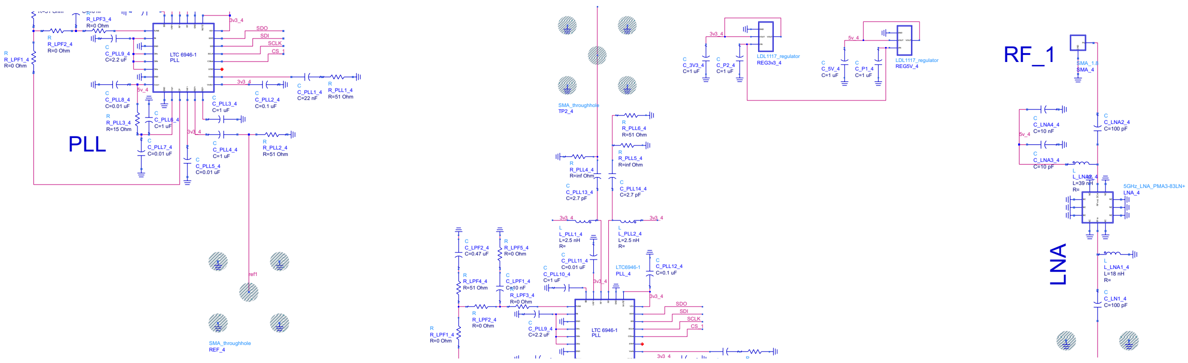
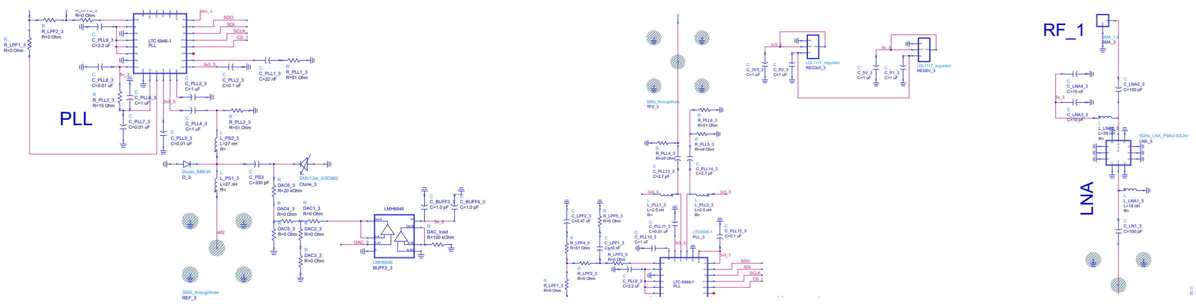


Figure A.49: DLL-HPC transmitted signal beamformer overall schematic



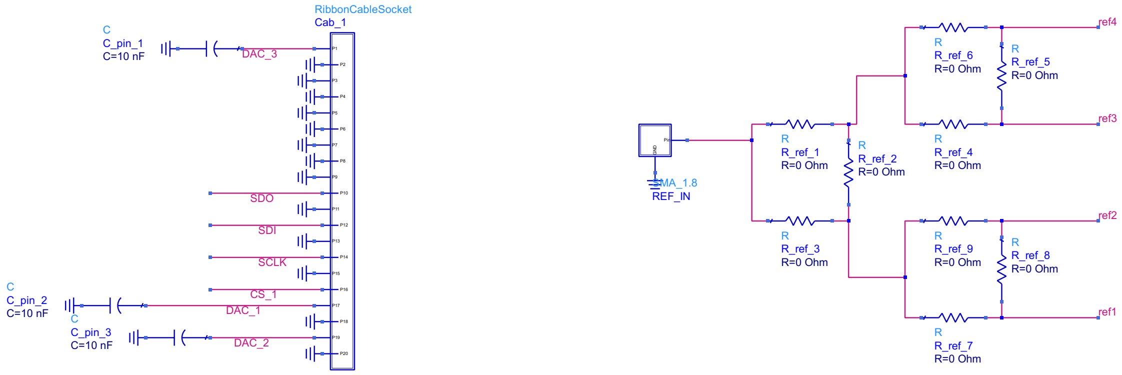
(a) (b) (c)

Figure A.50: DLL-HPC transmitted signal beamformer schematic for element 0 without phase shifter



(a) (b) (c)

Figure A.51: DLL-HPC transmitted signal beamformer schematic for element 1, which includes the phase shifter; elements 2 and 3 are copies of element 1



(a) (b)

Figure A.52: DLL-HPC transmitted beamformer schematic for (a) ribbon cable connections (b) PLL reference signal splitter

A.2.3 PCB Layout

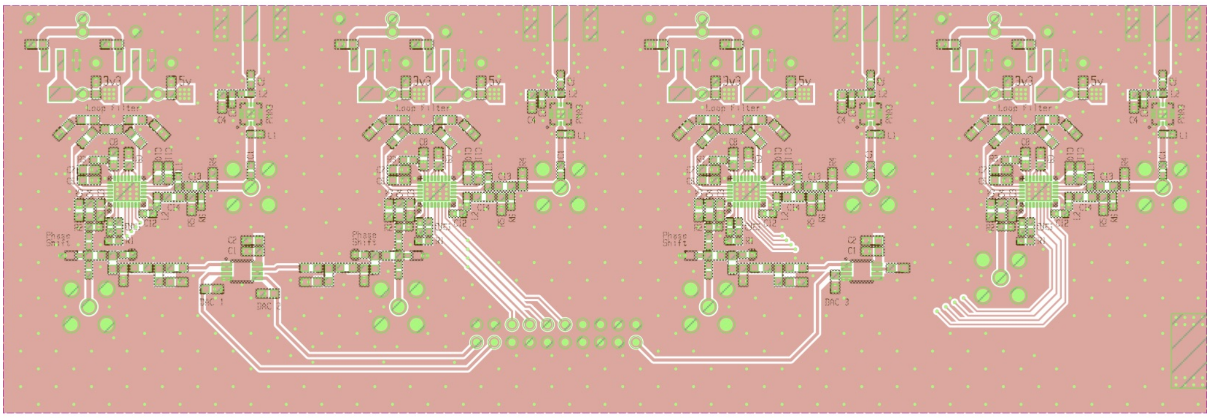


Figure A.53: DLL-HPC transmitted signal beamformer top-side PCB layout

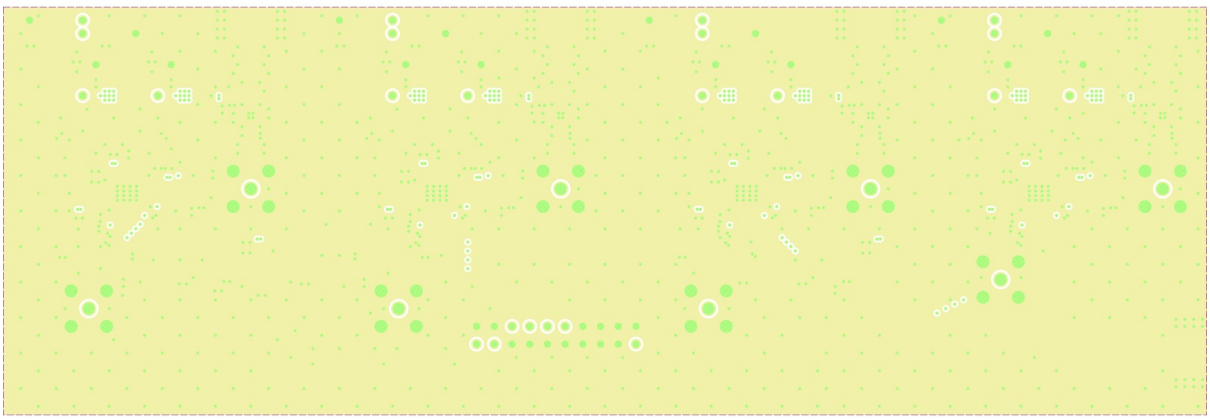


Figure A.54: DLL-HPC transmitted signal beamformer PCB inner layer 1

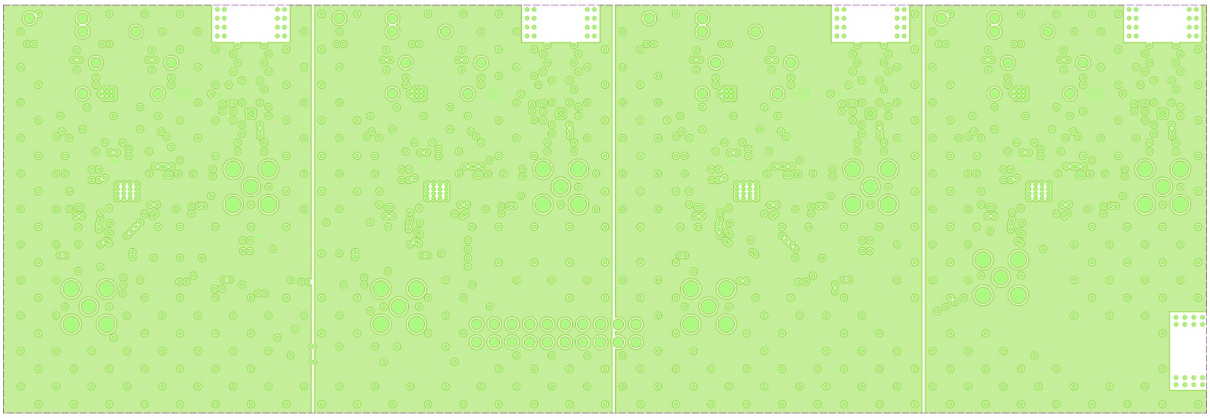


Figure A.55: DLL-HPC transmitted signal beamformer PCB inner layer 2

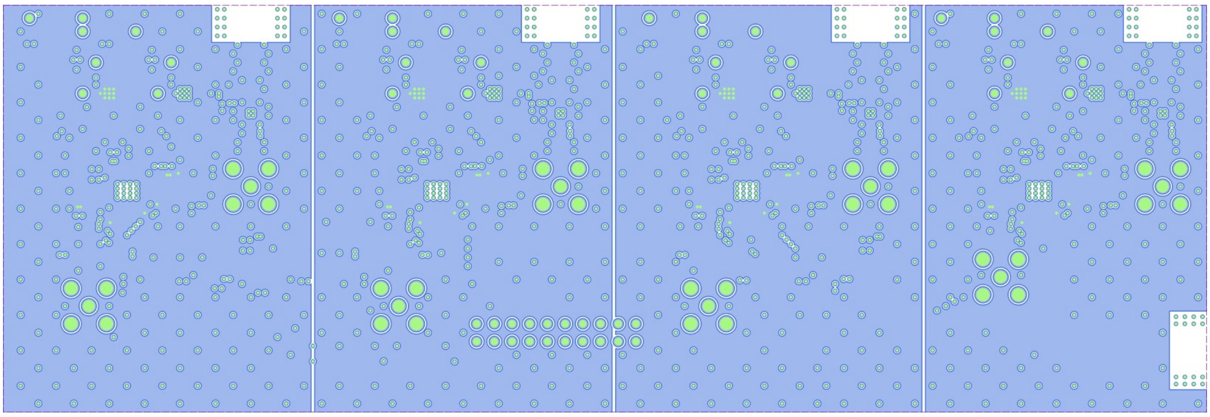


Figure A.56: DLL-HPC transmitted signal beamformer PCB inner layer 3

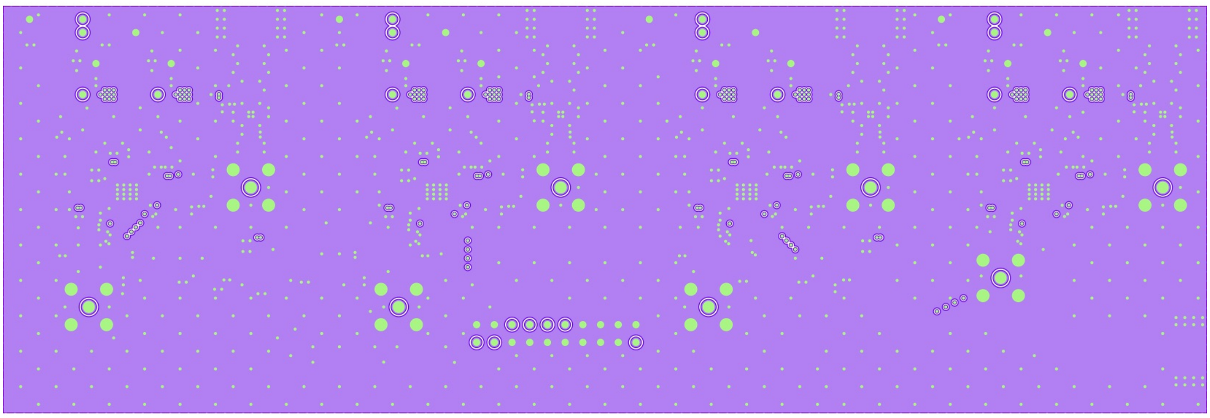
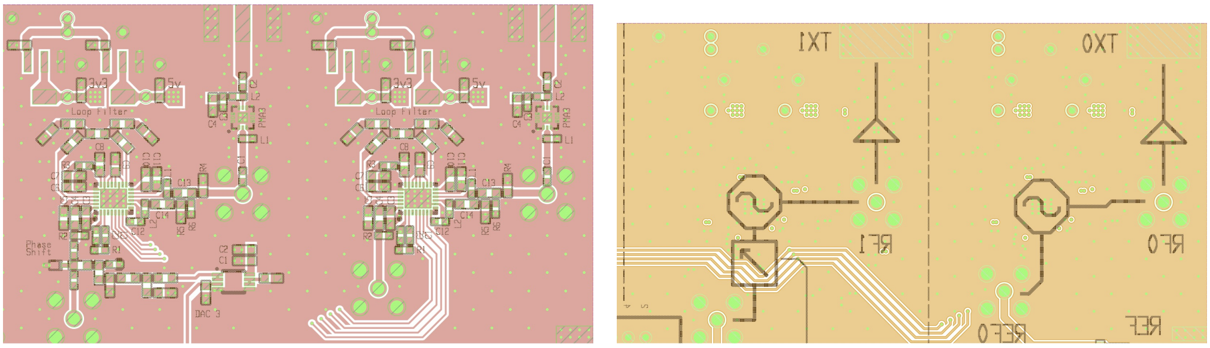
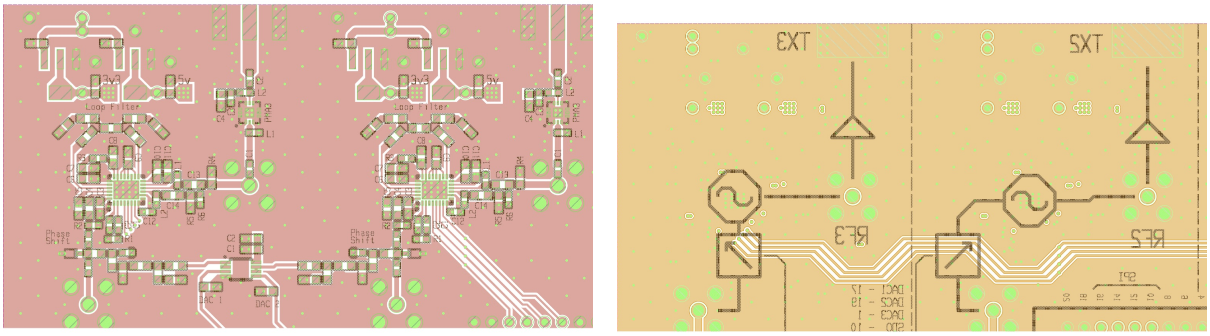


Figure A.57: DLL-HPC transmitted signal beamformer PCB inner layer 4



(a) (b)

Figure A.59: DLL-HPC transmitted signal beamformer PCB layouts for elements 0 and 1 (a) front-side (b) back-side



(a) (b)

Figure A.60: DLL-HPC transmitted signal beamformer PCB layouts for elements 2 and 3 (a) front-side (b) back-side

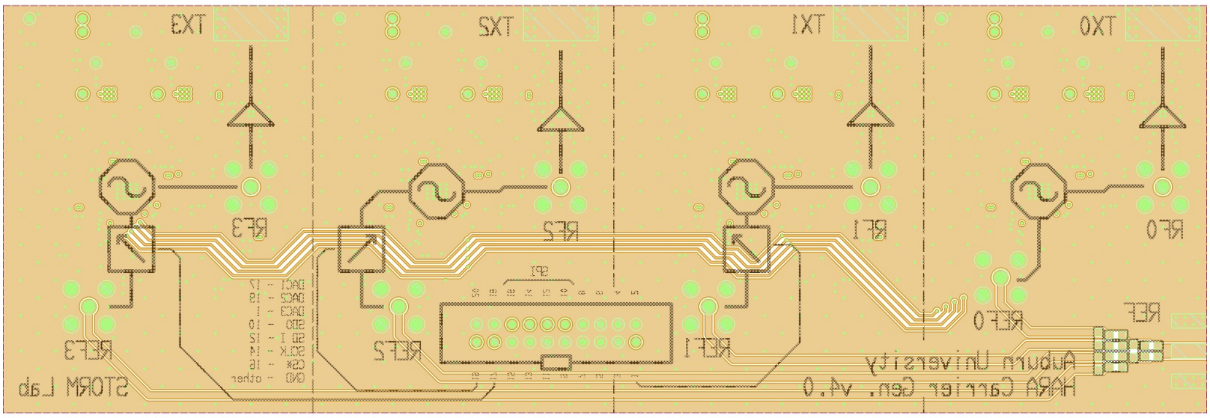


Figure A.58: DLL-HPC transmitted signal beamformer back-side PCB layout

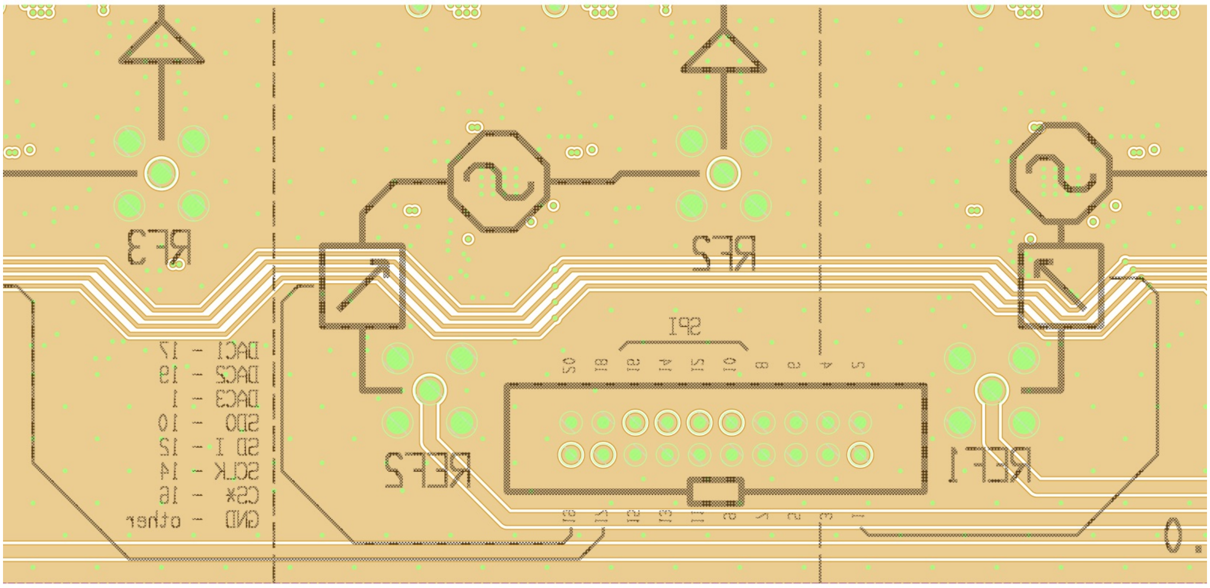
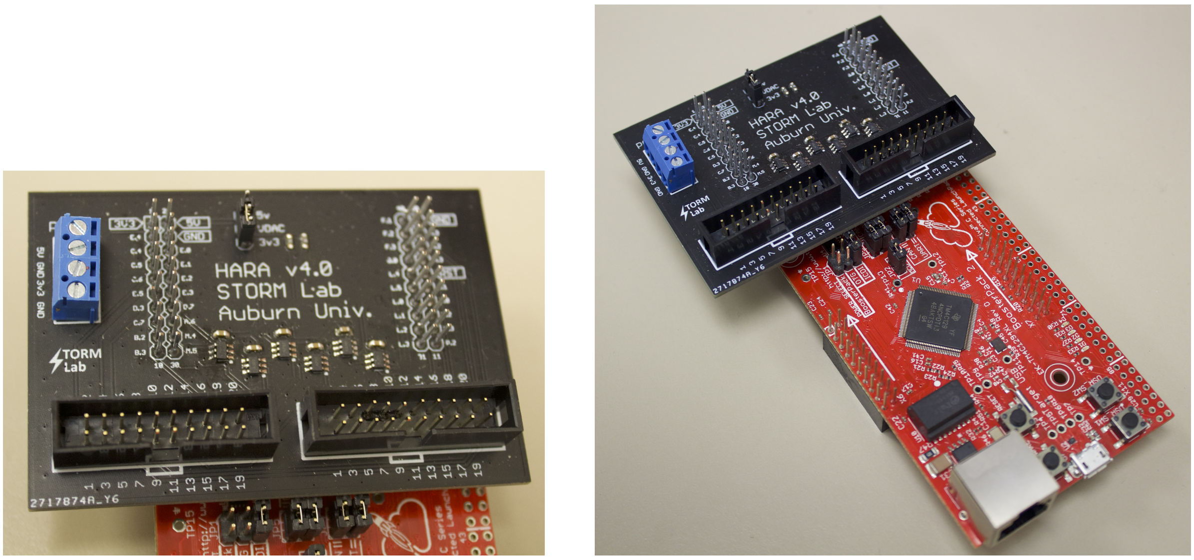


Figure A.61: DLL-HPC transmitted signal beamformer PCB layouts close up of ribbon cable documentation on back-side

A.3 TM4C BoosterPack Design Files



(a) (b)

Figure A.62: (a) TM4C BoosterPack (b) mounted on TM4C board

A.3.1 PCB Files

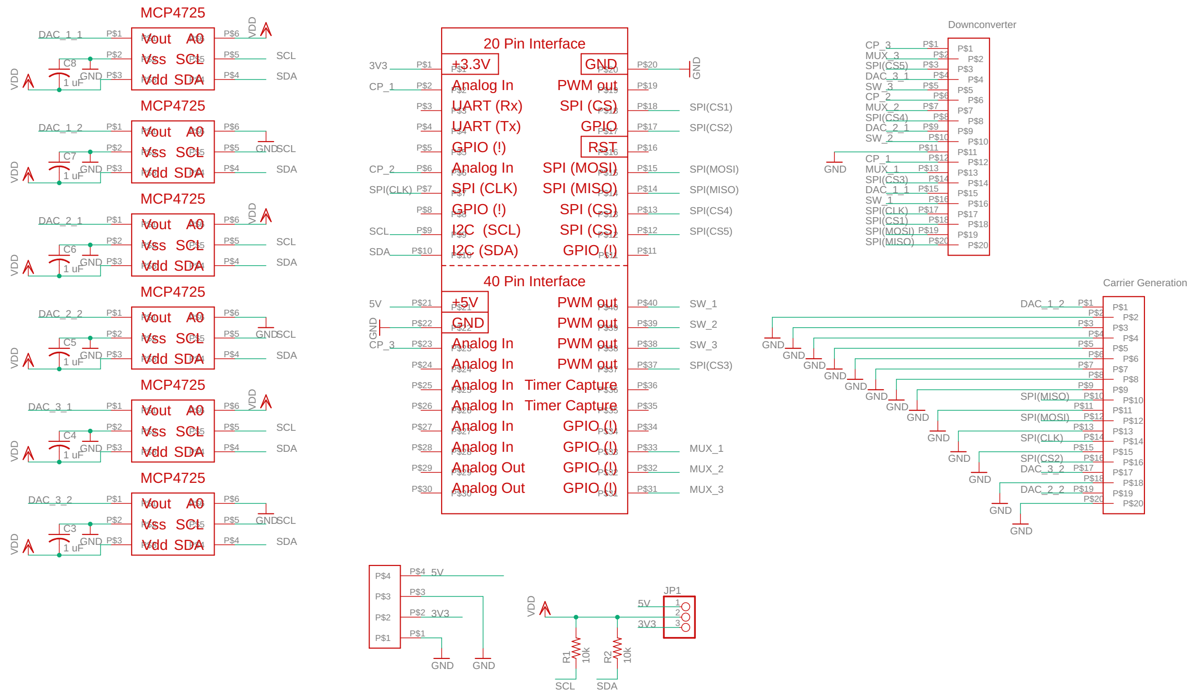


Figure A.63: TM4C BoosterPack schematic

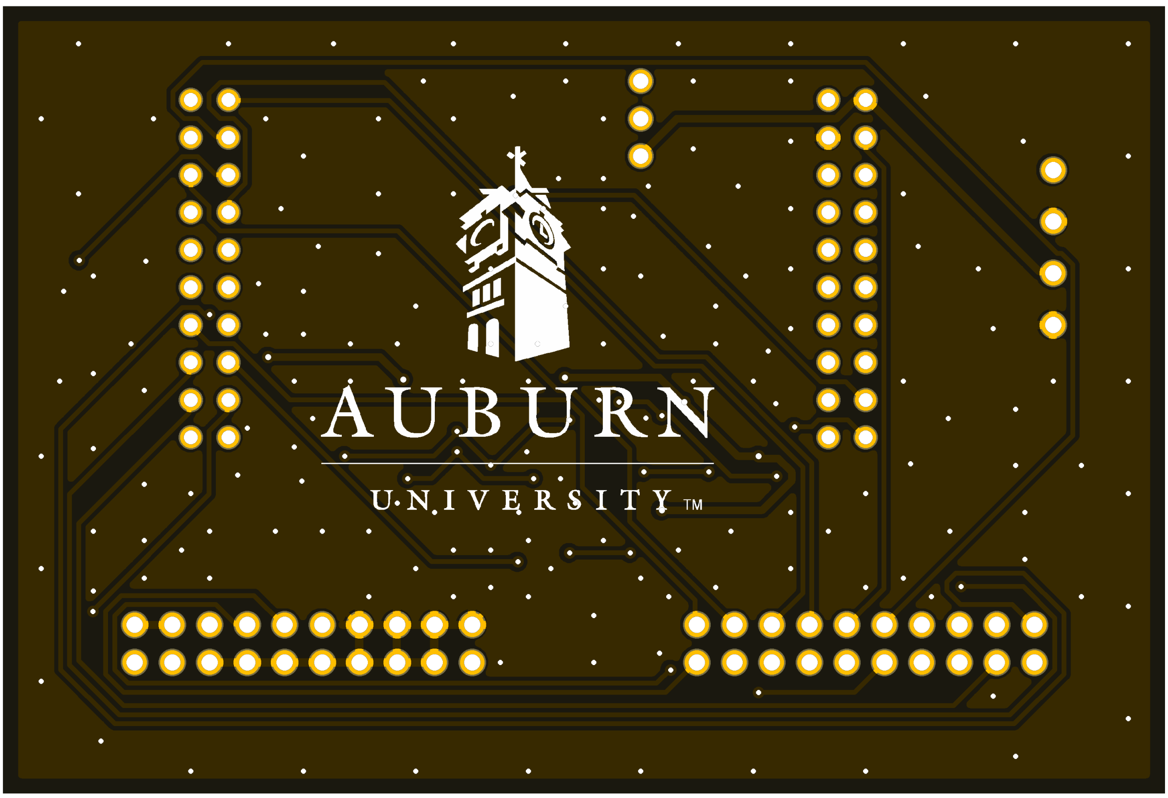


Figure A.64: TM4C BoosterPack backside render

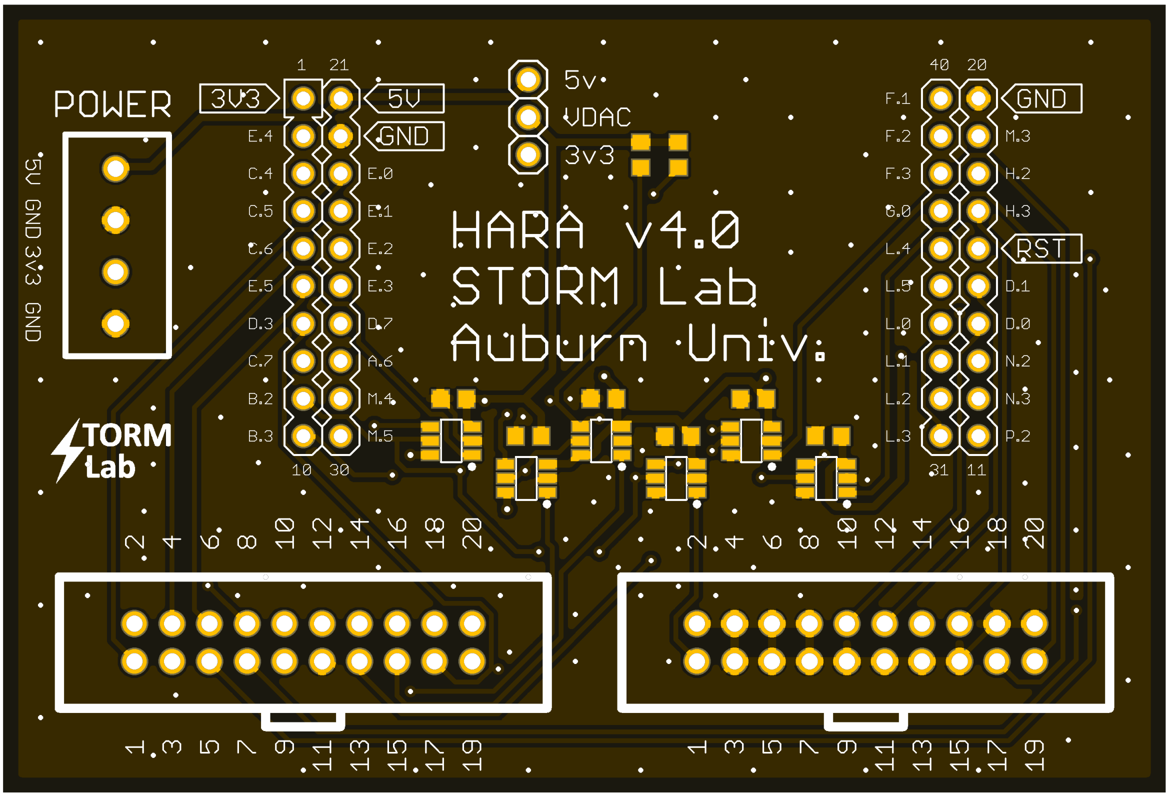


Figure A.65: TM4C BoosterPack topside render

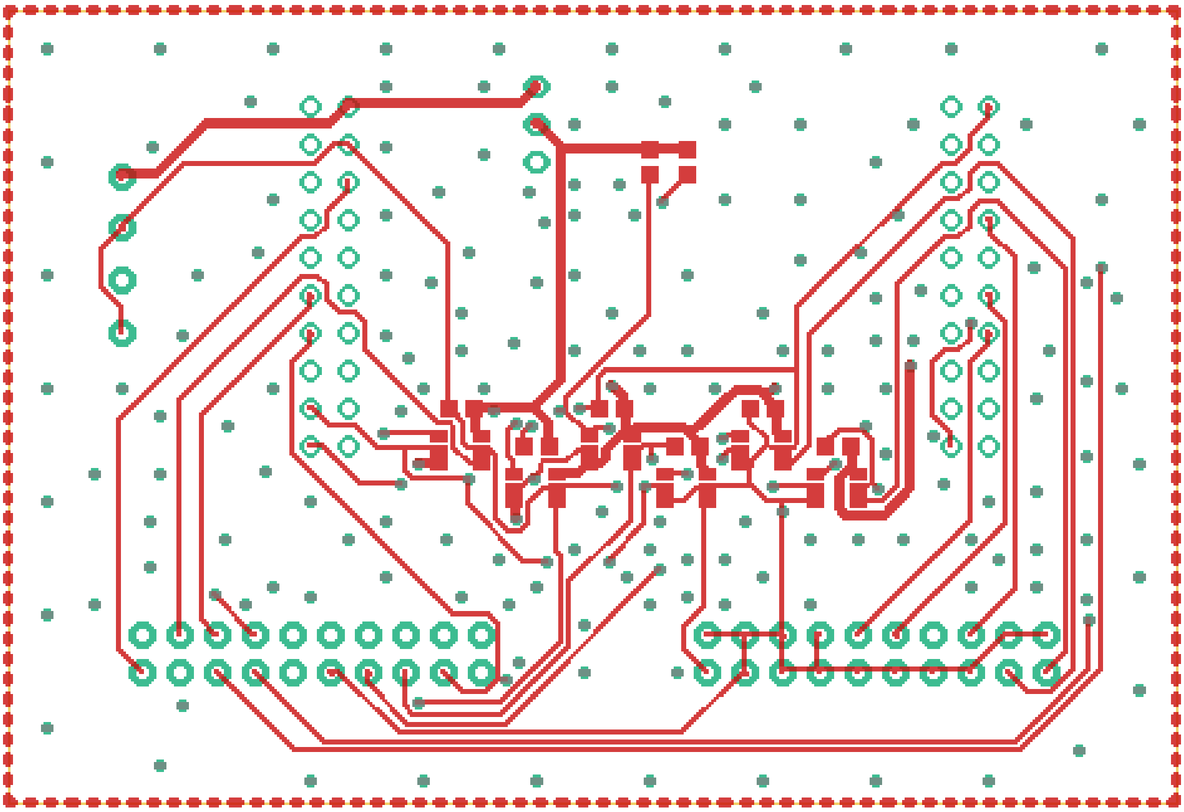


Figure A.66: TM4C BoosterPack topside copper layout

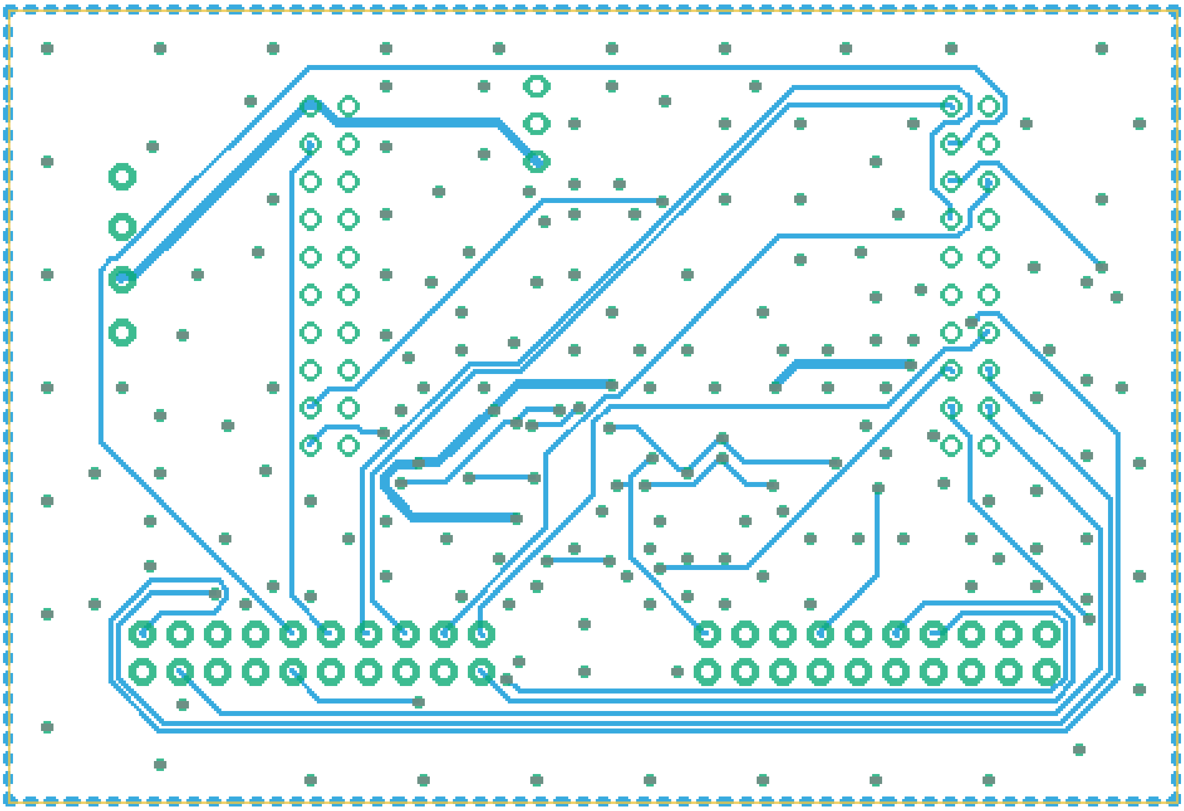


Figure A.67: TM4C BoosterPack backside copper layout

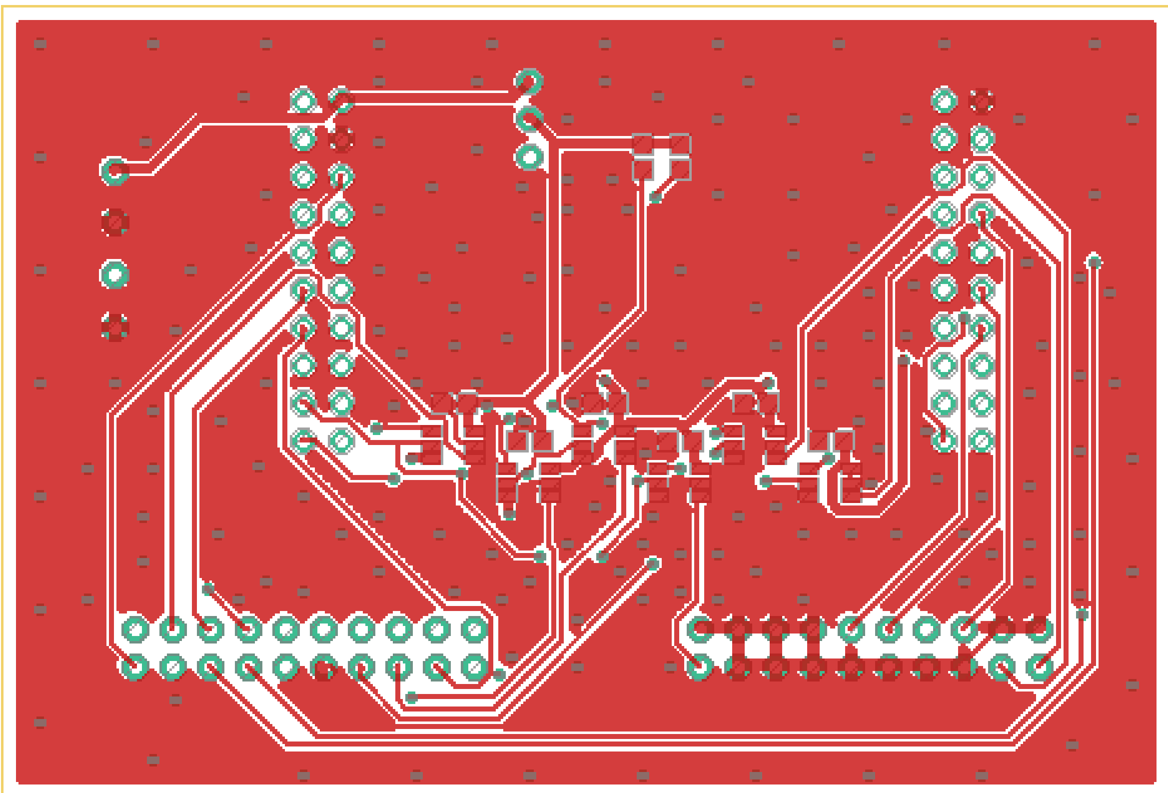


Figure A.68: TM4C BoosterPack topside copper layout with ground pour

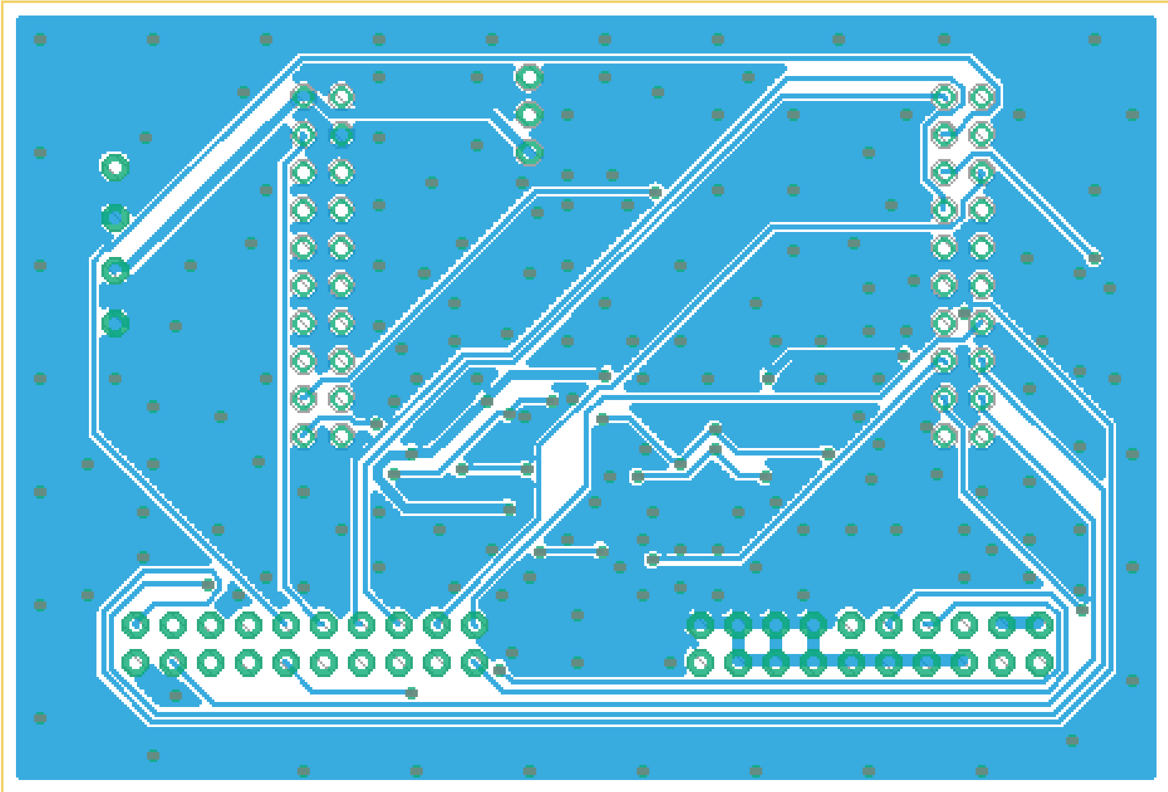


Figure A.69: TM4C BoosterPack backside copper layout with ground pout

A.3.2 BoosterPack Pin Connections

BoosterPack Pin #	Function	DLL-HPC Connection	TM4C Pin
1	Power	3.3 V	3V3
2	AnalogIn	CP 1 (A0)	PE4
3	UART (RX)		PC4
4	UART (Tx)		PC5
5	GPIO (!)		PC6
6	Analog In	CP 2 (A1)	PE5
7	SPI (CLK)	SPI (CLK)	PD3
8	GPIO (!)		PC7
9	I2C (SCL)	SCL	PB2
10	I2C (SDA)	SDA	PB3
11	GPIO (!)		PP2
12	SPI (CS)	SPI (CS 5)	PN3
13	SPI (CS)	SPI (CS 4)	PN2
14	SPI (MISO)	SPI (MISO)	PD0
15	SPI (MOSI)	SPI (MOSI)	PD1
16	RST		RST
17	GPIO	SPI (CS 2)	PH3
18	SPI (CS)	SPI (CS 1)	PH2
19	PWM out		PM3
20	Power	GND	GND
21	Power	5V	5V
22	Power	GND	GND
23	Analog In	CP 3 (A2)	PE0
24	Analog In		PE1
25	Analog In		PE2
26	Analog In		PE3
27	Analog In		PD7
28	Analog In		PA6
29	Analog Out		PM4
30	Analog Out		PM5
31	GPIO (!)	MUX 3	PL3
32	GPIO (!)	MUX 2	PL2
33	GPIO (!)	MUX 1	PL1
34	GPIO (!)		PL0
35	Timer Capture		PL5
36	Timer Capture		PL4
37	PWM out	SPI (CS 3)	PG0
38	PWM out	SW 3	PF3
39	PWM out	SW 2	PF2
40	PWM out	SW 1	PF1

Table A.1: TM4C BoosterPack pin connections

A.4 TM4C Firmware Source Code

A.4.1 main.c

```
//*********************************************************************
*****
//
//  HARA_BoosterPack.c
//
//  Adapted from project0.c project
//  intended to operate the HARA PLLs, DACs, and ADCs!
//
//*********************************************************************
*****

//static unsigned short phase_LUT[410];
//static unsigned short dac_LUT[3599];

#include <stdint.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include "inc/hw_types.h"
#include "inc/hw_memmap.h"
#include "inc/hw_ints.h"
#include "driverlib/sysctl.h"
#include "driverlib/interrupt.h"
#include "driverlib/gpio.h"
#include "driverlib/ssi.h"
#include "driverlib/i2c.h"
#include "driverlib/adc.h"
#include "driverlib/timer.h"
#include "driverlib/pin_map.h"
#include "HARABoosterPack.h"
#include "FIFO.h"
#include "HARA_LUTs.h"



//PLL structs
extern LTC_6946 PLL1;
extern LTC_6946 PLL2;
//HARA Structures
extern HARA hara[4];

//FIFO buffer
extern FIFO cmd_fifo;
extern unsigned int cmd;

//ADC globals
extern unsigned int ADC_data[];
extern unsigned int adc_waitForSample;

//state character for transmission to the host
extern unsigned char state;
```

```
//*****************************************************************************
//
// Main 'C' Language entry point.
//
//*****************************************************************************
int
main(void)
{
    //use 120 MHz PLL as SysClock
    uint32_t ui32SysClock;
    ui32SysClock = SysCtlClockFreqSet((SYSCTL_XTAL_25MHZ |
                                       SYSCTL_OSC_MAIN |
                                       SYSCTL_USE_PLL |
                                       SYSCTL_CFG_VCO_480), 120000000);
    //create FIFO buffer
    clearFIFO(&cmd_fifo);

    //////////////////////////////
    //  Configure Blinky LEDs   //
    //////////////////////////////
    //configure PN.0 and PN.1 for LEDs
    //enable GPION and wait for it to be ready
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPION);
        while(!SysCtlPeripheralReady(SYSCTL_PERIPH_GPION));
    //set pin directions
    GPIOPinTypeGPIOOutput(GPIO_PORTN_BASE, (GPIO_PIN_0|GPIO_PIN_1));
    GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_1, ~GPIO_PIN_1);

    //configure the BoosterPack peripherals
    HARABoosterPack_init(ui32SysClock, 10000, true);

    //initialize all angless to 0 degrees (theta = 0 degrees)
    int i = 3;
    for (i = 3; i != 0; i--) {
        //set theta and xi to 0 degrees
        writePhase(&hara[i], PS_TX, 0);
        writePhase(&hara[i], PS_RX, 0);
        //set to CP mode
        changeSW(&hara[i], SW_CP);
    }


    unsigned int    potVal;
    signed int      point;
    signed int      offset;
    unsigned char   num;
    unsigned char   dac;
    unsigned char   dBm;
    unsigned char   Odiv;
    unsigned char   mux;
    unsigned char   icp;
    unsigned char   apbw;
    unsigned char   polarity;
    unsigned char   mux_state;
```

```
    unsigned char   pkt[21];
    uint16_t r;
    uint16_t n;
    //eternal loop
    while(1)
    {
        //if there is a message
        if(!isEmpty(&cmd_fifo)) {
            //pop command
            cmd = pop(&cmd_fifo);
            //execute command
            switch(cmd & 0xF000) {
                ////////////////////////////////////////
                //  UART - adjust delta_i/j manually  //
                ////////////////////////////////////////
                case CMD_SW_ADJUST_DELTA_MANUALLY:
                    //change state
                    state = STATE_FINDING_PHASE_OFFSET;
                    //parse num, i/j, and offset and sign extend
                    num = (cmd & 0x0C00) >> 10;
                    dac = (cmd & 0x0100) >> 8;
                    offset = cmd & (0x00FF);
                    if (offset & 0x00080) {
                        offset |= 0xFFFFFF00;
                    }
                    // update offset
                    if (dac)    hara[num].delta_j += offset;
                    else        hara[num].delta_i += offset;
                    //rewrite current angle
                    if (dac)    writePhase(&hara[num], PS_TX, hara[num].xi);
                    else        writePhase(&hara[num], PS_RX, hara[num].theta);
                    //change state
                    state = 0;
                break;

                ///////////////////////////////
                //  UART - set delta_i by CP //
                ///////////////////////////////
                case CMD_SW_SET_DELTAS_BY_CP:
                    state = STATE_FINDING_PHASE_OFFSET;
                    //parse element number
                    num = (cmd & 0x0C00) >> 10;
                    //in lock with psi = 0:
                    //  theta = 0 - d_i = cpLUT[adc]
                    //  cpLUT[adc] = -d_i -> delta_i
                    hara[num].delta_i = cpLUT(hara[num].vadc, num);
                    //update delta_j so that the current angle is 0
                    hara[num].delta_j += hara[num].xi;
                    writePhase(&hara[num], PS_TX, 0);
                    //change state
                    state = 0;
                break;

                //////////////////////////
                //  UART - active steer //
```

```
            //////////////////////////
            case CMD_SW_ACTIVE_STEER:
                state = STATE_ACTIVE_STEER;
                //parse point value and sign extend
                point = (cmd & 0x0FFF);
                if(point & 0x0800)
                    point |= 0xFFFFF000;
                //point array
                pointArray(point, PS_TX);
                pointArray(point, PS_RX);
            break;

            ////////////////////////////////
            //  UART - stop active steer  //
            ///////////////////////////////
            case CMD_SW_STOP_ACTIVE_STEER:
                //change back to CP operation
                for (num = 3; num != 0; num--) {
                    changeSW(&hara[num], SW_CP);
                }
                //set state back to 0
                state = 0;
            break;

            //////////////////////////
            //  UART - Retrodirect  //
            //////////////////////////
            case CMD_SW_RETRODIRECT:
                state = STATE_RETRODIRECT;
                //kick off first ADC sample
                adc_waitForSample = ADC_SAMPLE_COUNT;
                //wait for next ADC sample
                while(adc_waitForSample);
                //kick off the ADC sample for the next counter
                adc_waitForSample = ADC_SAMPLE_COUNT;
                //perform element-wise phase conjugation
                for (num = 3; num != 0; num--) {
                    conjugate(&hara[num]);
                }
                //state is not set to 0 until CMD_SW_STOP_RETRODIRECT
                is receieved
            break;

            //////////////////////////////////
            //  UART - stop retrodirecting  //
            //////////////////////////////////
            case CMD_SW_STOP_RETRODIRECT:
                //return elements to 0 degrees
                for (num = 3; num != 0; num--) {
                    writePhase(&hara[num], PS_TX, 0);
                }
                //clear retrodirecting state
                state = 0;
            break;


            //////////////////////////
```

```
//  UART - Phase write  //
//////////////////////////
case CMD_SW_PHASE_WRITE:
    num = (cmd & 0x0C00) >> 10;
    dac = (cmd & 0x0200) >> 9;
    potVal = (cmd & 0x01FF);
    //write phase and mode
    writePhase(&hara[num], dac, potVal);
break;

//////////////////////////
//  UART - DAC write    //
//////////////////////////
case CMD_SW_DAC_WRITE:
    num = (cmd & 0x0F00) >> 8;
    dac = (cmd & 0x00F0) >> 4;
    cmd = pop(&cmd_fifo);
    if (dac) {
        if (DAC_write(&hara[num].DAC_tx, cmd))
            hara[num].txDAC = cmd;
    }
    else {
        //make sure to set SW to DAC
        changeSW(&hara[num], SW_DAC);
        if (DAC_write(&hara[num].DAC_rx, cmd))
            hara[num].rxDAC = cmd;
    }
break;

//////////////////////
//  UART PLL Recal  //
//////////////////////
case CMD_SW_PLL_RECAL:
    switch(cmd & 0x000F) {
    case 1:
        LTC_recal(&PLL1);
        break;
    case 2:
        LTC_recal(&PLL2);
        break;
    }
break;

//////////////////////////////
//  UART PLL Reconfigure    //
//////////////////////////////
case CMD_SW_PLL_RECONFIGURE:
    num  = (cmd & 0x0F00) >> 8;
    dBm = (cmd & 0x00F0) >> 4;
    Odiv = (cmd & 0x000F);
    cmd = pop(&cmd_fifo);
    switch(num) {
    case 1:
        LTC_init(&PLL1, 1, LTC6946_R_DIV, cmd, Odiv, dBm);
        LTC_configure(&PLL1);
        break;
    case 2:
```

```
                LTC_init(&PLL2, 2, LTC6946_R_DIV, cmd, Odiv, dBm);
                LTC_configure(&PLL2);
                break;
            }
        break;

        //////////////////////////////////////
        //  UART PFD Mux Out Reconfigure    //
        //////////////////////////////////////
        case CMD_SW_CHANGE_PFD_MUX_OUT:
            GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_1, GPIO_PIN_1);
            num =        (cmd & 0x0E00) >> 9;
            mux =        (cmd & 0x01C0) >> 6;
            icp =        (cmd & 0x0038) >> 3;
            apbw =       (cmd & 0x0006) >> 1;
            polarity =   (cmd & 0x0001);
            // pop R divider and N divider
            r = pop(&cmd_fifo);
            n = pop(&cmd_fifo);
            //reconfigure PFD
            PFD_init(&hara[num].PFD, hara[num].PFD.LE_pin, mux,
            icp, apbw, polarity, r, n);
            PFD_configure(&hara[num].PFD);
            GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_1, ~GPIO_PIN_1);
        break;

        //////////////////////
        //  check PFD Mux   //
        //////////////////////
        case CMD_PFD_MUX:
            mux_state = 0;
            mux_state |= (PFD_mux(&hara[1]) << 3);
            mux_state |= (PFD_mux(&hara[2]) << 4);
            mux_state |= (PFD_mux(&hara[3]) << 5);
        break;

        //////////////////////////
        //  send UART message   //
        //////////////////////////
        case CMD_UART_MSG:
            //store pfd mux values in state flag
            state |= mux_state;
            //downconversion chain message
            if (cmd & 0x01) {
                pkt[0] =  state;
                pkt[1] =  (ADC_data[0]      & 0xFF00) >> 8;
                //ADC0 - pot
                pkt[2] =  (ADC_data[0]      & 0x00FF);
                pkt[3] =  (hara[1].vadc     & 0xFF00) >> 8;
                //ADC1 - element 1
                pkt[4] =  (hara[1].vadc     & 0x00FF);
                pkt[5] =  (hara[2].vadc     & 0xFF00) >> 8;
                //ADC2 - element 2
                pkt[6] =  (hara[2].vadc     & 0x00FF);
                pkt[7] =  (hara[3].vadc     & 0xFF00) >> 8;
                //ADC3 - element 3
                pkt[8] =  (hara[3].vadc     & 0x00FF);
```

```
        pkt[9] =  (hara[1].theta    & 0xFF00) >> 8; //theta_1
        pkt[10] = (hara[1].theta    & 0x00FF);
        pkt[11] = (hara[2].theta    & 0xFF00) >> 8; //theta_2
        pkt[12] = (hara[2].theta    & 0x00FF);
        pkt[13] = (hara[3].theta    & 0xFF00) >> 8; //theta_3
        pkt[14] = (hara[3].theta    & 0x00FF);
        pkt[15] = (hara[1].psi      & 0xFF00) >> 8; //psi_1
        pkt[16] = (hara[1].psi      & 0x00FF);
        pkt[17] = (hara[2].psi      & 0xFF00) >> 8; //psi_2
        pkt[18] = (hara[2].psi      & 0x00FF);
        pkt[19] = (hara[3].psi      & 0xFF00) >> 8; //psi_3
        pkt[20] = (hara[3].psi      & 0x00FF);
    }
    //xi and switch state
    else if (cmd & 0x02) {
        pkt[0] =  state | 0x40;
        pkt[1] =  (ADC_data[0]      & 0xFF00) >> 8; //ADC0 - pot
        pkt[2] =  (ADC_data[0]      & 0x00FF);
        pkt[3] =  (hara[1].vadc     & 0xFF00) >> 8; //ADC1 - element 1
        pkt[4] =  (hara[1].vadc     & 0x00FF);
        pkt[5] =  (hara[2].vadc     & 0xFF00) >> 8; //ADC2 - element 2
        pkt[6] =  (hara[2].vadc     & 0x00FF);
        pkt[7] =  (hara[3].vadc     & 0xFF00) >> 8; //ADC3 - element 3
        pkt[8] =  (hara[3].vadc     & 0x00FF);
        pkt[9] =  (hara[1].sw       & 0xFF00) >> 8; //sw_1
        pkt[10] = (hara[1].sw       & 0x00FF);
        pkt[11] = (hara[2].sw       & 0xFF00) >> 8; //sw_2
        pkt[12] = (hara[2].sw       & 0x00FF);
        pkt[13] = (hara[3].sw       & 0xFF00) >> 8; //sw_3
        pkt[14] = (hara[3].sw       & 0x00FF);
        pkt[15] = (hara[1].xi       & 0xFF00) >> 8; //xi_1
        pkt[16] = (hara[1].xi       & 0x00FF);
        pkt[17] = (hara[2].xi       & 0xFF00) >> 8; //xi_2
        pkt[18] = (hara[2].xi       & 0x00FF);
        pkt[19] = (hara[3].xi       & 0xFF00) >> 8; //xi_3
        pkt[20] = (hara[3].xi       & 0x00FF);
    }
    //50th packet to update offsets
    else {
        pkt[0] =  state | 0x80;
```

```
                    pkt[1] =  (ADC_data[0]      & 0xFF00) >> 8; //ADC0 - pot
                    pkt[2] =  (ADC_data[0]      & 0x00FF);
                    pkt[3] =  (hara[1].vadc     & 0xFF00) >> 8; //ADC1 - element 1
                    pkt[4] =  (hara[1].vadc     & 0x00FF);
                    pkt[5] =  (hara[2].vadc     & 0xFF00) >> 8; //ADC2 - element 2
                    pkt[6] =  (hara[2].vadc     & 0x00FF);
                    pkt[7] =  (hara[3].vadc     & 0xFF00) >> 8; //ADC3 - element 3
                    pkt[8] =  (hara[3].vadc     & 0x00FF);
                    pkt[9] =  (hara[1].delta_i  & 0xFF00) >> 8; //deltai_1
                    pkt[10] = (hara[1].delta_i  & 0x00FF);
                    pkt[11] = (hara[2].delta_i  & 0xFF00) >> 8; //deltai_2
                    pkt[12] = (hara[2].delta_i  & 0x00FF);
                    pkt[13] = (hara[3].delta_i  & 0xFF00) >> 8; //deltai_3
                    pkt[14] = (hara[3].delta_i  & 0x00FF);
                    pkt[15] = (hara[1].delta_j  & 0xFF00) >> 8; //deltaj_1
                    pkt[16] = (hara[1].delta_j  & 0x00FF);
                    pkt[17] = (hara[2].delta_j  & 0xFF00) >> 8; //deltaj_2
                    pkt[18] = (hara[2].delta_j  & 0x00FF);
                    pkt[19] = (hara[3].delta_j  & 0xFF00) >> 8; //deltaj_3
                    pkt[20] = (hara[3].delta_j  & 0x00FF);
                }
                //send packet
                UARTsend(pkt, sizeof pkt);
                //remove flags from state
                state &= ~(0xF8);
            break;

            //other?
            default:

            break;

        }
        //if there's no other command..
        if (isEmpty(&cmd_fifo)) {
            //if we were retrodirecting, keep going:
            if (state == STATE_RETRODIRECT) {
                push(&cmd_fifo, CMD_SW_RETRODIRECT);
            }
            //else, if we were actively steering keep steering
            else if (state == CMD_SW_ACTIVE_STEER) {
                push(&cmd_fifo, cmd);
            }
        }
    }
    SysCtlDelay(100);
```

```
        }
}
```

A.4.2 HARABoosterPack.h

```
/*
 * HARABoosterPack.h
 *
 *    This is for the TM4C1294NCPDT microcontroller on the EK-TM4D1294XL wearing the HARA BoosterPack!
 */

#ifndef HARABOOSTERPACK_H_
#define HARABOOSTERPACK_H_

#include <stdint.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include "inc/hw_types.h"
#include "inc/hw_memmap.h"
#include "inc/hw_ints.h"
#include "driverlib/sysctl.h"
#include "driverlib/interrupt.h"
#include "driverlib/gpio.h"
#include "driverlib/ssi.h"
#include "driverlib/i2c.h"
#include "driverlib/adc.h"
#include "driverlib/timer.h"
#include "driverlib/pin_map.h"
#include "HARA_LUTs.h"
#include "FIFO.h"

#define DEBUG_MODE 0


//hardware definitions
#define LTC6946_MESSAGE_DELAY    5000    //number of cycles to delay between PLL messages
#define LTC6946_R_DIV           100      //Ref signal divider (just the ref freq in MHz)
#define    MCP4725_FULLSCALE        4096    //fullscale value for MCP_4725 DAC
#define ADC_FULLSCALE           4096    //fullscale value for ADC
//Vcal sweep definitions
#define SWEEP_RANGE_THRESH_H    3823    //2.8V
#define SWEEP_RANGE_THRESH_L    3004    //2.2V
#define VCAL_NUM_SWEEPS         3       //number of sweeps to perform to find Vcal
#define ADC_SAMPLE_COUNT        1       //number of samples to take before reading
#define VCAL_SWEEP_SETTLE_TIME  1000    //delay of 3000 instr ~=25uS
//Retrodirection definitions
#define BALANCE_THRESH_H        3004    //2.2 V
#define BALANCE_THRESH_L        2594    //1.9 V
#define BALANCE_MIDPOINT        2799    //2.05 V
#define PHASE_WRAP_TOP          4095
#define PHASE_WRAP_BOTTOM       1990
#define BALANCE_BOX_SIZE        2       //points to average around center
```

```
//generic definitions
#define BIT0    0x01
#define BIT1    0x02
#define BIT2    0x04
#define BIT3    0x08
#define BIT4    0x10
#define BIT5    0x20
#define BIT6    0x40
#define BIT7    0x80
//definitions to be categorized later:
#define PS_RX   0x00
#define PS_TX   0x01
#define SW_DAC  0x00
#define SW_CP   0x01

//structure to hold LTC_6946
typedef struct S_LTC_6946 {
    unsigned char    CS_pin;     //CS_n for SPI module

    unsigned short    R_div;      //reference divider (PFD = REF / R_div)
    unsigned short    N_div;      //divider for VCO (VCO = N_div * PFD
    unsigned char   O_div;      //output divider (RF = VCO / O_div)
    unsigned char   P_level;    //output power level (0, 1, 2, 3) -> (-9, -6, -3, 0)dBm single ended

    unsigned long   f_rf;               //calculated PLL output frequency, assumes 10 MHz reference signal
    unsigned short  config_msg[11];      //configuration message
    unsigned short  recal_msg;           //recalibration message
} LTC_6946;


//structure to hold DAC information
typedef struct S_MCP4725 {
    unsigned char    num;    //DAC number
    unsigned char    addr;    //I2C address
    unsigned short    val;    //current DAC value
} MCP_4725;


//structure to hold PFD information
typedef struct S_ADF4002 {
    unsigned short  LE_pin;         //LE pin for ADF4002 FD
    unsigned short  R_div;          //Reference divider
    unsigned short  N_div;          //RF divider
    unsigned char   CP_setting;     //CP current setting
    unsigned char   ABPW;           //Anti-Backlash-Pulse-Width setting
    unsigned char   CP_polarity;    //CP polarity bit
    unsigned char   mux;            //mux output select

    unsigned int   msg[4];         //4 24-bit messages

    unsigned char   mux_val;        //binary value of MUX output
} ADF_4002;


//structure to hold HARA element
```

```
typedef struct S_HARA {
    MCP_4725 DAC_rx;            //DAC for downconversion phase shifter
    MCP_4725 DAC_tx;            //DAC for transmission phase shifter
    ADF_4002 PFD;               //PFD for downconversion chain
    //physical system things
    unsigned short  rxDAC;      //DAC value (downconversion phase shifter)
    unsigned short  txDAC;      //DAC value (transmission phase shifter)
    unsigned short  vadc;       //ADC value (updates in ADC_ISR)
    unsigned short  lock_cnt;   //counter to determine if PFD sample needs to be resent
    unsigned short  number;     //identifier number
    unsigned short  sw;         //current state of phase shifter switch (1 = CP, 0 = DAC)
    //DLL-HPC system things
    signed short    psi;        //delta psi_n term: measured receive signal phase relative
                                //                    to the (n-1)th downconversion chain
    signed short    theta;      //theta_n term: downconversion phase delay control voltage
                                //                either controlled manually by DAC_rx or
                                //                automatically by the PFD/CP chip
    signed short    xi;         //xi_n term: carrier generation phase delay control voltage
                                //             always controlled manually by DAC_tx
    signed short    delta_i;    //delta_i_n term: physical system constant that encompasses
                                //                  all phase differences introduced by wiring,
                                //                  board layout, etc. in downconversion chain
                                //              note: this is actually -1*delta_i in the eqn.
                                //                    IF = theta - psi + delta_i, so it must be
                                //                  _added_ when driving theta manually and
                                //                  _subtracted_ when measuring psi in closed loop
    signed short    delta_j;    //delta_j_n term: physical system constant that encompasses
                                //                  all phase differences introduced by wiring,
                                //                  board layout, etc. in carrier gen. chain
} HARA;



void            LTC_init(LTC_6946 *temp, unsigned int CS_Pin, unsigned int R_div,
                         unsigned int N_div, unsigned int O_div, unsigned char P_level);
unsigned int    LTC_configure(LTC_6946 *pll);
```

```c
unsigned int    LTC_recal(LTC_6946 *pll);
unsigned int    DAC_init(MCP_4725 *dac, unsigned char num);
unsigned int    DAC_write(MCP_4725 *dac, unsigned int val);
void            PFD_init(ADF_4002 *pfd, unsigned char le_pin, unsigned char mux,
                         unsigned char icp, unsigned char apbw, unsigned char polarity,
                         unsigned short r, unsigned short n);
unsigned int    PFD_configure(ADF_4002 *pfd);
unsigned int    PFD_mux(HARA *hara);
void            HARA_init(HARA *hara, unsigned int num);
unsigned int    changeSW(HARA *hara, unsigned int sw);
unsigned int    writePhase(HARA *hara, unsigned char mode, signed int phase);
unsigned int    pointArray(signed int phi, unsigned char mode);
unsigned int    conjugate(HARA *hara);
void            HARABoosterPack_init(uint32_t ui32SysClock, uint32_t spi_dataRate, bool i2c_fastMode);
void            UARTsend(unsigned char buffer[], unsigned int len);







#endif /* HARABOOSTERPACK_H_ */
```

A.4.3 HARABoosterPack.c

```c
/*
 * HARABoosterPack.c
 *
 *  Created on: Apr 9, 2019
 *      Author: Michael Bolt
 */


#include <stdint.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include "inc/hw_types.h"
#include "inc/hw_memmap.h"
#include "inc/hw_ints.h"
#include "driverlib/sysctl.h"
#include "driverlib/interrupt.h"
#include "driverlib/gpio.h"
#include "driverlib/ssi.h"
#include "driverlib/i2c.h"
#include "driverlib/adc.h"
#include "driverlib/timer.h"
#include "driverlib/pin_map.h"
#include "driverlib/uart.h"
#include "HARABoosterPack.h"


//PLLs
LTC_6946 PLL1;
LTC_6946 PLL2;
//HARA elements
HARA hara[4];
extern FIFO cmd_fifo;


extern void ADC_ISR(void);
extern void Timer0B_ISR(void);
extern void Timer0A_ISR(void);
extern void PORTP_ISR(void);
extern void PORTH_ISR(void);
extern void PORTM_ISR(void);
extern void UART_ISR(void);

//global variables
//Timer timeout
extern unsigned int timeoutCounter;     //counter for timeouts


//*****************************************************************************
//
//! Initializes LTC_6946 structure given a few parameters
//!
//! \param temp is a pointer to the LTC_6946 structure to initialize
```

```
//! \param CS_Pin specifies which CS pin for the SPI interface to use (1, 2, 3)
//! \param R_div is the unsigned integer reference divider value for the PLL,
//! such that f_pfd = 10 MHz / R
//! \param N_div is the unsigned integer VCO divider value for the PLL, such
//! f_vco = N * f_pfd
//! \param O_div is the unsigned integer output divider for the PLL, such that
//! f_out = f_vco / O
//! \param P_level is the output power level setting (0 through 3)
//!
//! This function configures the PLL structure given to it with the information
//! provided, configuring the config_msg and recal_msg
//!
//! \return None.
//
//*****************************************************************************
void LTC_init(LTC_6946 *temp, unsigned int CS_Pin, unsigned int R_div,
              unsigned int N_div, unsigned int O_div, unsigned char P_level) {
    temp->CS_pin = CS_Pin;
    temp->R_div = R_div;
    temp->N_div = N_div;
    temp->O_div = O_div;
    temp->P_level = P_level;
    temp->f_rf = (10e6 / R_div) * N_div / O_div;
    //config_msg
    temp->config_msg[0] = (0x0100 << 1) | (0x00);
    temp->config_msg[1] = (0x0200 << 1) | (0x0A);
    temp->config_msg[2] = (0x0300 << 1) | (0x10 | ((R_div & 0x0300) >> 8));
    temp->config_msg[3] = (0x0400 << 1) | (R_div & 0x00FF);
    temp->config_msg[4] = (0x0500 << 1) | ((N_div & 0xFF00) >> 8);
    temp->config_msg[5] = (0x0600 << 1) | (N_div & 0x00FF);
    temp->config_msg[6] = (0x0700 << 1) | (0xF3);
    temp->config_msg[7] = (0x0800 << 1) | (0xE0 | ((P_level & 0x03) << 3) | (O_div & 0x07));
    temp->config_msg[8] = (0x0900 << 1) | (0x2B);
    temp->config_msg[9] = (0x0A00 << 1) | (0x00);
    temp->config_msg[10] = (0x0200 << 1) | (0x08);
    //recal_msg
    temp->recal_msg = (0x0700 << 1) | (0xF3);
}



//*****************************************************************************
//
//! Sends configuration messages to the specified PLL
//!
//! \param pll is the pointer to the LTC_6946 structure you would like to
//! configure
//!
```

```c
//! This function sends the config_msg for a chosen PLL, adjusting the settings
//!
//! \return timeoutCounter: if greater than 0, the SPI transmission did not
//! timeout
//
//*****************************************************************************
unsigned int LTC_configure(LTC_6946 *pll) {
    uint32_t port_base,pin;
    unsigned int i;
    //determine which pin to use
    switch(pll->CS_pin) {
        case 1: //CS_1 -> H.2
            port_base = GPIO_PORTH_BASE;
            pin = GPIO_PIN_2;
            break;
        case 2: //CS_2 -> H.3
            port_base = GPIO_PORTH_BASE;
            pin = GPIO_PIN_3;
            break;
        case 3: //CS_3 -> G.0
            port_base = GPIO_PORTG_BASE;
            pin = GPIO_PIN_0;
            break;
        case 4: //CS_4 -> N.2
            port_base = GPIO_PORTN_BASE;
            pin = GPIO_PIN_2;
            break;
        case 5: //CS_5 -> N.3
            port_base = GPIO_PORTN_BASE;
            pin = GPIO_PIN_3;
            break;
        default:    //if no valid CS pin is used, just return 0
            return 0;
    }
    //enable Timeout after 50 ms
    timeoutCounter = 5;
    TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);     //clear TimerA interrupt flags
    TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);    //enable Timer0A interrupt
    //write messages
    for(i=0; i<11; i++) {
        if (timeoutCounter) {                                   //check no timeout has occurred
            GPIOPinWrite(port_base, pin, 0);                    //CS LOW
            SSIDataPut(SSI2_BASE, (pll->config_msg[i] & 0xFF00) >> 8);      //write MSB
                while(SSIBusy(SSI2_BASE) && timeoutCounter);    //wait for transmission or timeout
            SSIDataPut(SSI2_BASE, (pll->config_msg[i] & 0x00FF) >> 0);      //write LSB
                while(SSIBusy(SSI2_BASE) && timeoutCounter);    //wait for transmission or timeout
```

```
            GPIOPinWrite(port_base, pin, pin); //CS_n HIGH
            SysCtlDelay(LTC6946_MESSAGE_DELAY); //delay between messages
        }
    }
    //disable TimeoutTimer (Timer0A)
    TimerIntDisable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
    TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
    //return timeoutCounter
    return timeoutCounter;
}



//*****************************************************************************
//
//! Sends recalibration message to the specified PLL
//!
//! \param pll is the pointer to the LTC_6946 structure you would like to
//! configure
//!
//! This function sends the recal_msg for a chosen PLL, recalibrating the VCO
//!
//! \return timeoutCounter: if greater than 0, the SPI transmission did not
//! timeout
//
//*****************************************************************************
unsigned int LTC_recal(LTC_6946 *pll) {
    uint32_t port_base,pin;
    //determine which pin to use
    switch(pll->CS_pin) {
        case 1: //CS_1 -> H.2
            port_base = GPIO_PORTH_BASE;
            pin = GPIO_PIN_2;
            break;
        case 2: //CS_2 -> H.3
            port_base = GPIO_PORTH_BASE;
            pin = GPIO_PIN_3;
            break;
        case 3: //CS_3 -> G.0
            port_base = GPIO_PORTG_BASE;
            pin = GPIO_PIN_0;
            break;
        case 4: //CS_4 -> N.2
            port_base = GPIO_PORTN_BASE;
            pin = GPIO_PIN_2;
            break;
        case 5: //CS_5 -> N.3
            port_base = GPIO_PORTN_BASE;
            pin = GPIO_PIN_3;
            break;
        default:    //if no valid CS pin is used, just return 0
            return 0;
```

```
    }
    //enable Timeout after 50 ms
    timeoutCounter = 5;
    TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);     //clear TimerA interrupt flags
    TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);    //enable Timer0A interrupt
    //write recal message
    GPIOPinWrite(port_base, pin, 0);                        //CS LOW
    SSIDataPut(SSI2_BASE, (pll->recal_msg & 0xFF00) >> 8);  //write MSB
    while(SSIBusy(SSI2_BASE) && timeoutCounter);            //wait for transmission or timeout
    SSIDataPut(SSI2_BASE, (pll->recal_msg & 0x00FF) >> 0);  //write LSB
    while(SSIBusy(SSI2_BASE) && timeoutCounter);            //wait for transmission or timeout
    GPIOPinWrite(port_base, pin, pin);                      //CS_n HIGH
    SysCtlDelay(LTC6946_MESSAGE_DELAY);                     //delay between messages
    //disable TimeoutTimer (Timer0A)
    TimerIntDisable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
    TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
    //return timeoutCounter
    return timeoutCounter;
}


//*****************************************************************************
//
//! Initializes MCP_4725 structure given a few parameters
//!
//! \param dac is a pointer to the MCP_4725 structure to initialize
//! \param num is the identifier of which DAC you are initializing; this
//! determines the last 3 bits of the I2C address
//!
//! This function configures the DAC structure given to it with the information
//! provided, setting the i2c address and current value; this function also
//! attempts to write a half-scale value to the DAC and returns a 1 on success
//! and 0 on failure
//!
//! \return timeoutCounter: if greater than 0, the I2C transmission did not
//! timeout
//!
//! \return None.
//
//*****************************************************************************
unsigned int DAC_init(MCP_4725 *dac, unsigned char num) {
    dac->num = num;
    dac->addr = 0x60 | num;
    dac->val = MCP4725_FULLSCALE >> 1;
    return DAC_write(dac, dac->val);
}


```

```c

//*****************************************************************************
//
//! Writes new value value to the output of the chosen DAC
//!
//! \param dac is the pointer to the DAC you would like to write to
//! \param val is the 12-bit unsigned integer value to write to the DAC
//!
//! This function writes a new value to the specified DAC
//!
//! \return timeoutCounter: if greater than 0, the I2C transmission did not
//! timeout
//
//*****************************************************************************
unsigned int DAC_write(MCP_4725 *dac, unsigned int val) {
    //check val for limits
    if (val >= 0x8000) {
        val = 0;
    }
    else if (val >= MCP4725_FULLSCALE) {
        val = MCP4725_FULLSCALE - 1;
    }

    //construct message
    unsigned char msg[3];
    msg[0] = 0x40;              //regular write, no power-down mode
    msg[1] = (val & 0x0FF0) >> 4;   //D[11:4]
    msg[2] = (val & 0x000F) << 4;   //D[3:0]

    //enable timeout after 10 ms
    timeoutCounter = 1;
    TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);     //clear TimerA interrupt flags
    TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);    //enable Timer0A interrupt
    //send message
    I2CMasterSlaveAddrSet(I2C0_BASE, dac->addr, false);
    //byte 0
    if(timeoutCounter) {
        I2CMasterDataPut(I2C0_BASE, msg[0]);
        I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_START);
            while(!I2CMasterBusy(I2C0_BASE) && timeoutCounter);
            while(I2CMasterBusy(I2C0_BASE)  && timeoutCounter);
        //byte 1
        if(timeoutCounter) {
            I2CMasterDataPut(I2C0_BASE, msg[1]);
            I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_CONT);
                while(!I2CMasterBusy(I2C0_BASE) && timeoutCounter);
                while(I2CMasterBusy(I2C0_BASE)  && timeoutCounter);
            //byte 2
            if(timeoutCounter) {
                I2CMasterDataPut(I2C0_BASE, msg[2]);
                I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_FINISH);
                    while(!I2CMasterBusy(I2C0_BASE) && timeoutCounter);
```

```
                        while(I2CMasterBusy(I2C0_BASE)  && timeoutCounter);
                    //update DAC value after message send successfully
                    if(timeoutCounter) {
                        dac->val = val;
                    }
                }
            }
        }
    //disable TimeoutTimer (Timer0A)
    TimerIntDisable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
    TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);

    //return timeoutCounter
    return timeoutCounter;
}



//*****************************************************************************
//
//! Initializes ADF_4002 structure given a few parameters
//!
//! \param pfd is a pointer to the ADF_4002 structure to initialize
//! \param le_pin is which CS pin to use for the SPI interface
//!
//! This function configures the PFD structure to some initial values determined
//! here. Refer to the data sheet to know what to change them to!
//!
//! \return None.
//
//*****************************************************************************
void PFD_init(ADF_4002 *pfd, unsigned char le_pin, unsigned char mux, unsigned char icp, unsigned char apbw, unsigned char polarity, unsigned short r, unsigned short n) {
    //configure PFD structure
    pfd->LE_pin = le_pin;
    pfd->R_div = r;                 //set to 1 for true PFD
    pfd->N_div = n;                 //set to 1 for true PFD
    pfd->CP_setting = icp;          //[0:7], 7 = 5 mA
    pfd->ABPW = apbw;               //1 = 2.9ns, 2 = 6.0ns
    pfd->CP_polarity = polarity;    //polarity of CP controls
    pfd->mux = mux;                 //mux output select

    //Alternate method (Counter Reset Method)
    //Function Latch - F1 = 1
    pfd->msg[0] = ((pfd->CP_setting & 0x07) << 18)
                | ((pfd->CP_setting & 0x07) << 15)
                | ((pfd->CP_polarity & 0x01) << 7)
                | ((pfd->mux & 0x07) << 4)
                | ((0x01) << 2)
                | (0x02);
    //R counter Latch
    pfd->msg[1] = (0x01 << 20)
                | ((pfd->ABPW & 0x03) << 16)
```

```c
                  | ((pfd->R_div & 0x3FFF) << 2)
                  | (0x00);
    //N counter Latch
    pfd->msg[2] = ((pfd->N_div & 0x1FFF) << 8)
                  | (0x01);
    //Function Latch - F1 = 0
    pfd->msg[3] = ((pfd->CP_setting & 0x07) << 18)
                  | ((pfd->CP_setting & 0x07) << 15)
                  | ((pfd->CP_polarity & 0x01) << 7)
                  | ((pfd->mux & 0x07) << 4)
                  | ((0x00) << 2)
                  | (0x02);
}


//****************************************************************************
//
//! Sends configuration messages to the specified PFD
//!
//! \param pfd is the pointer to the ADF_4002 structure you would like to
//! configure
//!
//! This function sends the config_msg for a chosen PFD, adjusting the settings
//!
//! \return timeoutCounter: if greater than 0, the SPI transmission did not
//! timeout
//
//****************************************************************************
unsigned int PFD_configure(ADF_4002 *pfd) {
    uint32_t port_base,pin;
    unsigned int i;

    //determine which pin to use
    switch(pfd->LE_pin) {
        case 1: //CS_1 -> H.2
            port_base = GPIO_PORTH_BASE;
            pin = GPIO_PIN_2;
            break;
        case 2: //CS_2 -> H.3
            port_base = GPIO_PORTH_BASE;
            pin = GPIO_PIN_3;
            break;
        case 3: //CS_3 -> G.0
            port_base = GPIO_PORTG_BASE;
            pin = GPIO_PIN_0;
            break;
        case 4: //CS_4 -> N.2
            port_base = GPIO_PORTN_BASE;
            pin = GPIO_PIN_2;
            break;
        case 5: //CS_5 -> N.3
            port_base = GPIO_PORTN_BASE;
            pin = GPIO_PIN_3;
            break;
```

```c
            default:    //if no valid CS pin is used, just return 0
                return 0;
        }

        //enable Timeout after 50 ms
        timeoutCounter = 5;
        TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);     //clear TimerA interrupt flags
        TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);    //enable Timer0A interrupt

        //write messages
        for(i=0; i<4; i++) {
            if (timeoutCounter) {                           //check no timeout has occurred
                GPIOPinWrite(port_base, pin, 0);                        //CS LOW
                SSIDataPut(SSI2_BASE, (pfd->msg[i] & 0x00FF0000) >> 16);//write MSB
                    while(SSIBusy(SSI2_BASE) && timeoutCounter);        //wait for transmission/timeout
                SSIDataPut(SSI2_BASE, (pfd->msg[i] & 0x0000FF00) >> 8); //write mSB
                    while(SSIBusy(SSI2_BASE) && timeoutCounter);        //wait for transmission/timeout
                SSIDataPut(SSI2_BASE, (pfd->msg[i] & 0x000000FF) >> 0); //write LSB
                    while(SSIBusy(SSI2_BASE) && timeoutCounter);        //wait for transmission/timeout
                GPIOPinWrite(port_base, pin, pin);                      //CS_n HIGH
                SysCtlDelay(LTC6946_MESSAGE_DELAY);                     //delay between messages
            }
        }

        //disable TimeoutTimer (Timer0A)
        TimerIntDisable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
        TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);

        //return timeoutCounter
        return timeoutCounter;
}


//*****************************************************************************
//
//! Read PFD mux value and resend PFD config message if it's been 1 second
//!
//! \param HARA is the HARA element to perform this on
//!
//! This function reads the appropriate pin of port L to determine the chosen
//! HARA element's PFD mux value and determines if the config message needs to
```

```
//! be resent; this is only done if 10 consecutive samples show no lock (mux low)
//!
//! \return value of the PFD mux
//
//***************************************************************************
unsigned int PFD_mux(HARA *hara) {
    //determine which pin to read from Port L
    uint32_t pin;
    switch(hara->number) {
    case 1:
        pin = GPIO_PIN_1;
        break;
    case 2:
        pin = GPIO_PIN_2;
        break;
    case 3:
        pin = GPIO_PIN_3;
        break;
    default:
        return 0;
    }
    //read in mux value
    if (GPIOPinRead(GPIO_PORTL_BASE, pin)) {
        hara->PFD.mux_val = 1;
        hara->lock_cnt = 10;
    }
    else {
        hara->PFD.mux_val = 0;
        hara->lock_cnt--;
    }
    //if there's been a second of no lock, resend the config message
    if (!hara->lock_cnt) {
        pin =    (hara->number << 9)
                | (hara->PFD.mux << 6)
                | (hara->PFD.CP_setting << 3)
                | (hara->PFD.ABPW << 1)
                | (hara->PFD.CP_polarity)
                | CMD_SW_CHANGE_PFD_MUX_OUT;
        push(&cmd_fifo, pin);
        push(&cmd_fifo, hara->PFD.R_div);
        push(&cmd_fifo, hara->PFD.N_div);
        hara->lock_cnt = 10;
    }
    //return mux value
    return hara->PFD.mux_val;
}


//***************************************************************************
//
//! Initializes HARA element structure given the element number
//!
//! \param num is the identifier of which HARA element to initialize (0-3)
//!
```

```c
//! This function initializes a HARA element structure by configuring the two
//! DACs and initializing all internal values to 0
//!
//! \return none
//
//******************************************************************************
void HARA_init(HARA *hara, unsigned int num) {
    //initiailze sub-strucs
    switch(num) {
        default:
        case 0:
            DAC_init(&hara->DAC_rx, 10);
            DAC_init(&hara->DAC_tx, 10);
            PFD_init(&hara->PFD, 10, 0, 0, 2, 1, 1, 1);
            break;
        case 1:
            DAC_init(&hara->DAC_rx, 3);
            DAC_init(&hara->DAC_tx, 2);
            PFD_init(&hara->PFD, 3, 0, 0, 2, 1, 1, 1);
            break;
        case 2:
            DAC_init(&hara->DAC_rx, 1);
            DAC_init(&hara->DAC_tx, 0);
            PFD_init(&hara->PFD, 4, 0, 0, 2, 1, 1, 1);
            break;
        case 3:
            DAC_init(&hara->DAC_rx, 5);
            DAC_init(&hara->DAC_tx, 4);
            PFD_init(&hara->PFD, 5, 0, 0, 2, 1, 1, 1);
            break;
    }
    //initialize hardware things, default to CP controlled theta
    hara->rxDAC = hara->DAC_rx.val;
    hara->txDAC = hara->DAC_tx.val;
    hara->lock_cnt = 0;
    hara->number = num;
    changeSW(hara, SW_CP);
    //DLL-HPC system initialization
    hara->psi = 0;
    hara->theta = 0;
    hara->xi = 0;
    hara->delta_i = 0;
    hara->delta_j = 0;
    //vadc is updated continuously in the ADC ISR, so it is untouched here
}



//******************************************************************************
//
//! Changes the phase shifter control signal for the chosen HARA element
//!
//! \param HARA is the HARA element to change the SW of
```

```
//! \param sw determines which contorl signal to use and should be set to either
//!        SW_CP or SW_DAC
//!
//! This function changes the SW for the specified HARA element
//!
//! \return SW_CP or SW_DAC depending on what was written
//
//*****************************************************************************
unsigned int changeSW(HARA *hara, unsigned int sw) {
    //determine the pin number
    uint32_t pin = GPIO_PIN_0 << (hara->number);
    //if setting to CP
    if(sw) {
        GPIOPinWrite(GPIO_PORTF_BASE, pin, pin);
        hara->sw = SW_CP;
        return SW_CP;
    }
    //else, set to DAC
    else {
        GPIOPinWrite(GPIO_PORTF_BASE, pin, 0);
        hara->sw = SW_DAC;
        return SW_DAC;
    }
}



//*****************************************************************************
//
//! Writes a new phase value to the chosen HARA element
//!
//! \param HARA is the HARA element to change the phase of
//! \param mode determines to write to the tx/rx phase shifter (PS_TX/PS_RX)
//! \param phase is the unsigned integer phase to write to the HARA element
//!
//! This function writes a new phase to the specified HARA element
//!
//! \return 1 if write is successful, 0 if not
//
//*****************************************************************************
unsigned int writePhase(HARA *hara, unsigned char mode, signed int phase) {
    //if writing to TX phase shifter (xi)
    if (mode) {
        //remove delta_j term
        phase -= hara->delta_j;
        //phase wrap if necessary
        while (phase > 359) phase -= 360;
        while (phase < 0)   phase += 360;
        //get DAC value to write from txLUT
        uint16_t dac_val = txLUT(phase, hara->number);
        //make 10 attempts to write to DAC_tx
        int i = 10;
```

```
        for(i = 10; i != 0; i--) {
            if (DAC_write(&hara->DAC_tx, dac_val)) {
                //add back delta_j to get the desired phase
                phase += hara->delta_j;
                //rewrap
                while (phase > 359) phase -= 360;
                while (phase < 0)   phase += 360;
                //store actual output phase difference to Xi
                hara->xi = phase;
                hara->txDAC = dac_val;
                return 1;
            }
        }
    }
    //else. we're writing to the RX phase shifter (theta)
    else {
        //ensure the phase shifter is being controlled by the DAC
        if(hara->sw)
            changeSW(hara, SW_DAC);
        //add in the negative delta_i term to compensate
        phase += hara->delta_i;
        //phase wrap as necessary
        while (phase > 359) phase -= 360;
        while (phase < 0)   phase += 360;
        //get DAC value to write to DAC_rx
        uint16_t dac_val = rxLUT(phase, hara->number);
        //make 10 attempts to write to DAC_rx
        int i = 10;
        for(i = 10; i != 0; i--) {
            if (DAC_write(&hara->DAC_rx, dac_val)) {
                //subtract back our the negative delta_i term to get desired phase
                phase -= hara-> delta_i;
                //rewrap as necessary
                while (phase > 359) phase -= 360;
                while (phase < 0)   phase += 360;
                //store actual output pahse difference to theta
                hara->theta = phase;
                hara->rxDAC = dac_val;
                return 1;
            }
        }
    }
    //if we made it here, we failed writing to a DAC
    return 0;
}



//****************************************************************************
//
//! Points the HARA array at direction phi
//!
//! \param phi is the direction to point the array in (-90 to 90)
//! \param mode determines to write to the tx/rx phase shifters (PS_TX/PS_RX)
```

```
//!
//! This function points the beam of the HARA system by applying a linear
//! phase shift (alpha) between each element based on the direction you would
//! like to point in (phi). Phi is defined as broadside being 0 degrees.
//!
//! \return 1 if writes are successful, 0 if not
//
//***************************************************************************
unsigned int pointArray(signed int phi, unsigned char mode) {
    signed int e;
    signed int alpha;
    //check valid values
    if (phi < -90) phi = -90;
    if (phi > 90)  phi = 90;
    //determine phase difference between elements
    if (phi < 0) {
        e = -1 * phi;
        alpha = activeSteer_LUT(e);
        alpha = alpha * -1;
    }
    else alpha = activeSteer_LUT(phi);
    //write new phases
    int n = 3;
    for (n = 3; n != 0; n--) {
        writePhase(&hara[n], mode, (alpha * n));
        //TODO: or maybe this should be just alpha, since it's relative to the (n-1)th element?
    }
    return e;
}




//***************************************************************************
//
//! Performs phase conjugation for the chosen HARA element
//!
//! \param hara is the HARA element to conjugate at
//! \param hara_o is the HARA element to conjugate relative to
//!
//! This function performs phase conjugation, assuming that rxOffset for the
//! chosen HARA element was recorded at steady state lock with theta_rx equal
//! to psi_n, all of which should be equal to 0.
//! 1. the RX phase shifter control voltage (CP) is sampled
//! 2. the ADC value is converted to a phase
//! 3. the total phase shift from 0 (rxTheta) is found as the difference
//! 4. rxTheta is conjugated and applied to the transmision PLL
//!
//! \return 1 is write to theta was successful, else 0
//
```

```c
//*****************************************************************************
unsigned int conjugate(HARA *hara) {
    //1. sample ADC
    //2. convert ADC to phase
    //3. calculate rx Theta
    //(these are all done in the ADC ISR)
    //4. conjugate phase
    signed int temp = -1 * hara->psi;
    return writePhase(hara, PS_TX, temp);
}



//*****************************************************************************
//
//! configure the HARA BoosterPack interfaces
//!
//! \param ui32SysClock is the system clock frequency returned by the
//! SysCtlClockFreqSet() instruction
//! \param spi_dataRate is the desired data-rate for the SPI module
//! \param i2c_fastMode determines whether or not the I2C module is configured
//! in Fast-Mode (400 kHz) or standard mode (200 kHz)
//!
//! This function initializes all pins and modules internal to the TM4C for the
//! HARA backpack
//!
//! \return None.
//
//*****************************************************************************
void HARABoosterPack_init(uint32_t ui32SysClock, uint32_t spi_dataRate, bool i2c_fastMode) {
    //GPIO Port Enables
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
        while(!SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOA));
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
        while(!SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOB));
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
        while(!SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOD));
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
        while(!SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOE));
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
        while(!SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOF));
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOG);
        while(!SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOG));
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOH);
        while(!SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOH));
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOL);
        while(!SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOL));
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPION);
        while(!SysCtlPeripheralReady(SYSCTL_PERIPH_GPION));

    /////////////////
```

```
//  GPIO Pins   //
//////////////////
//  PFD MUXes   //
//  PL.1 - MUX1 //
//  PL.2 - MUX2 //
//  PL.3 - MUX3 //
//////////////////
//  DAC/CP SW   //
//  PF.1 - SW1  //
//  PF.2 - SW2  //
//  PF.3 - SW3  //
//////////////////
//GPIO inputs for monitoring PFD Muxes
GPIOPinTypeGPIOInput(GPIO_PORTL_BASE, (GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3));
//GPIO open-drain outputs for DAC/CP switches - default HIGH
GPIOPinTypeGPIOOutputOD(GPIO_PORTF_BASE, (GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3));
GPIOPadConfigSet(GPIO_PORTF_BASE, (GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3), GPIO_STRENGTH_8MA, GPIO_PIN_TYPE_OD);
GPIOPinWrite(GPIO_PORTF_BASE, (GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3), (GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3));



//////////////////////////////////
//              SPI Setup             //
//////////////////////////////////
//    PD.0 - MISO                    //
//    PD.1 - MOSI                    //
//    PD.3 - SCLK                    //
//    PH.2 - CS1                    //
//  PH.3 - CS2                  //
//  PG.0 - CS3                  //
//    PN.2 - CS4                    //
//    PN.3 - CS5                    //
//////////////////////////////////
//enable SSI functionality on PD.0,1,3
GPIOPinTypeSSI(GPIO_PORTD_BASE, GPIO_PIN_0 | GPIO_PIN_1 | GPIO_PIN_3);
GPIOPinConfigure(GPIO_PD0_SSI2XDAT1);
GPIOPinConfigure(GPIO_PD1_SSI2XDAT0);
GPIOPinConfigure(GPIO_PD3_SSI2CLK);
//PH.2,3 - CS_1, CS_2 (start HIGH)
GPIOPinTypeGPIOOutput(GPIO_PORTH_BASE, (GPIO_PIN_2 | GPIO_PIN_3));
GPIOPadConfigSet(GPIO_PORTH_BASE, (GPIO_PIN_2 | GPIO_PIN_3), GPIO_STRENGTH_12MA, GPIO_PIN_TYPE_STD);
GPIOPinWrite(GPIO_PORTH_BASE, (GPIO_PIN_2 | GPIO_PIN_3), (GPIO_PIN_2 | GPIO_PIN_3));
//PG.0 - CS_3 (start HIGH)
GPIOPinTypeGPIOOutput(GPIO_PORTG_BASE, GPIO_PIN_0);
GPIOPadConfigSet(GPIO_PORTG_BASE, GPIO_PIN_0, GPIO_STRENGTH_12MA, GPIO_PIN_TYPE_STD);
GPIOPinWrite(GPIO_PORTG_BASE, GPIO_PIN_0, GPIO_PIN_0);
//PN.2,3 - CS_4, CS_5 (start HIGH)
GPIOPinTypeGPIOOutput(GPIO_PORTN_BASE, (GPIO_PIN_2 | GPIO_PIN_3));
GPIOPadConfigSet(GPIO_PORTN_BASE, (GPIO_PIN_2 | GPIO_PIN_3), GPIO_STRENGTH_12MA, GPIO_PIN_TYPE_STD);
```

```
    GPIOPinWrite(GPIO_PORTN_BASE, (GPIO_PIN_2 | GPIO_PIN_3), (GPIO_PIN_2 |
    GPIO_PIN_3));
    //enable SSI module and wait for it to be ready
    SysCtlPeripheralEnable(SYSCTL_PERIPH_SSI2);
        while(!SysCtlPeripheralReady(SYSCTL_PERIPH_SSI2));
    SSIConfigSetExpClk(SSI2_BASE, ui32SysClock, SSI_FRF_MOTO_MODE_0,
    SSI_MODE_MASTER, spi_dataRate, 8);
    //enable the SSI module
    SSIEnable(SSI2_BASE);

    //////////////////////////
    //        I2C Setup        //
    //////////////////////////
    //    PB.2 = SCL (I2C0)    //
    //    PB.3 = SDA (I2C0)    //
    //////////////////////////
    //configure pins
    GPIOPinTypeI2CSCL(GPIO_PORTB_BASE, GPIO_PIN_2);
    GPIOPinTypeI2C(GPIO_PORTB_BASE, GPIO_PIN_3);
    GPIOPinConfigure(GPIO_PB2_I2C0SCL);
    GPIOPinConfigure(GPIO_PB3_I2C0SDA);
    //enable I2C peripheral
    SysCtlPeripheralEnable(SYSCTL_PERIPH_I2C0);
    SysCtlPeripheralReset(SYSCTL_PERIPH_I2C0);
        while(!SysCtlPeripheralReady(SYSCTL_PERIPH_I2C0));
    I2CMasterInitExpClk(I2C0_BASE, ui32SysClock, i2c_fastMode);    //set
    to true for FAST mode

    /////////////////
    //    ADC Setup    //
    /////////////////
    //    PE.3 - CH_0    - A0
    //    PE.4 - CH_9 - A1
    //    PE.5 - CH_8 - A2
    //    PE.0 - CH_3    - A3
    /////////////////
    //configure the 4 analog inputs
    GPIOPinTypeADC(GPIO_PORTE_BASE, (GPIO_PIN_0 | GPIO_PIN_3 | GPIO_PIN_4
    | GPIO_PIN_5));
    //enable ADC module
    SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
        while(!SysCtlPeripheralReady(SYSCTL_PERIPH_ADC0));
    //configure ADC clock as 120 MHz / 6 for 20 MHz ADC clock
    ADCClockConfigSet(ADC0_BASE, ADC_CLOCK_SRC_PLL | ADC_CLOCK_RATE_FULL,
    6);
    //hardware oversample rate of 64
    ADCHardwareOversampleConfigure(ADC0_BASE, 64);
    //use internal 3V reference
    ADCReferenceSet(ADC0_BASE, ADC_REF_INT);
    //configure sequencer 0 to go through A[0:3] on Timer Interrupt
    // A0, A9, A8, A3
    ADCSequenceConfigure(ADC0_BASE, 0, ADC_TRIGGER_TIMER, 0);
    ADCSequenceStepConfigure(ADC0_BASE, 0, 0, (ADC_CTL_CH0));
    ADCSequenceStepConfigure(ADC0_BASE, 0, 1, (ADC_CTL_CH9));
    ADCSequenceStepConfigure(ADC0_BASE, 0, 2, (ADC_CTL_CH8));
    ADCSequenceStepConfigure(ADC0_BASE, 0, 3, (ADC_CTL_CH3 | ADC_CTL_IE |
    ADC_CTL_END));
```

```
    ADCIntClear(ADC0_BASE, 0);                //clear sequence 0 interrupt
    ADCIntRegister(ADC0_BASE, 0, ADC_ISR);    //register ISR
    ADCIntEnable(ADC0_BASE, 0);                //enable sequence 0 interrupt
    ADCSequenceEnable(ADC0_BASE, 0);        //enable sequence 0

    //////////////////////////////////////////
    //                 Timer Setup            //
    //////////////////////////////////////////
    //  Timer0.A - 100 Hz ADC Sample Rate   //
    //  Timer0.B - 10 Hz USB message rate   //
    //////////////////////////////////////////
    SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
        while(!SysCtlPeripheralReady(SYSCTL_PERIPH_TIMER0));
    TimerConfigure(TIMER0_BASE, (TIMER_CFG_SPLIT_PAIR |     //split pair of timers (A and B)
                                 TIMER_CFG_A_PERIODIC |     //timerA is downcounting repeatedly
                                 TIMER_CFG_B_PERIODIC       //timerB is downcounting repeatedly
                                ));
    //TimerA
    TimerClockSourceSet(TIMER0_BASE, TIMER_CLOCK_SYSTEM);    //use system clock (120 MHz PLL)
    TimerPrescaleSet(TIMER0_BASE, TIMER_A, 200);            //pre-scale by 200 (600 kHz)
    TimerLoadSet(TIMER0_BASE, TIMER_A, 6000);                //set value timerA counts to (0.01s period)
    TimerControlStall(TIMER0_BASE, TIMER_A, true);            //stall TimerA if debugger pauses
        TimerControlTrigger(TIMER0_BASE, TIMER_A, true);        //enable TIMER_A to trigger ADC conversion
    TimerIntRegister(TIMER0_BASE, TIMER_A, Timer0A_ISR);    //register TimerA ISR (for timeouts)
    //TimerB
    TimerPrescaleSet(TIMER0_BASE, TIMER_B, 200);            //pre-scale by 200 (600 kHz)
    TimerLoadSet(TIMER0_BASE, TIMER_B, 60000);              //set TimerB count value (0.1s period)
    TimerControlStall(TIMER0_BASE, TIMER_B, true);          //stall TimerB if debugger pauses
//        TimerControlTrigger(TIMER0_BASE, TIMER_B, true);    //enable TIMER_B to trigger ADC conversion
    TimerIntClear(TIMER0_BASE, TIMER_TIMB_TIMEOUT);         //clear TimerB interrupt flag
    TimerIntRegister(TIMER0_BASE, TIMER_B, Timer0B_ISR);    //register Timer0B ISR
    TimerIntEnable(TIMER0_BASE, TIMER_TIMB_TIMEOUT);        //enable TimerB interrupt for timeout
    //enable TimerA and TimerB
    TimerEnable(TIMER0_BASE, TIMER_BOTH);

    //////////////////
    //  UART Setup  //
    //////////////////
    //  PA.0 - RX   //
    //  PA.1 - TX   //
```

```
    //  debug UART  //
    //////////////////
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
        while(!SysCtlPeripheralReady(SYSCTL_PERIPH_UART0));
    //configure A[0:1] as UART pins
    GPIOPinTypeUART(GPIO_PORTA_BASE, (GPIO_PIN_0 | GPIO_PIN_1));
    GPIOPinConfigure(GPIO_PA0_U0RX);
    GPIOPinConfigure(GPIO_PA1_U0TX);
    //configure UART for 115.200 kBaud, 8-N-A operation
    //this function call also enables the UART interface
    UARTConfigSetExpClk(UART0_BASE, ui32SysClock, 115200, (
                        UART_CONFIG_WLEN_8 |     //word length = 8
                        UART_CONFIG_STOP_ONE |   //stop bit length = 1
                        UART_CONFIG_PAR_NONE     //parity type = none
                        ));
    //register UART_ISR to the receive and receive timeout interrupts
    UARTIntClear(UART0_BASE, (UART_INT_RX | UART_INT_RT));
    UARTIntRegister(UART0_BASE, UART_ISR);
    UARTIntEnable(UART0_BASE, (UART_INT_RX | UART_INT_RT));

    //initialize the PLL structures to default values
    LTC_init(&PLL1,1,LTC6946_R_DIV,5801,1,3);    //PLL1 -> 5.801 GHz
    LTC_init(&PLL2,2,LTC6946_R_DIV,5900,1,3);    //PLL2 -> 5.900 GHz

    //initialize the HARA structures to default values
    int i = 0;
    for (i = 0; i < 4; i++) {
        HARA_init(&hara[i], i);
    }
}




void UARTsend(unsigned char buffer[], unsigned int len) {
    int i = 0;
    while(i < len) {
        UARTCharPut(UART0_BASE, buffer[i]);
        i++;
    }
}
```

A.4.4 HARAInterrupts.c

```
/*
 * HARAInterrupts.c
 *
 *  Created on: Apr 16, 2019
 *      Author: Michael Bolt
 *
 *      A convenient place to store the ISRs!
 */

#ifndef HARAINTERRUPTS_C
#define HARAINTERRUPTS_C

//includes
#include <stdint.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include "inc/hw_types.h"
#include "inc/hw_memmap.h"
#include "inc/hw_ints.h"
#include "driverlib/sysctl.h"
#include "driverlib/interrupt.h"
#include "driverlib/gpio.h"
#include "driverlib/ssi.h"
#include "driverlib/i2c.h"
#include "driverlib/adc.h"
#include "driverlib/timer.h"
#include "driverlib/pin_map.h"
#include "driverlib/uart.h"
#include "FIFO.h"
#include "HARABoosterPack.h"


//globals
extern HARA hara[4];
extern FIFO cmd_fifo;
//TimerA
unsigned int timeoutCounter = 0;    //counter to implement I2C timeout
//TimerB
unsigned int pfd_cnt = 10;          //counter for PFD timeout
//ADC
unsigned int ADC_data[10];          //ADC data buffer
unsigned int adc_waitForSample;     //flag that can be used to wait for a single sample
//flags
unsigned int findingPhaseOffset = 0;    //indicates if we are currently finding the phase offset
unsigned int ledState = 0x00;
unsigned char state;
unsigned int connected = 0;
unsigned int connectionTimer = 10;
unsigned char pkt_cnt = 0;


```

```
//TimerA - 100 Hz (0.01 sec) timer
// - triggers ADC on rollover
// - use counter for I2C / SPI timeout
void Timer0A_ISR(void) {
    //clear ISR flag
    TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
    //decrement timeoutCounter if necessary
    if (timeoutCounter)
        timeoutCounter--;
    //disable timer if necessary
    if (!timeoutCounter) {
        TimerIntDisable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
    }
}

#define ADC_AVG 10
unsigned int adc_samples[4][ADC_AVG];
unsigned long adc_sum;

//ADC - 100 Hz ADC Sample
void ADC_ISR(void) {
    ADCIntClear(ADC0_BASE, 0);  //clear sequence 0 interrupt
    ADCSequenceDataGet(ADC0_BASE, 0, ADC_data); //read the data
    //if waitForSample is set, decrement
    if(adc_waitForSample)
        adc_waitForSample--;
    //update each HARA element
    int n = 3;
    for (n = 3; n != 0; n--) {
        //update vadc averages
        unsigned int i = 0;
        adc_sum = 0;
        for(i = 0; i < ADC_AVG-1; i++) {
            adc_samples[n][i] = adc_samples[n][i+1];
            adc_sum += adc_samples[n][i];
        }
        adc_samples[n][ADC_AVG-1] = ADC_data[n];
        adc_sum += ADC_data[n];
        hara[n].vadc = adc_sum / ADC_AVG;
        //find theta from LUT and update psi
        //theta = psi - d_i -> cpLUT[adc] = psi - d_i
        //theta = psi, so theta = cpLUT[adc] - delta_i;
        hara[n].theta = cpLUT(hara[n].vadc, n) - hara[n].delta_i;
        while(hara[n].theta > 359)  hara[n].theta -= 360;
        while(hara[n].theta < 0)    hara[n].theta += 360;
        hara[n].psi = hara[n].theta;
    }
}




//Timer0B - 10 Hz (0.1 sec) timer
//  - samples PFD mux pin
//  - triggers UART comms
//  - toggles LED on board to show it's alive
```

```c
void Timer0B_ISR(void) {
    TimerIntClear(TIMER0_BASE, TIMER_TIMB_TIMEOUT);

    //push command to read PFD mux values
    push(&cmd_fifo, CMD_PFD_MUX);

    //////////////////
    //  UART Comms  //
    //////////////////
    //if we aren't connected, send a known byte every second
    if(!connected) {
        connectionTimer--;
        if(!connectionTimer) {
            connectionTimer = 10;
            unsigned char buf[2] = {UART_SYNC_WORD, '\n'};
            UARTsend(buf, sizeof buf);
        }
    }
    else {
        //finish out the last second of waiting
        if(connectionTimer)
            connectionTimer--;
        else {
            //if 50th packet
            if(!pkt_cnt) {
                push(&cmd_fifo, CMD_UART_MSG | 0x0000);
                pkt_cnt = 49;
            }
            //normal packets
            else {
                if (pkt_cnt % 2) {
                    push(&cmd_fifo, CMD_UART_MSG | 0x0001);
                }
                else {
                    push(&cmd_fifo, CMD_UART_MSG | 0x0002);
                }
                pkt_cnt--;
            }
        }
    }
    //toggle LED
    ledState ^= 0x01;
    GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_0, ledState);
}



//UART
// - UART Rx/RxTimeout interrupt
void UART_ISR(void) {
    UARTIntClear(UART0_BASE, (UART_INT_RX | UART_INT_RT));
    //if characters are available..
    while(UARTCharsAvail(UART0_BASE)) {
        unsigned char buf[2];
        //if we aren't connected yet
        if(!connected) {
            buf[0] = UARTCharGetNonBlocking(UART0_BASE);
```

```
            //check for sync word message
            if(buf[0] == UART_SYNC_WORD) {
                connected = 1;
            }
        }
        //otherwise, if this is a command..
        else {
            buf[0] = UARTCharGetNonBlocking(UART0_BASE);
            buf[1] = UARTCharGetNonBlocking(UART0_BASE);
            push(&cmd_fifo, ((buf[0] << 8) | buf[1]));
        }
    }
}



#endif
```

A.4.5 FIFO.h

```
/*
 * FIFO.h
 *
 *  Created on: Apr 18, 2019
 *      Author: Michael Bolt
 */

#ifndef FIFO_H_
#define FIFO_H_

//commands
#define CMD_UART_MSG                0x0000

#define CMD_PFD_MUX                 0x2000
//#define CMD_SW_CHANGE_CP_SW         0x3000
#define CMD_SW_PLL_RECONFIGURE      0x4000
#define CMD_SW_PLL_RECAL            0x5000
//#define CMD_SW_FIND_TX_PHASE_OFFSET 0x6000
//#define CMD_SW_FIND_RX_PHASE_OFFSET 0x7000
#define CMD_SW_ADJUST_DELTA_MANUALLY    0x7000
#define CMD_SW_SET_DELTAS_BY_CP     0x8000
#define CMD_SW_CHANGE_PFD_MUX_OUT   0x9000
#define CMD_SW_ACTIVE_STEER         0xA000
#define CMD_SW_STOP_ACTIVE_STEER    0xB000
#define CMD_SW_RETRODIRECT          0xC000
#define CMD_SW_STOP_RETRODIRECT     0xD000
#define CMD_SW_DAC_WRITE            0xE000
#define CMD_SW_PHASE_WRITE          0xF000
//states
#define STATE_FINDING_PHASE_OFFSET  0x01
#define STATE_FINDING_VCAL          0x02
#define STATE_FINDING_BALANCE_POINT 0x03
#define STATE_ACTIVE_STEER          0x04
#define STATE_RETRODIRECT           0x05
//UART definitions
#define UART_SYNC_WORD  0xBA                //message that must be received to establish serial connection



//FIFO structure
#define FIFO_SIZE   100
typedef struct FIFO_S {
    unsigned short    front;              //pointer to front of FIFO
    unsigned short    rear;               //pointer to rear of FIFO
    unsigned short    size;               //current size of FIFO
    unsigned short    capacity;           //capacity of FIFO
    unsigned short    fifo[FIFO_SIZE];    //fifo array
} FIFO;


void            clearFIFO(FIFO *fifo);
unsigned int    isFull(FIFO* fifo);
unsigned int    isEmpty(FIFO* fifo);
```

```
void             push(FIFO* fifo, unsigned int val);
unsigned int     pop(FIFO* fifo);
unsigned int     front(FIFO*);
unsigned int     rear(FIFO*);

#endif /* FIFO_H_ */
```

A.4.6 FIFO.c

```c
/*
 * FIFO.c
 *
 *  Created on: Apr 18, 2019
 *      Author: Michael Bolt
 */


#include <stdint.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include "inc/hw_types.h"
#include "inc/hw_memmap.h"
#include "inc/hw_ints.h"
#include "driverlib/sysctl.h"
#include "driverlib/interrupt.h"
#include "driverlib/gpio.h"
#include "driverlib/ssi.h"
#include "driverlib/i2c.h"
#include "driverlib/adc.h"
#include "driverlib/timer.h"
#include "driverlib/pin_map.h"
//#include "HARABoosterPack.h"
#include "FIFO.h"


FIFO            cmd_fifo;    //fifo for commands
unsigned int    cmd;         //last value popped from the fifo





//*****************************************************************************
//
//! Initializes FIFO structure given a few parameters
//!
//! This function constructs and initializes a FIFO buffer. Example:
//!     createFIFO(&fifo);
//!
//! \return None.
//
//*****************************************************************************
void   clearFIFO(FIFO *fifo) {
    fifo->capacity = FIFO_SIZE; //size of FIFO
    fifo->size = 0;            //FIFO starts empty
    fifo->front = 0;           //FIFO starts pointing to 0th spot
    fifo->rear = FIFO_SIZE - 1; //the first time you add to the fifo,
                                //it will push rear around

 }
```

```
//*****************************************************************************
//
//! Determines if the FIFO structure is full
//!
//! \param fifo is the FIFO structure to check
//!
//! This function determines if a FIFO is full by checking if the current size
//! is equal to the capacity of the FIFO
//!
//! \return logical comparison of current FIFO size to FIFO capacity
//
//*****************************************************************************
unsigned int    isFull(FIFO* fifo) {
    return (fifo->size == fifo->capacity);
}


//*****************************************************************************
//
//! Determines if the FIFO structure is empty
//!
//! \param fifo is the FIFO structure to check
//!
//! This function determines if a FIFO is empty by checking if the current size
//! is 0
//!
//! \return logical comparison of current FIFO size to 0
//
//*****************************************************************************
unsigned int    isEmpty(FIFO* fifo) {
    return (fifo->size == 0);
}


//*****************************************************************************
//
//! places a new value at the rear of the FIFO buffer
//!
//! \param fifo is the FIFO structure to push the new value to
//! \param val is the unsigned character to place at the end of the FIFO
//!
//! This function places a new value in the FIFO buffer
//!
//! \return None.
//
//*****************************************************************************
```

```
void            push(FIFO* fifo, unsigned int val) {
    //return immediately if FIFO is already full
    if (isFull(fifo))
        return;
    //increment rear pointer cyclically
    fifo->rear = (fifo->rear + 1) % fifo->capacity;
    //place new value into rear spot
    fifo->fifo[fifo->rear] = val;
    //update size
    fifo->size++;
}


//*****************************************************************************
//
//! pops the value from the front of the FIFO buffer
//!
//! \param fifo is the FIFO structure to pop the value from
//!
//! This function pops the front value from the FIFO buffer
//!
//! \return unsigned char at the front of the FIFO buffer or 0 if no value is
//! available
//
//*****************************************************************************
unsigned int    pop(FIFO* fifo) {
    if (isEmpty(fifo))
        return 0;
    //pull current front
    unsigned int val = fifo->fifo[fifo->front];
    //increment front pointer cyclically
    fifo->front = (fifo->front + 1) % fifo->capacity;
    //update size
    fifo->size--;
    //return val
    return val;
}


//*****************************************************************************
//
//! reads the current front value of the FIFO buffer, but does NOT pop it off
//!
//! \param fifo is the FIFO structure to read the front of
//!
//! This function reads the current front value of the FIFO buffer without
//! popping it off
//!
//! \return unsigned char at the front of the FIFO buffer or 0 if no value is
//! available
//
```

```
//**************************************************************************
unsigned int    front(FIFO* fifo) {
    if (isEmpty(fifo))
        return 0;
    return fifo->fifo[fifo->front];
}


//**************************************************************************
//
//! reads the current rear value of the FIFO buffer, but does NOT pop it off
//!
//! \param fifo is the FIFO structure to read the rear of
//!
//! This function reads the current rear value of the FIFO buffer without
//! popping it off
//!
//! \return unsigned char at the rear of the FIFO buffer or 0 if no value is
//! available
//
//**************************************************************************
unsigned int    rear(FIFO* fifo) {
    if (isEmpty(fifo))
        return 0;
    return fifo->fifo[fifo->rear];
}
```

A.4.7 HARA_LUTs.h

```
/*
 * HARA_LUTs.h
 *
 *  Created on: May 28, 2019
 *      Author: MichaelBolt
 */

#ifndef HARA_LUTS_H_
#define HARA_LUTS_H_

#include <stdint.h>

uint16_t txLUT(unsigned int phase, unsigned int board);
uint16_t rxLUT(unsigned int phase, unsigned int board);
uint16_t cpLUT(unsigned int adc, unsigned int board);

uint8_t activeSteer_LUT(unsigned int point);


#endif /* HARA_LUTS_H_ */
```

A.4.8 txLUT.c

```
/*
 *    txLUT.c
 *
 *    Generated with data in: TxTester_F_2500MHz_100MHz_3
 */

#include "HARA_LUTs.h"


//****************************************************************************
//! Finds DAC value to write to achieve a phase difference
//!
//! \param phase is the phase (0-359) that you would like to write to the
//!             TX DAC in question
//! \param board is the hara element being used for the phase shifter (1-3)
//!
//! This function returns a 12-bit value from a LUT containing DAC values that
//! correspond to decimal degrees (0 - 359).
//!
//! \return DAC value to write to achieve the intended phase
//****************************************************************************
uint16_t txLUT(unsigned int phase, unsigned int board) {
    static const uint16_t txLUT_1[360] = {
        3997, 3984, 3966, 3948, 3931, 3914, 3896, 3879, 3862, 3845,
        3828, 3811, 3794, 3777, 3760, 3744, 3727, 3711, 3694, 3678,
        3661, 3645, 3629, 3613, 3596, 3580, 3564, 3548, 3532, 3517,
        3501, 3485, 3469, 3454, 3438, 3423, 3407, 3392, 3377, 3361,
        3346, 3331, 3316, 3301, 3287, 3272, 3257, 3242, 3228, 3213,
        3199, 3185, 3170, 3156, 3142, 3128, 3114, 3100, 3086, 3072,
        3059, 3045, 3031, 3018, 3004, 2991, 2977, 2964, 2951, 2938,
        2925, 2912, 2899, 2886, 2874, 2861, 2848, 2836, 2823, 2811,
        2799, 2786, 2774, 2762, 2750, 2738, 2726, 2714, 2702, 2691,
        2679, 2667, 2656, 2644, 2633, 2621, 2610, 2599, 2588, 2576,
        2565, 2554, 2543, 2533, 2522, 2511, 2500, 2490, 2479, 2469,
        2458, 2448, 2438, 2427, 2417, 2407, 2397, 2387, 2377, 2367,
        2357, 2347, 2337, 2327, 2317, 2308, 2298, 2288, 2279, 2269,
        2260, 2250, 2241, 2232, 2222, 2213, 2204, 2195, 2186, 2177,
        2168, 2159, 2150, 2141, 2132, 2124, 2115, 2106, 2098, 2089,
        2080, 2072, 2063, 2055, 2047, 2038, 2030, 2022, 2014, 2005,
        1997, 1989, 1981, 1973, 1965, 1957, 1949, 1941, 1933, 1925,
        1918, 1910, 1902, 1894, 1887, 1879, 1872, 1864, 1856, 1849,
        1842, 1834, 1827, 1819, 1812, 1805, 1798, 1790, 1783, 1776,
        1769, 1762, 1755, 1748, 1741, 1734, 1727, 1720, 1713, 1706,
        1699, 1693, 1686, 1679, 1673, 1666, 1659, 1653, 1646, 1640,
        1633, 1627, 1620, 1614, 1607, 1601, 1595, 1588, 1582, 1576,
        1570, 1563, 1557, 1551, 1545, 1539, 1533, 1527, 1521, 1515,
        1509, 1503, 1497, 1491, 1485, 1479, 1473, 1468, 1462, 1456,
        1450, 1445, 1439, 1433, 1428, 1422, 1416, 1411, 1405, 1400,
        1394, 1389, 1383, 1378, 1372, 1367, 1362, 1356, 1351, 1346,
        1340, 1335, 1330, 1325, 1320, 1314, 1309, 1304, 1299, 1294,
        1289, 1284, 1279, 1274, 1269, 1264, 1259, 1254, 1249, 1244,
```

```
        1240, 1235, 1230, 1225, 1220, 1216, 1211, 1206, 1201, 1197,
        1192, 1187, 1183, 1178, 1174, 1169, 1165, 1160, 1156, 1151,
        1147, 1142, 1138, 1133, 1129, 1124, 1120, 1116, 1111, 1107,
        1103, 1099, 1094, 1090, 1086, 1082, 1077, 1073, 1069, 1065,
        1061, 1057, 1052, 1048, 1044, 1040, 1036, 1032, 1028, 1024,
        1020, 1016, 1012, 1008, 1004, 1000,  996,  993,  989,  985,
         981,  977,  973,  969,  966,  962,  958,  954,  951,  947,
         943,  939,  936,  932,  928,  925,  921,  918,  914,  910,
        };
    static const uint16_t txLUT_2[360] = {
        3997, 3985, 3969, 3953, 3937, 3921, 3905, 3889, 3874, 3858,
        3843, 3827, 3812, 3797, 3782, 3767, 3752, 3738, 3723, 3708,
        3694, 3679, 3665, 3651, 3637, 3623, 3609, 3595, 3581, 3567,
        3553, 3539, 3525, 3512, 3498, 3485, 3471, 3458, 3445, 3431,
        3418, 3405, 3392, 3379, 3366, 3354, 3341, 3328, 3316, 3303,
        3290, 3278, 3266, 3253, 3241, 3229, 3217, 3205, 3193, 3181,
        3169, 3157, 3145, 3133, 3121, 3109, 3098, 3086, 3075, 3063,
        3052, 3040, 3029, 3018, 3007, 2995, 2984, 2973, 2962, 2951,
        2941, 2930, 2919, 2908, 2898, 2887, 2876, 2866, 2856, 2845,
        2835, 2824, 2814, 2804, 2794, 2784, 2774, 2764, 2754, 2744,
        2734, 2724, 2714, 2704, 2694, 2685, 2675, 2665, 2656, 2646,
        2637, 2627, 2618, 2608, 2599, 2590, 2580, 2571, 2562, 2553,
        2544, 2535, 2526, 2517, 2508, 2499, 2490, 2482, 2473, 2464,
        2455, 2447, 2438, 2430, 2421, 2413, 2404, 2396, 2387, 2379,
        2371, 2363, 2354, 2346, 2338, 2330, 2322, 2314, 2306, 2298,
        2290, 2282, 2274, 2266, 2258, 2250, 2242, 2234, 2227, 2219,
        2211, 2204, 2196, 2188, 2181, 2173, 2166, 2158, 2151, 2143,
        2136, 2129, 2121, 2114, 2107, 2100, 2092, 2085, 2078, 2071,
        2064, 2057, 2050, 2043, 2036, 2029, 2022, 2015, 2008, 2001,
        1994, 1987, 1981, 1974, 1967, 1961, 1954, 1947, 1941, 1934,
        1927, 1921, 1914, 1908, 1901, 1895, 1889, 1882, 1876, 1869,
        1863, 1857, 1850, 1844, 1838, 1832, 1826, 1819, 1813, 1807,
        1801, 1795, 1789, 1783, 1777, 1771, 1765, 1759, 1753, 1747,
        1741, 1735, 1729, 1723, 1717, 1712, 1706, 1700, 1694, 1689,
        1683, 1677, 1672, 1666, 1660, 1655, 1649, 1643, 1638, 1632,
        1627, 1621, 1616, 1610, 1605, 1600, 1594, 1589, 1583, 1578,
        1573, 1567, 1562, 1557, 1552, 1546, 1541, 1536, 1531, 1526,
        1520, 1515, 1510, 1505, 1500, 1495, 1490, 1485, 1480, 1475,
        1470, 1465, 1460, 1455, 1450, 1445, 1440, 1436, 1431, 1426,
        1421, 1416, 1412, 1407, 1402, 1397, 1393, 1388, 1383, 1379,
        1374, 1369, 1365, 1360, 1356, 1351, 1347, 1342, 1337, 1333,
        1328, 1324, 1320, 1315, 1311, 1306, 1302, 1297, 1293, 1289,
        1284, 1280, 1276, 1271, 1267, 1263, 1259, 1254, 1250, 1246,
        1242, 1237, 1233, 1229, 1225, 1221, 1217, 1213, 1209, 1204,
        1200, 1196, 1192, 1188, 1184, 1180, 1176, 1172, 1168, 1164,
        1160, 1156, 1152, 1149, 1145, 1141, 1137, 1133, 1129, 1125,
        };
    static const uint16_t txLUT_3[360] = {
        3998, 3988, 3975, 3962, 3949, 3937, 3924, 3912, 3900, 3888,
        3876, 3864, 3852, 3840, 3829, 3817, 3806, 3794, 3783, 3771,
        3760, 3749, 3738, 3726, 3715, 3704, 3693, 3682, 3671, 3661,
        3650, 3639, 3628, 3617, 3607, 3596, 3585, 3574, 3564, 3553,
        3543, 3532, 3521, 3511, 3500, 3490, 3479, 3469, 3459, 3448,
        3438, 3427, 3417, 3407, 3396, 3386, 3376, 3365, 3355, 3345,
        3335, 3325, 3314, 3304, 3294, 3284, 3274, 3264, 3254, 3244,
        3234, 3224, 3214, 3204, 3194, 3184, 3175, 3165, 3155, 3145,
        3135, 3126, 3116, 3106, 3097, 3087, 3077, 3068, 3058, 3049,
```

```
        3039, 3030, 3021, 3011, 3002, 2993, 2983, 2974, 2965, 2956,
        2947, 2937, 2928, 2919, 2910, 2901, 2892, 2884, 2875, 2866,
        2857, 2848, 2840, 2831, 2822, 2814, 2805, 2796, 2788, 2779,
        2771, 2763, 2754, 2746, 2738, 2729, 2721, 2713, 2705, 2696,
        2688, 2680, 2672, 2664, 2656, 2648, 2640, 2632, 2624, 2616,
        2609, 2601, 2593, 2585, 2577, 2570, 2562, 2555, 2547, 2539,
        2532, 2524, 2517, 2509, 2502, 2495, 2487, 2480, 2473, 2465,
        2458, 2451, 2444, 2436, 2429, 2422, 2415, 2408, 2401, 2394,
        2387, 2380, 2373, 2366, 2360, 2353, 2346, 2339, 2333, 2326,
        2319, 2312, 2306, 2299, 2293, 2286, 2279, 2273, 2266, 2260,
        2254, 2247, 2241, 2234, 2228, 2222, 2215, 2209, 2203, 2196,
        2190, 2184, 2178, 2172, 2165, 2159, 2153, 2147, 2141, 2135,
        2129, 2123, 2117, 2111, 2105, 2099, 2093, 2087, 2081, 2076,
        2070, 2064, 2058, 2052, 2046, 2041, 2035, 2029, 2024, 2018,
        2012, 2007, 2001, 1995, 1990, 1984, 1979, 1973, 1968, 1962,
        1957, 1951, 1946, 1940, 1935, 1929, 1924, 1919, 1913, 1908,
        1903, 1897, 1892, 1887, 1881, 1876, 1871, 1866, 1861, 1855,
        1850, 1845, 1840, 1835, 1830, 1825, 1819, 1814, 1809, 1804,
        1799, 1794, 1789, 1784, 1779, 1774, 1770, 1765, 1760, 1755,
        1750, 1745, 1740, 1735, 1731, 1726, 1721, 1716, 1712, 1707,
        1702, 1697, 1693, 1688, 1683, 1679, 1674, 1669, 1665, 1660,
        1655, 1651, 1646, 1642, 1637, 1633, 1628, 1624, 1619, 1615,
        1610, 1606, 1601, 1597, 1592, 1588, 1583, 1579, 1575, 1570,
        1566, 1562, 1557, 1553, 1549, 1544, 1540, 1536, 1531, 1527,
        1523, 1519, 1515, 1510, 1506, 1502, 1498, 1494, 1489, 1485,
        1481, 1477, 1473, 1469, 1465, 1461, 1457, 1452, 1448, 1444,
        1440, 1436, 1432, 1428, 1424, 1420, 1416, 1412, 1408, 1404,
        };
    //wrap phase
    if (phase > 359)
        phase = 359;
    //return the DAC value from the correct LUT
    switch (board) {
        case 1:
            return txLUT_1[phase];
        case 2:
            return txLUT_2[phase];
        case 3:
            return txLUT_3[phase];
        default:
            return 2048;
    }
}
```

A.4.9 rxLUT.c

```c
/*
 *    rxLUT.c
 *
 *    Generated with data in: RxTester_Terminated_2410MHz_100MHz_3
 */

#include "HARA_LUTs.h"


//****************************************************************************
//! Finds DAC value to write to achieve a phase difference
//!
//! \param phase is the phase (0-359) that you would like to write to the
//!              RX DAC in question
//! \param board is the hara element being used for the phase shifter (1-3)
//!
//! This function returns a 12-bit value from a LUT containing DAC values that
//! correspond to decimal degrees (0 - 359).
//!
//! \return DAC value to write to achieve the intended phase
//****************************************************************************
uint16_t rxLUT(unsigned int phase, unsigned int board) {
    static const uint16_t rxLUT_1[360] = {
        3996, 3979, 3958, 3936, 3915, 3894, 3874, 3854, 3834, 3814,
        3794, 3774, 3754, 3735, 3715, 3696, 3677, 3658, 3639, 3621,
        3602, 3584, 3565, 3547, 3529, 3511, 3493, 3475, 3457, 3439,
        3422, 3404, 3387, 3369, 3352, 3335, 3317, 3300, 3284, 3267,
        3250, 3233, 3217, 3201, 3184, 3168, 3152, 3136, 3120, 3104,
        3088, 3072, 3057, 3041, 3026, 3010, 2995, 2980, 2965, 2950,
        2935, 2921, 2906, 2892, 2877, 2863, 2849, 2835, 2821, 2807,
        2793, 2779, 2765, 2751, 2738, 2724, 2711, 2698, 2684, 2671,
        2658, 2645, 2633, 2620, 2607, 2595, 2582, 2570, 2558, 2545,
        2533, 2521, 2509, 2497, 2485, 2473, 2462, 2450, 2438, 2427,
        2415, 2404, 2392, 2381, 2370, 2359, 2348, 2337, 2326, 2315,
        2304, 2294, 2283, 2273, 2262, 2252, 2241, 2231, 2221, 2210,
        2200, 2190, 2180, 2170, 2160, 2150, 2140, 2131, 2121, 2111,
        2102, 2092, 2082, 2073, 2064, 2054, 2045, 2036, 2026, 2017,
        2008, 1999, 1990, 1981, 1972, 1963, 1954, 1946, 1937, 1928,
        1920, 1911, 1902, 1894, 1886, 1877, 1869, 1860, 1852, 1844,
        1836, 1828, 1819, 1811, 1803, 1795, 1787, 1779, 1771, 1764,
        1756, 1748, 1740, 1732, 1725, 1717, 1709, 1702, 1694, 1687,
        1679, 1672, 1664, 1657, 1650, 1642, 1635, 1628, 1621, 1614,
        1607, 1599, 1592, 1585, 1578, 1571, 1565, 1558, 1551, 1544,
        1537, 1531, 1524, 1517, 1511, 1504, 1497, 1491, 1484, 1478,
        1471, 1465, 1459, 1452, 1446, 1440, 1433, 1427, 1421, 1415,
        1408, 1402, 1396, 1390, 1384, 1378, 1372, 1366, 1360, 1354,
        1348, 1342, 1336, 1331, 1325, 1319, 1313, 1307, 1302, 1296,
        1290, 1285, 1279, 1274, 1268, 1262, 1257, 1251, 1246, 1241,
        1235, 1230, 1224, 1219, 1214, 1208, 1203, 1198, 1193, 1187,
        1182, 1177, 1172, 1167, 1162, 1157, 1152, 1147, 1142, 1137,
        1132, 1127, 1122, 1117, 1112, 1107, 1103, 1098, 1093, 1088,
```

```
        1084, 1079, 1074, 1069, 1065, 1060, 1056, 1051, 1046, 1042,
        1037, 1033, 1028, 1024, 1019, 1015, 1010, 1006, 1002,  997,
         993,  989,  984,  980,  976,  971,  967,  963,  959,  955,
         950,  946,  942,  938,  934,  930,  926,  922,  918,  913,
         909,  905,  901,  897,  894,  890,  886,  882,  878,  874,
         870,  866,  862,  858,  855,  851,  847,  843,  840,  836,
         832,  828,  825,  821,  817,  814,  810,  806,  803,  799,
         796,  792,  788,  785,  781,  778,  774,  771,  767,  764,
        };
    static const uint16_t rxLUT_2[360] = {
        3997, 3985, 3968, 3952, 3936, 3920, 3904, 3889, 3873, 3858,
        3842, 3827, 3812, 3797, 3782, 3768, 3753, 3738, 3724, 3709,
        3695, 3681, 3667, 3652, 3638, 3624, 3610, 3596, 3582, 3569,
        3555, 3541, 3527, 3514, 3500, 3487, 3473, 3460, 3447, 3433,
        3420, 3407, 3394, 3381, 3368, 3355, 3342, 3329, 3316, 3303,
        3291, 3278, 3266, 3253, 3241, 3228, 3216, 3204, 3191, 3179,
        3167, 3155, 3143, 3131, 3119, 3107, 3095, 3083, 3071, 3060,
        3048, 3036, 3025, 3013, 3002, 2990, 2979, 2968, 2956, 2945,
        2934, 2923, 2912, 2901, 2890, 2879, 2868, 2858, 2847, 2836,
        2826, 2815, 2805, 2794, 2784, 2774, 2763, 2753, 2743, 2733,
        2723, 2712, 2702, 2692, 2682, 2673, 2663, 2653, 2643, 2633,
        2624, 2614, 2604, 2595, 2585, 2576, 2566, 2557, 2548, 2538,
        2529, 2520, 2511, 2501, 2492, 2483, 2474, 2465, 2456, 2448,
        2439, 2430, 2421, 2413, 2404, 2395, 2387, 2378, 2370, 2361,
        2353, 2344, 2336, 2327, 2319, 2311, 2303, 2294, 2286, 2278,
        2270, 2262, 2254, 2246, 2238, 2230, 2222, 2214, 2206, 2199,
        2191, 2183, 2175, 2168, 2160, 2152, 2145, 2137, 2129, 2122,
        2114, 2107, 2100, 2092, 2085, 2077, 2070, 2063, 2056, 2048,
        2041, 2034, 2027, 2020, 2013, 2006, 1999, 1992, 1985, 1978,
        1971, 1964, 1957, 1950, 1944, 1937, 1930, 1923, 1917, 1910,
        1903, 1897, 1890, 1884, 1877, 1871, 1864, 1858, 1851, 1845,
        1838, 1832, 1826, 1819, 1813, 1807, 1801, 1794, 1788, 1782,
        1776, 1770, 1764, 1758, 1752, 1746, 1740, 1734, 1728, 1722,
        1716, 1710, 1704, 1698, 1692, 1686, 1680, 1675, 1669, 1663,
        1657, 1652, 1646, 1640, 1635, 1629, 1623, 1618, 1612, 1607,
        1601, 1596, 1590, 1585, 1579, 1574, 1568, 1563, 1558, 1552,
        1547, 1542, 1536, 1531, 1526, 1521, 1515, 1510, 1505, 1500,
        1495, 1490, 1485, 1479, 1474, 1469, 1464, 1459, 1454, 1449,
        1444, 1439, 1435, 1430, 1425, 1420, 1415, 1410, 1405, 1401,
        1396, 1391, 1386, 1382, 1377, 1372, 1368, 1363, 1358, 1354,
        1349, 1344, 1340, 1335, 1331, 1326, 1322, 1317, 1313, 1308,
        1304, 1299, 1295, 1291, 1286, 1282, 1277, 1273, 1269, 1264,
        1260, 1256, 1251, 1247, 1243, 1239, 1235, 1230, 1226, 1222,
        1218, 1214, 1209, 1205, 1201, 1197, 1193, 1189, 1185, 1181,
        1177, 1173, 1169, 1165, 1161, 1157, 1153, 1149, 1145, 1141,
        1137, 1133, 1129, 1125, 1122, 1118, 1114, 1110, 1106, 1102,
        };
    static const uint16_t rxLUT_3[360] = {
        3999, 3990, 3978, 3967, 3955, 3944, 3932, 3921, 3909, 3898,
        3886, 3875, 3864, 3852, 3841, 3829, 3818, 3807, 3795, 3784,
        3773, 3762, 3750, 3739, 3728, 3717, 3706, 3695, 3684, 3673,
        3662, 3651, 3640, 3629, 3618, 3607, 3597, 3586, 3575, 3565,
        3554, 3544, 3533, 3523, 3512, 3502, 3492, 3481, 3471, 3461,
        3451, 3440, 3430, 3420, 3410, 3400, 3390, 3380, 3370, 3360,
        3351, 3341, 3331, 3321, 3312, 3302, 3292, 3283, 3273, 3264,
        3254, 3245, 3235, 3226, 3217, 3207, 3198, 3189, 3180, 3171,
        3162, 3152, 3143, 3134, 3125, 3117, 3108, 3099, 3090, 3081,
```

```
        3073, 3064, 3055, 3046, 3038, 3029, 3021, 3012, 3004, 2995,
        2987, 2978, 2970, 2962, 2953, 2945, 2937, 2928, 2920, 2912,
        2904, 2896, 2887, 2879, 2871, 2863, 2855, 2847, 2839, 2831,
        2823, 2815, 2808, 2800, 2792, 2784, 2776, 2769, 2761, 2753,
        2745, 2738, 2730, 2722, 2715, 2707, 2700, 2692, 2685, 2677,
        2670, 2662, 2655, 2647, 2640, 2633, 2625, 2618, 2611, 2603,
        2596, 2589, 2582, 2575, 2567, 2560, 2553, 2546, 2539, 2532,
        2525, 2518, 2511, 2504, 2497, 2490, 2483, 2476, 2469, 2462,
        2455, 2449, 2442, 2435, 2428, 2421, 2415, 2408, 2401, 2395,
        2388, 2381, 2375, 2368, 2361, 2355, 2348, 2342, 2335, 2329,
        2322, 2316, 2309, 2303, 2296, 2290, 2284, 2277, 2271, 2264,
        2258, 2252, 2246, 2239, 2233, 2227, 2220, 2214, 2208, 2202,
        2196, 2190, 2183, 2177, 2171, 2165, 2159, 2153, 2147, 2141,
        2135, 2129, 2123, 2117, 2111, 2105, 2099, 2093, 2087, 2081,
        2075, 2070, 2064, 2058, 2052, 2046, 2041, 2035, 2029, 2023,
        2018, 2012, 2006, 2000, 1995, 1989, 1984, 1978, 1972, 1967,
        1961, 1956, 1950, 1944, 1939, 1933, 1928, 1922, 1917, 1912,
        1906, 1901, 1895, 1890, 1884, 1879, 1874, 1868, 1863, 1858,
        1852, 1847, 1842, 1837, 1831, 1826, 1821, 1816, 1811, 1805,
        1800, 1795, 1790, 1785, 1780, 1775, 1770, 1764, 1759, 1754,
        1749, 1744, 1739, 1734, 1729, 1724, 1719, 1715, 1710, 1705,
        1700, 1695, 1690, 1685, 1680, 1676, 1671, 1666, 1661, 1656,
        1652, 1647, 1642, 1637, 1633, 1628, 1623, 1619, 1614, 1609,
        1605, 1600, 1595, 1591, 1586, 1582, 1577, 1573, 1568, 1564,
        1559, 1554, 1550, 1546, 1541, 1537, 1532, 1528, 1523, 1519,
        1514, 1510, 1506, 1501, 1497, 1493, 1488, 1484, 1480, 1475,
        1471, 1467, 1463, 1458, 1454, 1450, 1446, 1441, 1437, 1433,
        };
    //wrap phase
    if (phase > 359)
        phase = 359;
    //return the DAC value from the correct LUT
    switch (board) {
        case 1:
            return rxLUT_1[phase];
        case 2:
            return rxLUT_2[phase];
        case 3:
            return rxLUT_3[phase];
        default:
            return 2048;
    }
}
```

A.4.10 cpLUT.c

```c
/*
 *    cpLUT.c
 *
 *    Generated with data in: RxTester_Terminated_2410MHz_100MHz_3
 */

#include "HARA_LUTs.h"


//**********************************************************************
//! Finds angle corresponding to measured ADC voltage
//!
//! \param adc is the adc value (0-4095) that you want to find the corresponding
//!              angle for
//! \param board is the hara element being used (1-3)
//!
//! This function returns a 12-bit value from a LUT containing angle values that
//! correspond to decimal degrees (0 - 359).
//!
//! \return angle value corresponding to input adc value
//**********************************************************************
uint16_t cpLUT(unsigned int adc, unsigned int board) {
    static const uint16_t cpLUT_1[4096] = {
          0, 359, 359, 358, 357, 357, 356, 355, 355, 354, 353, 353, 352, 352, 351, 350, 350, 349, 348, 348, 347, 346, 346, 345, 344, 344, 343, 343, 342, 341, 341, 340,
        339, 339, 338, 338, 337, 336, 336, 335, 334, 334, 333, 333, 332, 331, 331, 330, 329, 329, 328, 328, 327, 326, 326, 325, 324, 324, 323, 323, 322, 321, 321, 320,
        320, 319, 318, 318, 317, 317, 316, 315, 315, 314, 314, 313, 312, 312, 311, 311, 310, 309, 309, 308, 308, 307, 306, 306, 305, 305, 304, 303, 303, 302, 302, 301,
        300, 300, 299, 299, 298, 298, 297, 296, 296, 295, 295, 294, 293, 293, 292, 292, 291, 291, 290, 289, 289, 288, 288, 287, 287, 286, 285, 285, 284, 284, 283, 283,
        282, 281, 281, 280, 280, 279, 279, 278, 278, 277, 276, 276, 275, 275, 274, 274, 273, 273, 272, 271, 271, 270, 270, 269, 269, 268, 268, 267, 266, 266, 265, 265,
        264, 264, 263, 263, 262, 262, 261, 260, 260, 259, 259, 258, 258, 257, 257, 256, 256, 255, 255, 254, 254, 253, 252, 252, 251, 251, 250, 250, 249, 249, 248, 248,
        247, 247, 246, 246, 245, 245, 244, 243, 243, 242, 242, 241, 241, 240, 240, 239, 239, 238, 238, 237, 237, 236, 236, 235, 235, 234, 234, 233, 233, 232, 232, 231,
        231, 230, 230, 229, 229, 228, 228, 227, 227, 226, 226, 225, 225, 224, 224, 223, 223, 222, 222, 221, 221, 220, 220, 219, 219, 218, 218, 217, 217, 216, 216, 215,
        215, 214, 214, 213, 213, 212, 212, 211, 211, 210, 210, 209, 209, 208, 208, 207, 207, 207, 206, 206, 205, 205, 204, 204, 203, 203, 202, 202, 201, 201, 200, 200,
```

33 199, 199, 198, 198, 198, 197, 197, 196, 196, 195, 195, 194, 194,
193, 193, 192, 192, 191, 191, 191, 190, 190, 189, 189, 188, 188,
187, 187, 186, 186, 186, 185,
34 185, 184, 184, 183, 183, 182, 182, 181, 181, 181, 180, 180, 179,
179, 178, 178, 177, 177, 177, 176, 176, 175, 175, 174, 174, 173,
173, 173, 172, 172, 171, 171,
35 170, 170, 169, 169, 169, 168, 168, 167, 167, 166, 166, 166, 165,
165, 164, 164, 163, 163, 163, 162, 162, 161, 161, 160, 160, 160,
159, 159, 158, 158, 157, 157,
36 157, 156, 156, 155, 155, 154, 154, 154, 153, 153, 152, 152, 152,
151, 151, 150, 150, 149, 149, 149, 148, 148, 147, 147, 147, 146,
146, 145, 145, 145, 144, 144,
37 143, 143, 142, 142, 142, 141, 141, 140, 140, 140, 139, 139, 138,
138, 138, 137, 137, 136, 136, 136, 135, 135, 134, 134, 134, 133,
133, 132, 132, 132, 131, 131,
38 130, 130, 130, 129, 129, 129, 128, 128, 127, 127, 127, 126, 126,
125, 125, 125, 124, 124, 123, 123, 123, 122, 122, 122, 121, 121,
120, 120, 120, 119, 119, 118,
39 118, 118, 117, 117, 117, 116, 116, 115, 115, 115, 114, 114, 114,
113, 113, 112, 112, 112, 111, 111, 111, 110, 110, 109, 109, 109,
108, 108, 108, 107, 107, 106,
40 106, 106, 105, 105, 105, 104, 104, 104, 103, 103, 102, 102, 102,
101, 101, 101, 100, 100, 100, 99, 99, 99, 98, 98, 97, 97,
97, 96, 96, 96, 95, 95,
41 95, 94, 94, 94, 93, 93, 92, 92, 92, 91, 91, 91, 90,
90, 90, 89, 89, 89, 88, 88, 88, 87, 87, 87, 86, 86,
85, 85, 85, 84, 84, 84,
42 83, 83, 83, 82, 82, 82, 81, 81, 81, 80, 80, 80, 79,
79, 79, 78, 78, 78, 77, 77, 77, 76, 76, 76, 75, 75,
75, 74, 74, 74, 73, 73,
43 73, 72, 72, 72, 71, 71, 71, 70, 70, 70, 69, 69, 69,
68, 68, 68, 67, 67, 67, 66, 66, 66, 65, 65, 65, 64,
64, 64, 63, 63, 63, 62,
44 62, 62, 62, 61, 61, 61, 60, 60, 60, 59, 59, 59, 58,
58, 58, 57, 57, 57, 56, 56, 56, 56, 55, 55, 55, 54,
54, 54, 53, 53, 53, 52,
45 52, 52, 51, 51, 51, 51, 50, 50, 50, 49, 49, 49, 48,
48, 48, 47, 47, 47, 47, 46, 46, 46, 45, 45, 45, 44,
44, 44, 44, 43, 43, 43,
46 42, 42, 42, 41, 41, 41, 41, 40, 40, 40, 39, 39, 39,
38, 38, 38, 38, 37, 37, 37, 36, 36, 36, 35, 35, 35,
35, 34, 34, 34, 33, 33,
47 33, 33, 32, 32, 32, 31, 31, 31, 31, 30, 30, 30, 29,
29, 29, 29, 28, 28, 28, 27, 27, 27, 27, 26, 26, 26,
25, 25, 25, 25, 24, 24,
48 24, 23, 23, 23, 23, 22, 22, 22, 21, 21, 21, 21, 20,
20, 20, 19, 19, 19, 19, 18, 18, 18, 18, 17, 17, 17,
16, 16, 16, 16, 15, 15,
49 15, 15, 14, 14, 14, 13, 13, 13, 13, 12, 12, 12, 12,
11, 11, 11, 10, 10, 10, 10, 9, 9, 9, 9, 8, 8,
8, 8, 7, 7, 7, 6,
50 6, 6, 6, 5, 5, 5, 5, 4, 4, 4, 4, 3, 3,
3, 3, 2, 2, 2, 1, 1, 1, 1, 0, 0, 360, 360,
359, 359, 359, 359, 358, 358,
51 358, 358, 357, 357, 357, 357, 356, 356, 356, 356, 355, 355, 355,
355, 354, 354, 354, 354, 353, 353, 353, 353, 352, 352, 352, 352,
351, 351, 351, 351, 350, 350,

52 350, 350, 349, 349, 349, 349, 348, 348, 348, 348, 347, 347, 347,
347, 346, 346, 346, 346, 345, 345, 345, 345, 344, 344, 344, 344,
343, 343, 343, 343, 342, 342,
53 342, 342, 341, 341, 341, 341, 340, 340, 340, 340, 340, 339, 339,
339, 339, 338, 338, 338, 338, 337, 337, 337, 337, 336, 336, 336,
336, 336, 335, 335, 335, 335,
54 334, 334, 334, 334, 333, 333, 333, 333, 332, 332, 332, 332, 332,
331, 331, 331, 331, 330, 330, 330, 330, 329, 329, 329, 329, 329,
328, 328, 328, 328, 327, 327,
55 327, 327, 326, 326, 326, 326, 326, 325, 325, 325, 325, 324, 324,
324, 324, 324, 323, 323, 323, 323, 322, 322, 322, 322, 322, 321,
321, 321, 321, 320, 320, 320,
56 320, 320, 319, 319, 319, 319, 318, 318, 318, 318, 318, 317, 317,
317, 317, 316, 316, 316, 316, 316, 315, 315, 315, 315, 314, 314,
314, 314, 314, 313, 313, 313,
57 313, 313, 312, 312, 312, 312, 311, 311, 311, 311, 311, 310, 310,
310, 310, 310, 309, 309, 309, 309, 308, 308, 308, 308, 308, 307,
307, 307, 307, 307, 306, 306,
58 306, 306, 306, 305, 305, 305, 305, 305, 304, 304, 304, 304, 303,
303, 303, 303, 303, 302, 302, 302, 302, 302, 301, 301, 301, 301,
301, 300, 300, 300, 300, 300,
59 299, 299, 299, 299, 299, 298, 298, 298, 298, 298, 297, 297, 297,
297, 297, 296, 296, 296, 296, 296, 295, 295, 295, 295, 295, 294,
294, 294, 294, 294, 293, 293,
60 293, 293, 293, 292, 292, 292, 292, 292, 291, 291, 291, 291, 291,
290, 290, 290, 290, 290, 289, 289, 289, 289, 289, 288, 288, 288,
288, 288, 287, 287, 287, 287,
61 287, 286, 286, 286, 286, 286, 286, 285, 285, 285, 285, 285, 284,
284, 284, 284, 284, 283, 283, 283, 283, 283, 282, 282, 282, 282,
282, 282, 281, 281, 281, 281,
62 281, 280, 280, 280, 280, 280, 279, 279, 279, 279, 279, 279, 278,
278, 278, 278, 278, 277, 277, 277, 277, 277, 276, 276, 276, 276,
276, 276, 275, 275, 275, 275,
63 275, 274, 274, 274, 274, 274, 274, 273, 273, 273, 273, 273, 272,
272, 272, 272, 272, 272, 271, 271, 271, 271, 271, 270, 270, 270,
270, 270, 270, 269, 269, 269,
64 269, 269, 268, 268, 268, 268, 268, 268, 267, 267, 267, 267, 267,
267, 266, 266, 266, 266, 266, 265, 265, 265, 265, 265, 265, 264,
264, 264, 264, 264, 264, 263,
65 263, 263, 263, 263, 262, 262, 262, 262, 262, 262, 261, 261, 261,
261, 261, 261, 260, 260, 260, 260, 260, 260, 259, 259, 259, 259,
259, 259, 258, 258, 258, 258,
66 258, 257, 257, 257, 257, 257, 257, 256, 256, 256, 256, 256, 256,
255, 255, 255, 255, 255, 255, 254, 254, 254, 254, 254, 254, 253,
253, 253, 253, 253, 253, 252,
67 252, 252, 252, 252, 252, 251, 251, 251, 251, 251, 251, 250, 250,
250, 250, 250, 250, 249, 249, 249, 249, 249, 249, 248, 248, 248,
248, 248, 248, 247, 247, 247,
68 247, 247, 247, 247, 246, 246, 246, 246, 246, 246, 245, 245, 245,
245, 245, 245, 244, 244, 244, 244, 244, 244, 243, 243, 243, 243,
243, 243, 243, 242, 242, 242,
69 242, 242, 242, 241, 241, 241, 241, 241, 241, 240, 240, 240, 240,
240, 240, 239, 239, 239, 239, 239, 239, 239, 238, 238, 238, 238,
238, 238, 237, 237, 237, 237,
70 237, 237, 237, 236, 236, 236, 236, 236, 236, 235, 235, 235, 235,
235, 235, 235, 234, 234, 234, 234, 234, 234, 233, 233, 233, 233,
233, 233, 233, 232, 232, 232,

71 232, 232, 232, 231, 231, 231, 231, 231, 231, 231, 230, 230, 230,
230, 230, 230, 230, 229, 229, 229, 229, 229, 229, 228, 228, 228,
228, 228, 228, 228, 227, 227,
72 227, 227, 227, 227, 227, 226, 226, 226, 226, 226, 226, 225, 225,
225, 225, 225, 225, 225, 224, 224, 224, 224, 224, 224, 224, 223,
223, 223, 223, 223, 223, 223,
73 222, 222, 222, 222, 222, 222, 222, 221, 221, 221, 221, 221, 221,
221, 220, 220, 220, 220, 220, 220, 220, 219, 219, 219, 219, 219,
219, 219, 218, 218, 218, 218,
74 218, 218, 218, 217, 217, 217, 217, 217, 217, 217, 216, 216, 216,
216, 216, 216, 216, 215, 215, 215, 215, 215, 215, 215, 214, 214,
214, 214, 214, 214, 214, 213,
75 213, 213, 213, 213, 213, 213, 212, 212, 212, 212, 212, 212, 212,
212, 211, 211, 211, 211, 211, 211, 211, 210, 210, 210, 210, 210,
210, 210, 209, 209, 209, 209,
76 209, 209, 209, 208, 208, 208, 208, 208, 208, 208, 208, 207, 207,
207, 207, 207, 207, 207, 206, 206, 206, 206, 206, 206, 206, 206,
205, 205, 205, 205, 205, 205,
77 205, 204, 204, 204, 204, 204, 204, 204, 203, 203, 203, 203, 203,
203, 203, 203, 202, 202, 202, 202, 202, 202, 202, 202, 201, 201,
201, 201, 201, 201, 201, 200,
78 200, 200, 200, 200, 200, 200, 200, 199, 199, 199, 199, 199, 199,
199, 198, 198, 198, 198, 198, 198, 198, 198, 197, 197, 197, 197,
197, 197, 197, 197, 196, 196,
79 196, 196, 196, 196, 196, 196, 195, 195, 195, 195, 195, 195, 195,
195, 194, 194, 194, 194, 194, 194, 194, 193, 193, 193, 193, 193,
193, 193, 193, 192, 192, 192,
80 192, 192, 192, 192, 192, 191, 191, 191, 191, 191, 191, 191, 191,
190, 190, 190, 190, 190, 190, 190, 190, 189, 189, 189, 189, 189,
189, 189, 189, 188, 188, 188,
81 188, 188, 188, 188, 188, 187, 187, 187, 187, 187, 187, 187, 187,
186, 186, 186, 186, 186, 186, 186, 186, 186, 185, 185, 185, 185,
185, 185, 185, 185, 184, 184,
82 184, 184, 184, 184, 184, 184, 183, 183, 183, 183, 183, 183, 183,
183, 182, 182, 182, 182, 182, 182, 182, 182, 182, 181, 181, 181,
181, 181, 181, 181, 181, 180,
83 180, 180, 180, 180, 180, 180, 180, 179, 179, 179, 179, 179, 179,
179, 179, 179, 178, 178, 178, 178, 178, 178, 178, 178, 177, 177,
177, 177, 177, 177, 177, 177,
84 177, 176, 176, 176, 176, 176, 176, 176, 176, 175, 175, 175, 175,
175, 175, 175, 175, 175, 174, 174, 174, 174, 174, 174, 174, 174,
174, 173, 173, 173, 173, 173,
85 173, 173, 173, 172, 172, 172, 172, 172, 172, 172, 172, 172, 171,
171, 171, 171, 171, 171, 171, 171, 171, 170, 170, 170, 170, 170,
170, 170, 170, 170, 169, 169,
86 169, 169, 169, 169, 169, 169, 168, 168, 168, 168, 168, 168, 168,
168, 168, 167, 167, 167, 167, 167, 167, 167, 167, 167, 166, 166,
166, 166, 166, 166, 166, 166,
87 166, 165, 165, 165, 165, 165, 165, 165, 165, 165, 164, 164, 164,
164, 164, 164, 164, 164, 164, 163, 163, 163, 163, 163, 163, 163,
163, 163, 163, 162, 162, 162,
88 162, 162, 162, 162, 162, 162, 161, 161, 161, 161, 161, 161, 161,
161, 161, 160, 160, 160, 160, 160, 160, 160, 160, 160, 159, 159,
159, 159, 159, 159, 159, 159,
89 159, 159, 158, 158, 158, 158, 158, 158, 158, 158, 158, 157, 157,
157, 157, 157, 157, 157, 157, 157, 156, 156, 156, 156, 156, 156,
156, 156, 156, 156, 155, 155,

90 155, 155, 155, 155, 155, 155, 155, 154, 154, 154, 154, 154, 154,
154, 154, 154, 154, 153, 153, 153, 153, 153, 153, 153, 153, 153,
152, 152, 152, 152, 152, 152,
91 152, 152, 152, 152, 151, 151, 151, 151, 151, 151, 151, 151, 151,
151, 150, 150, 150, 150, 150, 150, 150, 150, 150, 150, 149, 149,
149, 149, 149, 149, 149, 149,
92 149, 148, 148, 148, 148, 148, 148, 148, 148, 148, 148, 147, 147,
147, 147, 147, 147, 147, 147, 147, 147, 146, 146, 146, 146, 146,
146, 146, 146, 146, 146, 145,
93 145, 145, 145, 145, 145, 145, 145, 145, 145, 144, 144, 144, 144,
144, 144, 144, 144, 144, 144, 143, 143, 143, 143, 143, 143, 143,
143, 143, 143, 142, 142, 142,
94 142, 142, 142, 142, 142, 142, 142, 141, 141, 141, 141, 141, 141,
141, 141, 141, 141, 141, 140, 140, 140, 140, 140, 140, 140, 140,
140, 140, 139, 139, 139, 139,
95 139, 139, 139, 139, 139, 139, 138, 138, 138, 138, 138, 138, 138,
138, 138, 138, 138, 137, 137, 137, 137, 137, 137, 137, 137, 137,
137, 136, 136, 136, 136, 136,
96 136, 136, 136, 136, 136, 136, 135, 135, 135, 135, 135, 135, 135,
135, 135, 135, 134, 134, 134, 134, 134, 134, 134, 134, 134, 134,
134, 133, 133, 133, 133, 133,
97 133, 133, 133, 133, 133, 133, 132, 132, 132, 132, 132, 132, 132,
132, 132, 132, 131, 131, 131, 131, 131, 131, 131, 131, 131, 131,
131, 130, 130, 130, 130, 130,
98 130, 130, 130, 130, 130, 130, 129, 129, 129, 129, 129, 129, 129,
129, 129, 129, 129, 128, 128, 128, 128, 128, 128, 128, 128, 128,
128, 128, 127, 127, 127, 127,
99 127, 127, 127, 127, 127, 127, 127, 126, 126, 126, 126, 126, 126,
126, 126, 126, 126, 126, 125, 125, 125, 125, 125, 125, 125, 125,
125, 125, 125, 124, 124, 124,
100 124, 124, 124, 124, 124, 124, 124, 124, 124, 123, 123, 123, 123,
123, 123, 123, 123, 123, 123, 123, 122, 122, 122, 122, 122, 122,
122, 122, 122, 122, 122, 121,
101 121, 121, 121, 121, 121, 121, 121, 121, 121, 121, 121, 120, 120,
120, 120, 120, 120, 120, 120, 120, 120, 120, 119, 119, 119, 119,
119, 119, 119, 119, 119, 119,
102 119, 119, 118, 118, 118, 118, 118, 118, 118, 118, 118, 118, 118,
117, 117, 117, 117, 117, 117, 117, 117, 117, 117, 117, 117, 116,
116, 116, 116, 116, 116, 116,
103 116, 116, 116, 116, 116, 115, 115, 115, 115, 115, 115, 115, 115,
115, 115, 115, 115, 114, 114, 114, 114, 114, 114, 114, 114, 114,
114, 114, 114, 113, 113, 113,
104 113, 113, 113, 113, 113, 113, 113, 113, 113, 112, 112, 112, 112,
112, 112, 112, 112, 112, 112, 112, 112, 111, 111, 111, 111, 111,
111, 111, 111, 111, 111, 111,
105 111, 110, 110, 110, 110, 110, 110, 110, 110, 110, 110, 110, 110,
109, 109, 109, 109, 109, 109, 109, 109, 109, 109, 109, 109, 108,
108, 108, 108, 108, 108, 108,
106 108, 108, 108, 108, 108, 108, 107, 107, 107, 107, 107, 107, 107,
107, 107, 107, 107, 107, 106, 106, 106, 106, 106, 106, 106, 106,
106, 106, 106, 106, 106, 105,
107 105, 105, 105, 105, 105, 105, 105, 105, 105, 105, 105, 104, 104,
104, 104, 104, 104, 104, 104, 104, 104, 104, 104, 104, 103, 103,
103, 103, 103, 103, 103, 103,
108 103, 103, 103, 103, 103, 102, 102, 102, 102, 102, 102, 102, 102,
102, 102, 102, 102, 101, 101, 101, 101, 101, 101, 101, 101, 101,
101, 101, 101, 101, 100, 100,

109 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 99, 99,
99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 98, 98,
98, 98, 98, 98, 98, 98,

110 98, 98, 98, 98, 98, 97, 97, 97, 97, 97, 97, 97, 97,
97, 97, 97, 97, 97, 97, 96, 96, 96, 96, 96, 96, 96,
96, 96, 96, 96, 96, 96,

111 95, 95, 95, 95, 95, 95, 95, 95, 95, 95, 95, 95, 95,
94, 94, 94, 94, 94, 94, 94, 94, 94, 94, 94, 94, 94,
94, 93, 93, 93, 93, 93,

112 93, 93, 93, 93, 93, 93, 93, 93, 92, 92, 92, 92, 92,
92, 92, 92, 92, 92, 92, 92, 92, 92, 91, 91, 91, 91,
91, 91, 91, 91, 91, 91,

113 91, 91, 91, 91, 90, 90, 90, 90, 90, 90, 90, 90, 90,
90, 90, 90, 90, 90, 89, 89, 89, 89, 89, 89, 89, 89,
89, 89, 89, 89, 89, 89,

114 88, 88, 88, 88, 88, 88, 88, 88, 88, 88, 88, 88, 88,
88, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
87, 87, 86, 86, 86, 86,

115 86, 86, 86, 86, 86, 86, 86, 86, 86, 86, 85, 85, 85,
85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 84, 84,
84, 84, 84, 84, 84, 84,

116 84, 84, 84, 84, 84, 84, 84, 83, 83, 83, 83, 83, 83,
83, 83, 83, 83, 83, 83, 83, 83, 82, 82, 82, 82, 82,
82, 82, 82, 82, 82, 82,

117 82, 82, 82, 82, 81, 81, 81, 81, 81, 81, 81, 81, 81,
81, 81, 81, 81, 81, 80, 80, 80, 80, 80, 80, 80, 80,
80, 80, 80, 80, 80, 80,

118 80, 79, 79, 79, 79, 79, 79, 79, 79, 79, 79, 79, 79,
79, 79, 79, 78, 78, 78, 78, 78, 78, 78, 78, 78, 78,
78, 78, 78, 78, 78, 77,

119 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77,
77, 76, 76, 76, 76, 76, 76, 76, 76, 76, 76, 76, 76,
76, 76, 76, 75, 75, 75,

120 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75,
74, 74, 74, 74, 74, 74, 74, 74, 74, 74, 74, 74, 74,
74, 74, 73, 73, 73, 73,

121 73, 73, 73, 73, 73, 73, 73, 73, 73, 73, 73, 72, 72,
72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72,
72, 71, 71, 71, 71, 71,

122 71, 71, 71, 71, 71, 71, 71, 71, 71, 71, 71, 70, 70,
70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70,
69, 69, 69, 69, 69, 69,

123 69, 69, 69, 69, 69, 69, 69, 69, 69, 69, 68, 68, 68,
68, 68, 68, 68, 68, 68, 68, 68, 68, 68, 68, 68, 68,
67, 67, 67, 67, 67, 67,

124 67, 67, 67, 67, 67, 67, 67, 67, 67, 67, 66, 66, 66,
66, 66, 66, 66, 66, 66, 66, 66, 66, 66, 66, 66, 66,
66, 65, 65, 65, 65, 65,

125 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 64, 64,
64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64,
64, 63, 63, 63, 63, 63,

126 63, 63, 63, 63, 63, 63, 63, 63, 63, 63, 63, 63, 62,
62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62,
62, 62, 61, 61, 61, 61,

127 61, 61, 61, 61, 61, 61, 61, 61, 61, 61, 61, 61, 61,
60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60,
60, 60, 60, 60, 59, 59,

128 59, 59, 59, 59, 59, 59, 59, 59, 59, 59, 59, 59, 59,
59, 59, 58, 58, 58, 58, 58, 58, 58, 58, 58, 58, 58,
58, 58, 58, 58, 58, 58,
129 57, 57, 57, 57, 57, 57, 57, 57, 57, 57, 57, 57, 57,
57, 57, 57, 57, 56, 56, 56, 56, 56, 56, 56, 56, 56,
56, 56, 56, 56, 56, 56,
130 56, 56, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55,
55, 55, 55, 55, 55, 55, 55, 54, 54, 54, 54, 54, 54,
54, 54, 54, 54, 54, 54,
131 54, 54, 54, 54, 54, 53, 53, 53, 53, 53, 53, 53, 53,
53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 52, 52, 52,
52, 52, 52, 52, 52, 52,
132 52, 52, 52, 52, 52, 52, 52, 52, 51, 51, 51, 51, 51,
51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51,
50, 50, 50, 50, 50, 50,
133 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 49,
49, 49, 49, 49, 49, 49, 49, 49, 49, 49, 49, 49, 49,
49, 49, 49, 49, 48, 48,
134 48, 48, 48, 48, 48, 48, 48, 48, 48, 48, 48, 48, 48,
48, 48, 48, 47, 47, 47, 47, 47, 47, 47, 47, 47, 47,
47, 47, 47, 47, 47, 47,
135 47, 47, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46,
46, 46, 46, 46, 46, 46, 46, 46, 45, 45, 45, 45, 45,
45, 45, 45, 45, 45, 45,
136 45, 45, 45, 45, 45, 45, 45, 44, 44, 44, 44, 44, 44,
44, 44, 44, 44, 44, 44, 44, 44, 44, 44, 44, 44, 43,
43, 43, 43, 43, 43, 43,
137 43, 43, 43, 43, 43, 43, 43, 43, 43, 43, 43, 43, 42,
42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42,
42, 42, 42, 42, 42, 41,
138 41, 41, 41, 41, 41, 41, 41, 41, 41, 41, 41, 41, 41,
41, 41, 41, 41, 41, 40, 40, 40, 40, 40, 40, 40, 40,
40, 40, 40, 40, 40, 40,
139 40, 40, 40, 40, 39, 39, 39, 39, 39, 39, 39, 39, 39,
39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 38, 38, 38,
38, 38, 38, 38, 38, 38,
140 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 37, 37,
37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37,
37, 37, 37, 37, 36, 36,
141 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36,
36, 36, 36, 36, 35, 35, 35, 35, 35, 35, 35, 35, 35,
35, 35, 35, 35, 35, 35,
142 35, 35, 35, 35, 34, 34, 34, 34, 34, 34, 34, 34, 34,
34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 33, 33,
33, 33, 33, 33, 33, 33,
143 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 32,
32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 31,
144 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31,
31, 31, 31, 31, 31, 31, 30, 30, 30, 30, 30, 30, 30,
30, 30, 30, 30, 30, 30,
145 30, 30, 30, 30, 30, 30, 30, 29, 29, 29, 29, 29, 29,
29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29,
29, 28, 28, 28, 28, 28,
146 28, 28, 28, 28, 28, 28, 28, 28, 28, 28, 28, 28, 28,
28, 28, 27, 27, 27, 27, 27, 27, 27, 27, 27, 27, 27,
27, 27, 27, 27, 27, 27,

```
     27,  27,  27,  26,  26,  26,  26,  26,  26,  26,  26,  26,  26,
     26,  26,  26,  26,  26,  26,  26,  26,  26,  26,  26,  25,  25,
     25,  25,  25,  25,  25,  25,
     25,  25,  25,  25,  25,  25,  25,  25,  25,  25,  25,  25,  24,
     24,  24,  24,  24,  24,  24,  24,  24,  24,  24,  24,  24,  24,
     24,  24,  24,  24,  24,  24,
     24,  23,  23,  23,  23,  23,  23,  23,  23,  23,  23,  23,  23,
     23,  23,  23,  23,  23,  23,  23,  23,  22,  22,  22,  22,  22,
     22,  22,  22,  22,  22,  22,
     22,  22,  22,  22,  22,  22,  22,  22,  22,  22,  21,  21,  21,
     21,  21,  21,  21,  21,  21,  21,  21,  21,  21,  21,  21,  21,
     21,  21,  21,  21,  21,  20,
     20,  20,  20,  20,  20,  20,  20,  20,  20,  20,  20,  20,  20,
     20,  20,  20,  20,  20,  20,  20,  19,  19,  19,  19,  19,  19,
     19,  19,  19,  19,  19, };
static const uint16_t cpLUT_2[4096] = {
      0, 359, 359, 358, 357, 356, 356, 355, 354, 354, 353, 352, 352,
      351, 350, 349, 349, 348, 347, 347, 346, 345, 345, 344, 343, 342,
      342, 341, 340, 340, 339, 338,
    338, 337, 336, 336, 335, 334, 333, 333, 332, 331, 331, 330, 329,
    329, 328, 327, 327, 326, 325, 325, 324, 323, 323, 322, 321, 321,
    320, 319, 319, 318, 317, 317,
    316, 315, 315, 314, 313, 313, 312, 311, 311, 310, 309, 309, 308,
    307, 307, 306, 305, 305, 304, 303, 303, 302, 301, 301, 300, 299,
    299, 298, 297, 297, 296, 295,
    295, 294, 294, 293, 292, 292, 291, 290, 290, 289, 288, 288, 287,
    286, 286, 285, 285, 284, 283, 283, 282, 281, 281, 280, 279, 279,
    278, 278, 277, 276, 276, 275,
    274, 274, 273, 273, 272, 271, 271, 270, 269, 269, 268, 268, 267,
    266, 266, 265, 264, 264, 263, 263, 262, 261, 261, 260, 260, 259,
    258, 258, 257, 256, 256, 255,
    255, 254, 253, 253, 252, 252, 251, 250, 250, 249, 249, 248, 247,
    247, 246, 246, 245, 244, 244, 243, 243, 242, 241, 241, 240, 240,
    239, 238, 238, 237, 237, 236,
    235, 235, 234, 234, 233, 232, 232, 231, 231, 230, 230, 229, 228,
    228, 227, 227, 226, 225, 225, 224, 224, 223, 223, 222, 221, 221,
    220, 220, 219, 218, 218, 217,
    217, 216, 216, 215, 214, 214, 213, 213, 212, 212, 211, 210, 210,
    209, 209, 208, 208, 207, 207, 206, 205, 205, 204, 204, 203, 203,
    202, 201, 201, 200, 200, 199,
    199, 198, 198, 197, 196, 196, 195, 195, 194, 194, 193, 193, 192,
    191, 191, 190, 190, 189, 189, 188, 188, 187, 187, 186, 185, 185,
    184, 184, 183, 183, 182, 182,
    181, 181, 180, 180, 179, 178, 178, 177, 177, 176, 176, 175, 175,
    174, 174, 173, 173, 172, 171, 171, 170, 170, 169, 169, 168, 168,
    167, 167, 166, 166, 165, 165,
    164, 164, 163, 163, 162, 162, 161, 160, 160, 159, 159, 158, 158,
    157, 157, 156, 156, 155, 155, 154, 154, 153, 153, 152, 152, 151,
    151, 150, 150, 149, 149, 148,
    148, 147, 147, 146, 146, 145, 145, 144, 144, 143, 143, 142, 142,
    141, 141, 140, 140, 139, 139, 138, 138, 137, 137, 136, 136, 135,
    135, 134, 134, 133, 133, 132,
    132, 131, 131, 130, 130, 129, 129, 128, 128, 127, 127, 126, 126,
    125, 125, 124, 124, 123, 123, 122, 122, 121, 121, 120, 120, 119,
    119, 118, 118, 117, 117, 117,
```

166 116, 116, 115, 115, 114, 114, 113, 113, 112, 112, 111, 111, 110,
110, 109, 109, 108, 108, 108, 107, 107, 106, 106, 105, 105, 104,
104, 103, 103, 102, 102, 101,
167 101, 101, 100, 100, 99, 99, 98, 98, 97, 97, 96, 96, 95,
95, 95, 94, 94, 93, 93, 92, 92, 91, 91, 90, 90, 90,
89, 89, 88, 88, 87, 87,
168 86, 86, 85, 85, 85, 84, 84, 83, 83, 82, 82, 81, 81,
81, 80, 80, 79, 79, 78, 78, 77, 77, 77, 76, 76, 75,
75, 74, 74, 73, 73, 73,
169 72, 72, 71, 71, 70, 70, 70, 69, 69, 68, 68, 67, 67,
66, 66, 66, 65, 65, 64, 64, 63, 63, 63, 62, 62, 61,
61, 60, 60, 60, 59, 59,
170 58, 58, 58, 57, 57, 56, 56, 55, 55, 55, 54, 54, 53,
53, 52, 52, 52, 51, 51, 50, 50, 50, 49, 49, 48, 48,
47, 47, 47, 46, 46, 45,
171 45, 45, 44, 44, 43, 43, 43, 42, 42, 41, 41, 40, 40,
40, 39, 39, 38, 38, 38, 37, 37, 36, 36, 36, 35, 35,
34, 34, 34, 33, 33, 32,
172 32, 32, 31, 31, 30, 30, 30, 29, 29, 28, 28, 28, 27,
27, 26, 26, 26, 25, 25, 24, 24, 24, 23, 23, 22, 22,
22, 21, 21, 21, 20, 20,
173 19, 19, 19, 18, 18, 17, 17, 17, 16, 16, 15, 15, 15,
14, 14, 14, 13, 13, 12, 12, 12, 11, 11, 11, 10, 10,
9, 9, 9, 8, 8, 7,
174 7, 7, 6, 6, 6, 5, 5, 4, 4, 4, 3, 3, 3,
2, 2, 1, 1, 1, 0, 360, 360, 359, 359, 359, 358, 358,
357, 357, 357, 356, 356, 356,
175 355, 355, 354, 354, 354, 353, 353, 353, 352, 352, 352, 351, 351,
350, 350, 350, 349, 349, 349, 348, 348, 348, 347, 347, 346, 346,
346, 345, 345, 345, 344, 344,
176 344, 343, 343, 343, 342, 342, 341, 341, 341, 340, 340, 340, 339,
339, 339, 338, 338, 338, 337, 337, 337, 336, 336, 335, 335, 335,
334, 334, 334, 333, 333, 333,
177 332, 332, 332, 331, 331, 331, 330, 330, 330, 329, 329, 329, 328,
328, 328, 327, 327, 327, 326, 326, 326, 325, 325, 324, 324, 324,
323, 323, 323, 322, 322, 322,
178 321, 321, 321, 320, 320, 320, 319, 319, 319, 318, 318, 318, 317,
317, 317, 316, 316, 316, 315, 315, 315, 314, 314, 314, 313, 313,
313, 312, 312, 312, 311, 311,
179 311, 310, 310, 310, 310, 309, 309, 309, 308, 308, 308, 307, 307,
307, 306, 306, 306, 305, 305, 305, 304, 304, 304, 303, 303, 303,
302, 302, 302, 301, 301, 301,
180 300, 300, 300, 300, 299, 299, 299, 298, 298, 298, 297, 297, 297,
296, 296, 296, 295, 295, 295, 294, 294, 294, 294, 293, 293, 293,
292, 292, 292, 291, 291, 291,
181 290, 290, 290, 289, 289, 289, 289, 288, 288, 288, 287, 287, 287,
286, 286, 286, 285, 285, 285, 285, 284, 284, 284, 283, 283, 283,
282, 282, 282, 282, 281, 281,
182 281, 280, 280, 280, 279, 279, 279, 279, 278, 278, 278, 277, 277,
277, 276, 276, 276, 276, 275, 275, 275, 274, 274, 274, 273, 273,
273, 273, 272, 272, 272, 271,
183 271, 271, 271, 270, 270, 270, 269, 269, 269, 268, 268, 268, 268,
267, 267, 267, 266, 266, 266, 266, 265, 265, 265, 264, 264, 264,
264, 263, 263, 263, 262, 262,
184 262, 262, 261, 261, 261, 260, 260, 260, 260, 259, 259, 259, 258,
258, 258, 258, 257, 257, 257, 256, 256, 256, 256, 255, 255, 255,
254, 254, 254, 254, 253, 253,

185 253, 253, 252, 252, 252, 251, 251, 251, 251, 250, 250, 250, 249,
249, 249, 249, 248, 248, 248, 248, 247, 247, 247, 246, 246, 246,
246, 245, 245, 245, 245, 244,
186 244, 244, 243, 243, 243, 243, 242, 242, 242, 242, 241, 241, 241,
241, 240, 240, 240, 239, 239, 239, 239, 238, 238, 238, 238, 237,
237, 237, 237, 236, 236, 236,
187 235, 235, 235, 235, 234, 234, 234, 234, 233, 233, 233, 233, 232,
232, 232, 232, 231, 231, 231, 230, 230, 230, 230, 229, 229, 229,
229, 228, 228, 228, 228, 227,
188 227, 227, 227, 226, 226, 226, 226, 225, 225, 225, 225, 224, 224,
224, 224, 223, 223, 223, 223, 222, 222, 222, 222, 221, 221, 221,
220, 220, 220, 220, 219, 219,
189 219, 219, 218, 218, 218, 218, 217, 217, 217, 217, 216, 216, 216,
216, 215, 215, 215, 215, 214, 214, 214, 214, 214, 213, 213, 213,
213, 212, 212, 212, 212, 211,
190 211, 211, 211, 210, 210, 210, 210, 209, 209, 209, 209, 208, 208,
208, 208, 207, 207, 207, 207, 206, 206, 206, 206, 205, 205, 205,
205, 204, 204, 204, 204, 204,
191 203, 203, 203, 203, 202, 202, 202, 202, 201, 201, 201, 201, 200,
200, 200, 200, 199, 199, 199, 199, 199, 198, 198, 198, 198, 197,
197, 197, 197, 196, 196, 196,
192 196, 196, 195, 195, 195, 195, 194, 194, 194, 194, 193, 193, 193,
193, 192, 192, 192, 192, 192, 191, 191, 191, 191, 190, 190, 190,
190, 190, 189, 189, 189, 189,
193 188, 188, 188, 188, 187, 187, 187, 187, 187, 186, 186, 186, 186,
185, 185, 185, 185, 185, 184, 184, 184, 184, 183, 183, 183, 183,
183, 182, 182, 182, 182, 181,
194 181, 181, 181, 181, 180, 180, 180, 180, 179, 179, 179, 179, 179,
178, 178, 178, 178, 177, 177, 177, 177, 177, 176, 176, 176, 176,
175, 175, 175, 175, 175, 174,
195 174, 174, 174, 173, 173, 173, 173, 173, 172, 172, 172, 172, 172,
171, 171, 171, 171, 170, 170, 170, 170, 170, 169, 169, 169, 169,
169, 168, 168, 168, 168, 167,
196 167, 167, 167, 167, 166, 166, 166, 166, 166, 165, 165, 165, 165,
165, 164, 164, 164, 164, 163, 163, 163, 163, 163, 162, 162, 162,
162, 162, 161, 161, 161, 161,
197 161, 160, 160, 160, 160, 160, 159, 159, 159, 159, 158, 158, 158,
158, 158, 157, 157, 157, 157, 157, 156, 156, 156, 156, 156, 155,
155, 155, 155, 155, 154, 154,
198 154, 154, 154, 153, 153, 153, 153, 153, 152, 152, 152, 152, 152,
151, 151, 151, 151, 151, 150, 150, 150, 150, 150, 149, 149, 149,
149, 149, 148, 148, 148, 148,
199 148, 147, 147, 147, 147, 147, 146, 146, 146, 146, 146, 145, 145,
145, 145, 145, 144, 144, 144, 144, 144, 143, 143, 143, 143, 143,
142, 142, 142, 142, 142, 141,
200 141, 141, 141, 141, 140, 140, 140, 140, 140, 140, 139, 139, 139,
139, 139, 138, 138, 138, 138, 138, 137, 137, 137, 137, 137, 136,
136, 136, 136, 136, 135, 135,
201 135, 135, 135, 135, 134, 134, 134, 134, 134, 133, 133, 133, 133,
133, 132, 132, 132, 132, 132, 132, 131, 131, 131, 131, 131, 130,
130, 130, 130, 130, 129, 129,
202 129, 129, 129, 129, 128, 128, 128, 128, 128, 127, 127, 127, 127,
127, 127, 126, 126, 126, 126, 126, 125, 125, 125, 125, 125, 124,
124, 124, 124, 124, 124, 123,
203 123, 123, 123, 123, 122, 122, 122, 122, 122, 122, 121, 121, 121,
121, 121, 120, 120, 120, 120, 120, 120, 119, 119, 119, 119, 119,
118, 118, 118, 118, 118, 118,

204 117, 117, 117, 117, 117, 117, 116, 116, 116, 116, 116, 115, 115,
115, 115, 115, 115, 114, 114, 114, 114, 114, 114, 113, 113, 113,
113, 113, 112, 112, 112, 112,
205 112, 112, 111, 111, 111, 111, 111, 111, 110, 110, 110, 110, 110,
109, 109, 109, 109, 109, 109, 108, 108, 108, 108, 108, 108, 107,
107, 107, 107, 107, 107, 106,
206 106, 106, 106, 106, 106, 105, 105, 105, 105, 105, 105, 104, 104,
104, 104, 104, 103, 103, 103, 103, 103, 103, 102, 102, 102, 102,
102, 102, 101, 101, 101, 101,
207 101, 101, 100, 100, 100, 100, 100, 100, 99, 99, 99, 99, 99,
99, 98, 98, 98, 98, 98, 98, 97, 97, 97, 97, 97, 97,
96, 96, 96, 96, 96, 96,
208 95, 95, 95, 95, 95, 95, 94, 94, 94, 94, 94, 94, 93,
93, 93, 93, 93, 93, 92, 92, 92, 92, 92, 92, 92, 91,
91, 91, 91, 91, 91, 90,
209 90, 90, 90, 90, 90, 89, 89, 89, 89, 89, 89, 88, 88,
88, 88, 88, 88, 87, 87, 87, 87, 87, 87, 87, 86, 86,
86, 86, 86, 86, 85, 85,
210 85, 85, 85, 85, 84, 84, 84, 84, 84, 84, 83, 83, 83,
83, 83, 83, 83, 82, 82, 82, 82, 82, 82, 81, 81, 81,
81, 81, 81, 80, 80, 80,
211 80, 80, 80, 80, 79, 79, 79, 79, 79, 79, 78, 78, 78,
78, 78, 78, 78, 77, 77, 77, 77, 77, 77, 76, 76, 76,
76, 76, 76, 76, 75, 75,
212 75, 75, 75, 75, 74, 74, 74, 74, 74, 74, 74, 73, 73,
73, 73, 73, 73, 72, 72, 72, 72, 72, 72, 72, 71, 71,
71, 71, 71, 71, 71, 70,
213 70, 70, 70, 70, 70, 69, 69, 69, 69, 69, 69, 69, 68,
68, 68, 68, 68, 68, 68, 67, 67, 67, 67, 67, 67, 66,
66, 66, 66, 66, 66, 66,
214 65, 65, 65, 65, 65, 65, 65, 64, 64, 64, 64, 64, 64,
64, 63, 63, 63, 63, 63, 63, 63, 62, 62, 62, 62, 62,
62, 61, 61, 61, 61, 61,
215 61, 61, 60, 60, 60, 60, 60, 60, 60, 59, 59, 59, 59,
59, 59, 59, 58, 58, 58, 58, 58, 58, 58, 57, 57, 57,
57, 57, 57, 57, 56, 56,
216 56, 56, 56, 56, 56, 55, 55, 55, 55, 55, 55, 55, 54,
54, 54, 54, 54, 54, 54, 53, 53, 53, 53, 53, 53, 53,
52, 52, 52, 52, 52, 52,
217 52, 51, 51, 51, 51, 51, 51, 51, 50, 50, 50, 50, 50,
50, 50, 50, 49, 49, 49, 49, 49, 49, 49, 48, 48, 48,
48, 48, 48, 48, 47, 47,
218 47, 47, 47, 47, 47, 46, 46, 46, 46, 46, 46, 46, 46,
45, 45, 45, 45, 45, 45, 45, 44, 44, 44, 44, 44, 44,
44, 43, 43, 43, 43, 43,
219 43, 43, 43, 42, 42, 42, 42, 42, 42, 42, 41, 41, 41,
41, 41, 41, 41, 40, 40, 40, 40, 40, 40, 40, 40, 39,
39, 39, 39, 39, 39, 39,
220 38, 38, 38, 38, 38, 38, 38, 38, 37, 37, 37, 37, 37,
37, 37, 36, 36, 36, 36, 36, 36, 36, 36, 35, 35, 35,
35, 35, 35, 35, 35, 34,
221 34, 34, 34, 34, 34, 34, 33, 33, 33, 33, 33, 33, 33,
33, 32, 32, 32, 32, 32, 32, 32, 31, 31, 31, 31, 31,
31, 31, 31, 30, 30, 30,
222 30, 30, 30, 30, 30, 29, 29, 29, 29, 29, 29, 29, 29,
28, 28, 28, 28, 28, 28, 28, 28, 27, 27, 27, 27, 27,
27, 27, 26, 26, 26, 26,

223 26, 26, 26, 26, 25, 25, 25, 25, 25, 25, 25, 25, 24,
24, 24, 24, 24, 24, 24, 24, 23, 23, 23, 23, 23, 23,
23, 23, 22, 22, 22, 22,
224 22, 22, 22, 22, 21, 21, 21, 21, 21, 21, 21, 21, 20,
20, 20, 20, 20, 20, 20, 20, 19, 19, 19, 19, 19, 19,
19, 19, 18, 18, 18, 18,
225 18, 18, 18, 18, 17, 17, 17, 17, 17, 17, 17, 17, 16,
16, 16, 16, 16, 16, 16, 16, 16, 15, 15, 15, 15, 15,
15, 15, 15, 14, 14, 14,
226 14, 14, 14, 14, 14, 13, 13, 13, 13, 13, 13, 13, 13,
12, 12, 12, 12, 12, 12, 12, 12, 12, 11, 11, 11, 11,
11, 11, 11, 11, 10, 10,
227 10, 10, 10, 10, 10, 10, 9, 9, 9, 9, 9, 9, 9,
9, 9, 8, 8, 8, 8, 8, 8, 8, 8, 7, 7, 7,
7, 7, 7, 7, 7, 7,
228 6, 6, 6, 6, 6, 6, 6, 6, 5, 5, 5, 5, 5,
5, 5, 5, 5, 4, 4, 4, 4, 4, 4, 4, 4, 3,
3, 3, 3, 3, 3, 3,
229 3, 3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1,
1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 360, 360,
360, 360, 359, 359, 359, 359,
230 359, 359, 359, 359, 359, 358, 358, 358, 358, 358, 358, 358, 358,
358, 357, 357, 357, 357, 357, 357, 357, 357, 357, 356, 356, 356,
356, 356, 356, 356, 356, 356,
231 355, 355, 355, 355, 355, 355, 355, 355, 354, 354, 354, 354, 354,
354, 354, 354, 354, 353, 353, 353, 353, 353, 353, 353, 353, 353,
352, 352, 352, 352, 352, 352,
232 352, 352, 352, 351, 351, 351, 351, 351, 351, 351, 351, 351, 351,
350, 350, 350, 350, 350, 350, 350, 350, 350, 349, 349, 349, 349,
349, 349, 349, 349, 349, 348,
233 348, 348, 348, 348, 348, 348, 348, 348, 347, 347, 347, 347, 347,
347, 347, 347, 347, 346, 346, 346, 346, 346, 346, 346, 346, 346,
346, 345, 345, 345, 345, 345,
234 345, 345, 345, 345, 344, 344, 344, 344, 344, 344, 344, 344, 344,
343, 343, 343, 343, 343, 343, 343, 343, 343, 343, 342, 342, 342,
342, 342, 342, 342, 342, 342,
235 341, 341, 341, 341, 341, 341, 341, 341, 341, 340, 340, 340, 340,
340, 340, 340, 340, 340, 340, 339, 339, 339, 339, 339, 339, 339,
339, 339, 339, 338, 338, 338,
236 338, 338, 338, 338, 338, 338, 337, 337, 337, 337, 337, 337, 337,
337, 337, 337, 336, 336, 336, 336, 336, 336, 336, 336, 336, 335,
335, 335, 335, 335, 335, 335,
237 335, 335, 335, 334, 334, 334, 334, 334, 334, 334, 334, 334, 334,
333, 333, 333, 333, 333, 333, 333, 333, 333, 333, 332, 332, 332,
332, 332, 332, 332, 332, 332,
238 332, 331, 331, 331, 331, 331, 331, 331, 331, 331, 331, 330, 330,
330, 330, 330, 330, 330, 330, 330, 329, 329, 329, 329, 329, 329,
329, 329, 329, 329, 328, 328,
239 328, 328, 328, 328, 328, 328, 328, 328, 327, 327, 327, 327, 327,
327, 327, 327, 327, 327, 327, 326, 326, 326, 326, 326, 326, 326,
326, 326, 326, 325, 325, 325,
240 325, 325, 325, 325, 325, 325, 325, 324, 324, 324, 324, 324, 324,
324, 324, 324, 324, 323, 323, 323, 323, 323, 323, 323, 323, 323,
323, 322, 322, 322, 322, 322,
241 322, 322, 322, 322, 322, 322, 321, 321, 321, 321, 321, 321, 321,
321, 321, 321, 320, 320, 320, 320, 320, 320, 320, 320, 320, 320,
319, 319, 319, 319, 319, 319,

242 319, 319, 319, 319, 319, 318, 318, 318, 318, 318, 318, 318, 318,
318, 318, 317, 317, 317, 317, 317, 317, 317, 317, 317, 317, 317,
316, 316, 316, 316, 316, 316,

243 316, 316, 316, 316, 315, 315, 315, 315, 315, 315, 315, 315, 315,
315, 315, 314, 314, 314, 314, 314, 314, 314, 314, 314, 314, 314,
313, 313, 313, 313, 313, 313,

244 313, 313, 313, 313, 313, 312, 312, 312, 312, 312, 312, 312, 312,
312, 312, 311, 311, 311, 311, 311, 311, 311, 311, 311, 311, 311,
310, 310, 310, 310, 310, 310,

245 310, 310, 310, 310, 310, 309, 309, 309, 309, 309, 309, 309, 309,
309, 309, 309, 308, 308, 308, 308, 308, 308, 308, 308, 308, 308,
308, 307, 307, 307, 307, 307,

246 307, 307, 307, 307, 307, 307, 306, 306, 306, 306, 306, 306, 306,
306, 306, 306, 306, 305, 305, 305, 305, 305, 305, 305, 305, 305,
305, 305, 304, 304, 304, 304,

247 304, 304, 304, 304, 304, 304, 304, 304, 303, 303, 303, 303, 303,
303, 303, 303, 303, 303, 303, 302, 302, 302, 302, 302, 302, 302,
302, 302, 302, 302, 301, 301,

248 301, 301, 301, 301, 301, 301, 301, 301, 301, 300, 300, 300, 300,
300, 300, 300, 300, 300, 300, 300, 300, 299, 299, 299, 299, 299,
299, 299, 299, 299, 299, 299,

249 298, 298, 298, 298, 298, 298, 298, 298, 298, 298, 298, 298, 297,
297, 297, 297, 297, 297, 297, 297, 297, 297, 297, 296, 296, 296,
296, 296, 296, 296, 296, 296,

250 296, 296, 296, 295, 295, 295, 295, 295, 295, 295, 295, 295, 295,
295, 295, 294, 294, 294, 294, 294, 294, 294, 294, 294, 294, 294,
293, 293, 293, 293, 293, 293,

251 293, 293, 293, 293, 293, 293, 292, 292, 292, 292, 292, 292, 292,
292, 292, 292, 292, 292, 291, 291, 291, 291, 291, 291, 291, 291,
291, 291, 291, 291, 290, 290,

252 290, 290, 290, 290, 290, 290, 290, 290, 290, 290, 289, 289, 289,
289, 289, 289, 289, 289, 289, 289, 289, 289, 288, 288, 288, 288,
288, 288, 288, 288, 288, 288,

253 288, 288, 287, 287, 287, 287, 287, 287, 287, 287, 287, 287, 287,
287, 286, 286, 286, 286, 286, 286, 286, 286, 286, 286, 286, 286,
285, 285, 285, 285, 285, 285,

254 285, 285, 285, 285, 285, 285, 285, 284, 284, 284, 284, 284, 284,
284, 284, 284, 284, 284, 284, 283, 283, 283, 283, 283, 283, 283,
283, 283, 283, 283, 283, 282,

255 282, 282, 282, 282, 282, 282, 282, 282, 282, 282, 282, 282, 281,
281, 281, 281, 281, 281, 281, 281, 281, 281, 281, 281, 280, 280,
280, 280, 280, 280, 280, 280,

256 280, 280, 280, 280, 280, 279, 279, 279, 279, 279, 279, 279, 279,
279, 279, 279, 279, 278, 278, 278, 278, 278, 278, 278, 278, 278,
278, 278, 278, 278, 277, 277,

257 277, 277, 277, 277, 277, 277, 277, 277, 277, 277, 276, 276, 276,
276, 276, 276, 276, 276, 276, 276, 276, 276, 276, 275, 275, 275,
275, 275, 275, 275, 275, 275,

258 275, 275, 275, 275, 274, 274, 274, 274, 274, 274, 274, 274, 274,
274, 274, 274, 274, 273, 273, 273, 273, 273, 273, 273, 273, 273,
273, 273, 273, 273, 272, 272,

259 272, 272, 272, 272, 272, 272, 272, 272, 272, 272, 272, 271, 271,
271, 271, 271, 271, 271, 271, 271, 271, 271, 271, 271, 270, 270,
270, 270, 270, 270, 270, 270,

260 270, 270, 270, 270, 270, 269, 269, 269, 269, 269, 269, 269, 269,
269, 269, 269, 269, 269, 268, 268, 268, 268, 268, 268, 268, 268,
268, 268, 268, 268, 268, 267,

261 267, 267, 267, 267, 267, 267, 267, 267, 267, 267, 267, 267, 267,
266, 266, 266, 266, 266, 266, 266, 266, 266, 266, 266, 266, 266,
265, 265, 265, 265, 265, 265,
262 265, 265, 265, 265, 265, 265, 265, 264, 264, 264, 264, 264, 264,
264, 264, 264, 264, 264, 264, 264, 264, 263, 263, 263, 263, 263,
263, 263, 263, 263, 263, 263,
263 263, 263, 262, 262, 262, 262, 262, 262, 262, 262, 262, 262, 262,
262, 262, 262, 261, 261, 261, 261, 261, 261, 261, 261, 261, 261,
261, 261, 261, 260, 260, 260,
264 260, 260, 260, 260, 260, 260, 260, 260, 260, 260, 260, 259, 259,
259, 259, 259, 259, 259, 259, 259, 259, 259, 259, 259, 259, 258,
258, 258, 258, 258, 258, 258,
265 258, 258, 258, 258, 258, 258, 258, 257, 257, 257, 257, 257, 257,
257, 257, 257, 257, 257, 257, 257, 256, 256, 256, 256, 256, 256,
256, 256, 256, 256, 256, 256,
266 256, 256, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 254, 254, 254, 254, 254, 254, 254, 254, 254, 254,
254, 254, 254, 254, 253, 253,
267 253, 253, 253, 253, 253, 253, 253, 253, 253, 253, 253, 253, 252,
252, 252, 252, 252, 252, 252, 252, 252, 252, 252, 252, 252, 252,
252, 251, 251, 251, 251, 251,
268 251, 251, 251, 251, 251, 251, 251, 251, 251, 250, 250, 250, 250,
250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 249, 249, 249,
249, 249, 249, 249, 249, 249,
269 249, 249, 249, 249, 249, 248, 248, 248, 248, 248, 248, 248, 248,
248, 248, 248, 248, 248, 248, 248, 247, 247, 247, 247, 247, 247,
247, 247, 247, 247, 247, 247,
270 247, 247, 246, 246, 246, 246, 246, 246, 246, 246, 246, 246, 246,
246, 246, 246, 246, 245, 245, 245, 245, 245, 245, 245, 245, 245,
245, 245, 245, 245, 245, 244,
271 244, 244, 244, 244, 244, 244, 244, 244, 244, 244, 244, 244, 244,
244, 243, 243, 243, 243, 243, 243, 243, 243, 243, 243, 243, 243,
243, 243, 242, 242, 242, 242,
272 242, 242, 242, 242, 242, 242, 242, 242, 242, 242, 242, 241, 241,
241, 241, 241, 241, 241, 241, 241, 241, 241, 241, 241, 241, 241,
240, 240, 240, 240, 240, 240,
273 240, 240, 240, 240, 240, 240, 240, 240, 240, 239, 239, 239, 239,
239, 239, 239, 239, 239, 239, 239, 239, 239, 239, 239, 238, 238,
238, 238, 238, 238, 238, 238,
274 238, 238, 238, 238, 238, 238, 238, 237, 237, 237, 237, 237, 237,
237, 237, 237, 237, 237, 237, 237, 237, 237, 236, 236, 236, 236,
236, 236, 236, 236, 236, 236,
275 236, 236, 236, 236, 236, 235, 235, 235, 235, 235, 235, 235, 235,
235, 235, 235, 235, 235, 235, 235, 234, 234, 234, 234, 234, 234,
234, 234, 234, 234, 234, 234,
276 234, 234, 234, 233, 233, 233, 233, 233, 233, 233, 233, 233, 233,
233, 233, 233, 233, 233, 232, 232, 232, 232, 232, 232, 232, 232,
232, 232, 232, 232, 232, 232,
277 232, 232, 231, 231, 231, 231, 231, 231, 231, 231, 231, 231, 231,
231, 231, 231, 231, 230, 230, 230, 230, 230, 230, 230, 230, 230,
230, 230, 230, 230, 230, 230,
278 229, 229, 229, 229, 229, 229, 229, 229, 229, 229, 229, 229, 229,
229, 229, 229, 228, 228, 228, 228, 228, 228, 228, 228, 228, 228,
228, 228, 228, 228, 228, 227,
279 227, 227, 227, 227, 227, 227, 227, 227, 227, 227, 227, 227, 227,
227, 227, 226, 226, 226, 226, 226, 226, 226, 226, 226, 226, 226,
226, 226, 226, 226, 226, 225,

```
        225, 225, 225, 225, 225, 225, 225, 225, 225, 225, 225, 225, 225,
        225, 225, 224, 224, 224, 224, 224, 224, 224, 224, 224, 224, 224,
        224, 224, 224, 224, 224, };
    static const uint16_t cpLUT_3[4096] = {
          0, 359, 359, 358, 357, 357, 356, 355, 355, 354, 353, 353, 352,
          351, 351, 350, 349, 348, 348, 347, 346, 346, 345, 344, 344, 343,
          342, 342, 341, 340, 340, 339,
        338, 338, 337, 336, 336, 335, 335, 334, 333, 333, 332, 331, 331,
        330, 329, 329, 328, 327, 327, 326, 325, 325, 324, 323, 323, 322,
        321, 321, 320, 319, 319, 318,
        318, 317, 316, 316, 315, 314, 314, 313, 312, 312, 311, 310, 310,
        309, 308, 308, 307, 307, 306, 305, 305, 304, 303, 303, 302, 301,
        301, 300, 300, 299, 298, 298,
        297, 296, 296, 295, 295, 294, 293, 293, 292, 291, 291, 290, 290,
        289, 288, 288, 287, 286, 286, 285, 285, 284, 283, 283, 282, 281,
        281, 280, 280, 279, 278, 278,
        277, 276, 276, 275, 275, 274, 273, 273, 272, 272, 271, 270, 270,
        269, 269, 268, 267, 267, 266, 266, 265, 264, 264, 263, 262, 262,
        261, 261, 260, 259, 259, 258,
        258, 257, 256, 256, 255, 255, 254, 253, 253, 252, 252, 251, 251,
        250, 249, 249, 248, 248, 247, 246, 246, 245, 245, 244, 243, 243,
        242, 242, 241, 240, 240, 239,
        239, 238, 238, 237, 236, 236, 235, 235, 234, 233, 233, 232, 232,
        231, 231, 230, 229, 229, 228, 228, 227, 227, 226, 225, 225, 224,
        224, 223, 223, 222, 221, 221,
        220, 220, 219, 219, 218, 217, 217, 216, 216, 215, 215, 214, 213,
        213, 212, 212, 211, 211, 210, 209, 209, 208, 208, 207, 207, 206,
        206, 205, 204, 204, 203, 203,
        202, 202, 201, 201, 200, 199, 199, 198, 198, 197, 197, 196, 196,
        195, 194, 194, 193, 193, 192, 192, 191, 191, 190, 190, 189, 188,
        188, 187, 187, 186, 186, 185,
        185, 184, 184, 183, 183, 182, 181, 181, 180, 180, 179, 179, 178,
        178, 177, 177, 176, 176, 175, 174, 174, 173, 173, 172, 172, 171,
        171, 170, 170, 169, 169, 168,
        168, 167, 166, 166, 165, 165, 164, 164, 163, 163, 162, 162, 161,
        161, 160, 160, 159, 159, 158, 158, 157, 157, 156, 156, 155, 154,
        154, 153, 153, 152, 152, 151,
        151, 150, 150, 149, 149, 148, 148, 147, 147, 146, 146, 145, 145,
        144, 144, 143, 143, 142, 142, 141, 141, 140, 140, 139, 139, 138,
        138, 137, 137, 136, 136, 135,
        135, 134, 134, 133, 133, 132, 132, 131, 131, 130, 130, 129, 129,
        128, 128, 127, 127, 126, 126, 125, 125, 124, 124, 123, 123, 122,
        122, 121, 121, 120, 120, 119,
        119, 118, 118, 117, 117, 116, 116, 115, 115, 114, 114, 113, 113,
        112, 112, 111, 111, 110, 110, 109, 109, 109, 108, 108, 107, 107,
        106, 106, 105, 105, 104, 104,
        103, 103, 102, 102, 101, 101, 100, 100,  99,  99,  98,  98,  98,
        97,  97,  96,  96,  95,  95,  94,  94,  93,  93,  92,  92,  91,
        91,  90,  90,  90,  89,  89,
         88,  88,  87,  87,  86,  86,  85,  85,  84,  84,  84,  83,  83,
         82,  82,  81,  81,  80,  80,  79,  79,  78,  78,  78,  77,  77,
         76,  76,  75,  75,  74,  74,
         73,  73,  73,  72,  72,  71,  71,  70,  70,  69,  69,  68,  68,
         68,  67,  67,  66,  66,  65,  65,  64,  64,  64,  63,  63,  62,
         62,  61,  61,  60,  60,  60,
```

299 59, 59, 58, 58, 57, 57, 56, 56, 56, 55, 55, 54, 54,
53, 53, 52, 52, 52, 51, 51, 50, 50, 49, 49, 49, 48,
48, 47, 47, 46, 46, 46,
300 45, 45, 44, 44, 43, 43, 43, 42, 42, 41, 41, 40, 40,
40, 39, 39, 38, 38, 37, 37, 37, 36, 36, 35, 35, 34,
34, 34, 33, 33, 32, 32,
301 31, 31, 31, 30, 30, 29, 29, 28, 28, 28, 27, 27, 26,
26, 26, 25, 25, 24, 24, 23, 23, 23, 22, 22, 21, 21,
21, 20, 20, 19, 19, 19,
302 18, 18, 17, 17, 16, 16, 16, 15, 15, 14, 14, 14, 13,
13, 12, 12, 12, 11, 11, 10, 10, 10, 9, 9, 8, 8,
8, 7, 7, 6, 6, 6,
303 5, 5, 4, 4, 4, 3, 3, 2, 2, 2, 1, 1, 0,
360, 360, 359, 359, 358, 358, 358, 357, 357, 356, 356, 356, 355,
355, 354, 354, 354, 353, 353,
304 352, 352, 352, 351, 351, 350, 350, 350, 349, 349, 349, 348, 348,
347, 347, 347, 346, 346, 345, 345, 345, 344, 344, 344, 343, 343,
342, 342, 342, 341, 341, 340,
305 340, 340, 339, 339, 339, 338, 338, 337, 337, 337, 336, 336, 335,
335, 335, 334, 334, 334, 333, 333, 332, 332, 332, 331, 331, 331,
330, 330, 329, 329, 329, 328,
306 328, 328, 327, 327, 326, 326, 326, 325, 325, 325, 324, 324, 324,
323, 323, 322, 322, 322, 321, 321, 321, 320, 320, 319, 319, 319,
318, 318, 318, 317, 317, 317,
307 316, 316, 315, 315, 315, 314, 314, 314, 313, 313, 313, 312, 312,
311, 311, 311, 310, 310, 310, 309, 309, 309, 308, 308, 308, 307,
307, 306, 306, 306, 305, 305,
308 305, 304, 304, 304, 303, 303, 303, 302, 302, 301, 301, 301, 300,
300, 300, 299, 299, 299, 298, 298, 298, 297, 297, 297, 296, 296,
296, 295, 295, 294, 294, 294,
309 293, 293, 293, 292, 292, 292, 291, 291, 291, 290, 290, 290, 289,
289, 289, 288, 288, 288, 287, 287, 287, 286, 286, 286, 285, 285,
284, 284, 284, 283, 283, 283,
310 282, 282, 282, 281, 281, 281, 280, 280, 280, 279, 279, 279, 278,
278, 278, 277, 277, 277, 276, 276, 276, 275, 275, 275, 274, 274,
274, 273, 273, 273, 272, 272,
311 272, 271, 271, 271, 270, 270, 270, 269, 269, 269, 268, 268, 268,
267, 267, 267, 266, 266, 266, 265, 265, 265, 264, 264, 264, 264,
263, 263, 263, 262, 262, 262,
312 261, 261, 261, 260, 260, 260, 259, 259, 259, 258, 258, 258, 257,
257, 257, 256, 256, 256, 255, 255, 255, 254, 254, 254, 254, 253,
253, 253, 252, 252, 252, 251,
313 251, 251, 250, 250, 250, 249, 249, 249, 248, 248, 248, 248, 247,
247, 247, 246, 246, 246, 245, 245, 245, 244, 244, 244, 243, 243,
243, 243, 242, 242, 242, 241,
314 241, 241, 240, 240, 240, 239, 239, 239, 239, 238, 238, 238, 237,
237, 237, 236, 236, 236, 235, 235, 235, 235, 234, 234, 234, 233,
233, 233, 232, 232, 232, 231,
315 231, 231, 231, 230, 230, 230, 229, 229, 229, 228, 228, 228, 228,
227, 227, 227, 226, 226, 226, 225, 225, 225, 225, 224, 224, 224,
223, 223, 223, 223, 222, 222,
316 222, 221, 221, 221, 220, 220, 220, 220, 219, 219, 219, 218, 218,
218, 218, 217, 217, 217, 216, 216, 216, 215, 215, 215, 215, 214,
214, 214, 213, 213, 213, 213,
317 212, 212, 212, 211, 211, 211, 211, 210, 210, 210, 209, 209, 209,
209, 208, 208, 208, 207, 207, 207, 207, 206, 206, 206, 205, 205,
205, 205, 204, 204, 204, 203,

318 203, 203, 203, 202, 202, 202, 201, 201, 201, 201, 200, 200, 200,
199, 199, 199, 199, 198, 198, 198, 197, 197, 197, 197, 196, 196,
196, 196, 195, 195, 195, 194,

319 194, 194, 194, 193, 193, 193, 193, 192, 192, 192, 191, 191, 191,
191, 190, 190, 190, 189, 189, 189, 189, 188, 188, 188, 188, 187,
187, 187, 186, 186, 186, 186,

320 185, 185, 185, 185, 184, 184, 184, 184, 183, 183, 183, 182, 182,
182, 182, 181, 181, 181, 181, 180, 180, 180, 179, 179, 179, 179,
178, 178, 178, 178, 177, 177,

321 177, 177, 176, 176, 176, 175, 175, 175, 175, 174, 174, 174, 174,
173, 173, 173, 173, 172, 172, 172, 172, 171, 171, 171, 170, 170,
170, 170, 169, 169, 169, 169,

322 168, 168, 168, 168, 167, 167, 167, 167, 166, 166, 166, 166, 165,
165, 165, 165, 164, 164, 164, 164, 163, 163, 163, 162, 162, 162,
162, 161, 161, 161, 161, 160,

323 160, 160, 160, 159, 159, 159, 159, 158, 158, 158, 158, 157, 157,
157, 157, 156, 156, 156, 156, 155, 155, 155, 155, 154, 154, 154,
154, 153, 153, 153, 153, 152,

324 152, 152, 152, 151, 151, 151, 151, 150, 150, 150, 150, 149, 149,
149, 149, 148, 148, 148, 148, 147, 147, 147, 147, 146, 146, 146,
146, 145, 145, 145, 145, 144,

325 144, 144, 144, 143, 143, 143, 143, 143, 142, 142, 142, 142, 141,
141, 141, 141, 140, 140, 140, 140, 139, 139, 139, 139, 138, 138,
138, 138, 137, 137, 137, 137,

326 136, 136, 136, 136, 136, 135, 135, 135, 135, 134, 134, 134, 134,
133, 133, 133, 133, 132, 132, 132, 132, 131, 131, 131, 131, 131,
130, 130, 130, 130, 129, 129,

327 129, 129, 128, 128, 128, 128, 127, 127, 127, 127, 127, 126, 126,
126, 126, 125, 125, 125, 125, 124, 124, 124, 124, 124, 123, 123,
123, 123, 122, 122, 122, 122,

328 121, 121, 121, 121, 121, 120, 120, 120, 120, 119, 119, 119, 119,
118, 118, 118, 118, 118, 117, 117, 117, 117, 116, 116, 116, 116,
115, 115, 115, 115, 115, 114,

329 114, 114, 114, 113, 113, 113, 113, 113, 112, 112, 112, 112, 111,
111, 111, 111, 111, 110, 110, 110, 110, 109, 109, 109, 109, 108,
108, 108, 108, 108, 107, 107,

330 107, 107, 106, 106, 106, 106, 106, 105, 105, 105, 105, 105, 104,
104, 104, 104, 103, 103, 103, 103, 103, 102, 102, 102, 102, 101,
101, 101, 101, 101, 100, 100,

331 100, 100, 99, 99, 99, 99, 99, 98, 98, 98, 98, 98, 97,
97, 97, 97, 96, 96, 96, 96, 96, 95, 95, 95, 95, 94,
94, 94, 94, 94, 93, 93,

332 93, 93, 93, 92, 92, 92, 92, 91, 91, 91, 91, 91, 90,
90, 90, 90, 90, 89, 89, 89, 89, 89, 88, 88, 88, 88,
87, 87, 87, 87, 87, 86,

333 86, 86, 86, 86, 85, 85, 85, 85, 85, 84, 84, 84, 84,
83, 83, 83, 83, 83, 82, 82, 82, 82, 82, 81, 81, 81,
81, 81, 80, 80, 80, 80,

334 80, 79, 79, 79, 79, 78, 78, 78, 78, 78, 77, 77, 77,
77, 77, 76, 76, 76, 76, 76, 75, 75, 75, 75, 75, 74,
74, 74, 74, 74, 73, 73,

335 73, 73, 73, 72, 72, 72, 72, 72, 71, 71, 71, 71, 71,
70, 70, 70, 70, 70, 69, 69, 69, 69, 69, 68, 68, 68,
68, 68, 67, 67, 67, 67,

336 67, 66, 66, 66, 66, 66, 65, 65, 65, 65, 65, 64, 64,
64, 64, 64, 63, 63, 63, 63, 63, 62, 62, 62, 62, 62,
61, 61, 61, 61, 61, 60,

337 60, 60, 60, 60, 59, 59, 59, 59, 59, 58, 58, 58, 58,
58, 57, 57, 57, 57, 57, 56, 56, 56, 56, 56, 56, 55,
55, 55, 55, 55, 54, 54,
338 54, 54, 54, 53, 53, 53, 53, 53, 52, 52, 52, 52, 52,
51, 51, 51, 51, 51, 50, 50, 50, 50, 50, 50, 49, 49,
49, 49, 49, 48, 48, 48,
339 48, 48, 47, 47, 47, 47, 47, 46, 46, 46, 46, 46, 46,
45, 45, 45, 45, 45, 44, 44, 44, 44, 44, 43, 43, 43,
43, 43, 43, 42, 42, 42,
340 42, 42, 41, 41, 41, 41, 41, 40, 40, 40, 40, 40, 40,
39, 39, 39, 39, 39, 38, 38, 38, 38, 38, 38, 37, 37,
37, 37, 37, 36, 36, 36,
341 36, 36, 35, 35, 35, 35, 35, 35, 34, 34, 34, 34, 34,
33, 33, 33, 33, 33, 33, 32, 32, 32, 32, 32, 31, 31,
31, 31, 31, 31, 30, 30,
342 30, 30, 30, 29, 29, 29, 29, 29, 29, 28, 28, 28, 28,
28, 27, 27, 27, 27, 27, 27, 26, 26, 26, 26, 26, 25,
25, 25, 25, 25, 25, 24,
343 24, 24, 24, 24, 24, 23, 23, 23, 23, 23, 22, 22, 22,
22, 22, 22, 21, 21, 21, 21, 21, 20, 20, 20, 20, 20,
20, 19, 19, 19, 19, 19,
344 19, 18, 18, 18, 18, 18, 17, 17, 17, 17, 17, 17, 16,
16, 16, 16, 16, 16, 15, 15, 15, 15, 15, 15, 14, 14,
14, 14, 14, 13, 13, 13,
345 13, 13, 13, 12, 12, 12, 12, 12, 12, 11, 11, 11, 11,
11, 11, 10, 10, 10, 10, 10, 10, 9, 9, 9, 9, 9,
8, 8, 8, 8, 8, 8,
346 7, 7, 7, 7, 7, 7, 6, 6, 6, 6, 6, 6, 5,
5, 5, 5, 5, 5, 4, 4, 4, 4, 4, 4, 3, 3,
3, 3, 3, 3, 2, 2,
347 2, 2, 2, 2, 1, 1, 1, 1, 1, 0, 0, 0, 360,
360, 360, 359, 359, 359, 359, 359, 359, 358, 358, 358, 358, 358,
358, 357, 357, 357, 357, 357,
348 357, 356, 356, 356, 356, 356, 356, 355, 355, 355, 355, 355, 355,
354, 354, 354, 354, 354, 354, 353, 353, 353, 353, 353, 353, 353,
352, 352, 352, 352, 352, 352,
349 351, 351, 351, 351, 351, 351, 350, 350, 350, 350, 350, 350, 349,
349, 349, 349, 349, 349, 348, 348, 348, 348, 348, 348, 347, 347,
347, 347, 347, 347, 346, 346,
350 346, 346, 346, 346, 345, 345, 345, 345, 345, 345, 345, 344, 344,
344, 344, 344, 344, 343, 343, 343, 343, 343, 343, 342, 342, 342,
342, 342, 342, 341, 341, 341,
351 341, 341, 341, 341, 340, 340, 340, 340, 340, 340, 339, 339, 339,
339, 339, 339, 338, 338, 338, 338, 338, 338, 337, 337, 337, 337,
337, 337, 337, 336, 336, 336,
352 336, 336, 336, 335, 335, 335, 335, 335, 335, 334, 334, 334, 334,
334, 334, 334, 333, 333, 333, 333, 333, 333, 332, 332, 332, 332,
332, 332, 332, 331, 331, 331,
353 331, 331, 331, 330, 330, 330, 330, 330, 330, 329, 329, 329, 329,
329, 329, 329, 328, 328, 328, 328, 328, 328, 327, 327, 327, 327,
327, 327, 327, 326, 326, 326,
354 326, 326, 326, 325, 325, 325, 325, 325, 325, 325, 324, 324, 324,
324, 324, 324, 323, 323, 323, 323, 323, 323, 323, 322, 322, 322,
322, 322, 322, 322, 321, 321,
355 321, 321, 321, 321, 320, 320, 320, 320, 320, 320, 320, 319, 319,
319, 319, 319, 319, 318, 318, 318, 318, 318, 318, 318, 317, 317,
317, 317, 317, 317, 317, 316,

356 316, 316, 316, 316, 316, 315, 315, 315, 315, 315, 315, 315, 314, 314, 314, 314, 314, 314, 314, 313, 313, 313, 313, 313, 313, 312, 312, 312, 312, 312, 312, 312,

357 311, 311, 311, 311, 311, 311, 311, 310, 310, 310, 310, 310, 310, 310, 309, 309, 309, 309, 309, 309, 309, 308, 308, 308, 308, 308, 308, 307, 307, 307, 307, 307,

358 307, 307, 306, 306, 306, 306, 306, 306, 306, 305, 305, 305, 305, 305, 305, 305, 304, 304, 304, 304, 304, 304, 304, 303, 303, 303, 303, 303, 303, 303, 302, 302,

359 302, 302, 302, 302, 302, 301, 301, 301, 301, 301, 301, 300, 300, 300, 300, 300, 300, 300, 299, 299, 299, 299, 299, 299, 299, 298, 298, 298, 298, 298, 298, 298,

360 297, 297, 297, 297, 297, 297, 297, 296, 296, 296, 296, 296, 296, 296, 295, 295, 295, 295, 295, 295, 295, 294, 294, 294, 294, 294, 294, 294, 294, 293, 293, 293,

361 293, 293, 293, 293, 292, 292, 292, 292, 292, 292, 292, 291, 291, 291, 291, 291, 291, 291, 290, 290, 290, 290, 290, 290, 290, 289, 289, 289, 289, 289, 289, 289,

362 288, 288, 288, 288, 288, 288, 288, 287, 287, 287, 287, 287, 287, 287, 286, 286, 286, 286, 286, 286, 286, 286, 285, 285, 285, 285, 285, 285, 285, 284, 284, 284,

363 284, 284, 284, 284, 283, 283, 283, 283, 283, 283, 283, 282, 282, 282, 282, 282, 282, 282, 282, 281, 281, 281, 281, 281, 281, 281, 280, 280, 280, 280, 280, 280,

364 280, 279, 279, 279, 279, 279, 279, 279, 279, 278, 278, 278, 278, 278, 278, 278, 277, 277, 277, 277, 277, 277, 277, 276, 276, 276, 276, 276, 276, 276, 276, 275,

365 275, 275, 275, 275, 275, 275, 274, 274, 274, 274, 274, 274, 274, 274, 273, 273, 273, 273, 273, 273, 273, 272, 272, 272, 272, 272, 272, 272, 272, 271, 271, 271,

366 271, 271, 271, 271, 270, 270, 270, 270, 270, 270, 270, 270, 269, 269, 269, 269, 269, 269, 269, 268, 268, 268, 268, 268, 268, 268, 268, 267, 267, 267, 267, 267,

367 267, 267, 266, 266, 266, 266, 266, 266, 266, 266, 265, 265, 265, 265, 265, 265, 265, 265, 264, 264, 264, 264, 264, 264, 264, 263, 263, 263, 263, 263, 263, 263,

368 263, 262, 262, 262, 262, 262, 262, 262, 262, 261, 261, 261, 261, 261, 261, 261, 260, 260, 260, 260, 260, 260, 260, 260, 259, 259, 259, 259, 259, 259, 259, 259,

369 258, 258, 258, 258, 258, 258, 258, 258, 257, 257, 257, 257, 257, 257, 257, 256, 256, 256, 256, 256, 256, 256, 256, 255, 255, 255, 255, 255, 255, 255, 255, 254,

370 254, 254, 254, 254, 254, 254, 254, 253, 253, 253, 253, 253, 253, 253, 253, 252, 252, 252, 252, 252, 252, 252, 252, 251, 251, 251, 251, 251, 251, 251, 251, 250,

371 250, 250, 250, 250, 250, 250, 249, 249, 249, 249, 249, 249, 249, 249, 248, 248, 248, 248, 248, 248, 248, 248, 247, 247, 247, 247, 247, 247, 247, 247, 246, 246,

372 246, 246, 246, 246, 246, 246, 245, 245, 245, 245, 245, 245, 245, 245, 245, 244, 244, 244, 244, 244, 244, 244, 244, 243, 243, 243, 243, 243, 243, 243, 243, 242,

373 242, 242, 242, 242, 242, 242, 242, 241, 241, 241, 241, 241, 241, 241, 241, 240, 240, 240, 240, 240, 240, 240, 240, 239, 239, 239, 239, 239, 239, 239, 239, 238,

374 238, 238, 238, 238, 238, 238, 238, 238, 237, 237, 237, 237, 237, 237, 237, 237, 236, 236, 236, 236, 236, 236, 236, 236, 235, 235, 235, 235, 235, 235, 235, 235,

375 234, 234, 234, 234, 234, 234, 234, 234, 234, 233, 233, 233, 233,
233, 233, 233, 233, 232, 232, 232, 232, 232, 232, 232, 232, 231,
231, 231, 231, 231, 231, 231,
376 231, 231, 230, 230, 230, 230, 230, 230, 230, 230, 229, 229, 229,
229, 229, 229, 229, 229, 229, 228, 228, 228, 228, 228, 228, 228,
228, 227, 227, 227, 227, 227,
377 227, 227, 227, 227, 226, 226, 226, 226, 226, 226, 226, 226, 225,
225, 225, 225, 225, 225, 225, 225, 225, 224, 224, 224, 224, 224,
224, 224, 224, 223, 223, 223,
378 223, 223, 223, 223, 223, 223, 222, 222, 222, 222, 222, 222, 222,
222, 222, 221, 221, 221, 221, 221, 221, 221, 221, 220, 220, 220,
220, 220, 220, 220, 220, 220,
379 219, 219, 219, 219, 219, 219, 219, 219, 219, 218, 218, 218, 218,
218, 218, 218, 218, 217, 217, 217, 217, 217, 217, 217, 217, 217,
216, 216, 216, 216, 216, 216,
380 216, 216, 216, 215, 215, 215, 215, 215, 215, 215, 215, 215, 214,
214, 214, 214, 214, 214, 214, 214, 214, 213, 213, 213, 213, 213,
213, 213, 213, 213, 212, 212,
381 212, 212, 212, 212, 212, 212, 211, 211, 211, 211, 211, 211, 211,
211, 211, 210, 210, 210, 210, 210, 210, 210, 210, 210, 209, 209,
209, 209, 209, 209, 209, 209,
382 209, 208, 208, 208, 208, 208, 208, 208, 208, 208, 207, 207, 207,
207, 207, 207, 207, 207, 207, 206, 206, 206, 206, 206, 206, 206,
206, 206, 206, 205, 205, 205,
383 205, 205, 205, 205, 205, 205, 204, 204, 204, 204, 204, 204, 204,
204, 204, 203, 203, 203, 203, 203, 203, 203, 203, 203, 202, 202,
202, 202, 202, 202, 202, 202,
384 202, 201, 201, 201, 201, 201, 201, 201, 201, 201, 200, 200, 200,
200, 200, 200, 200, 200, 200, 200, 199, 199, 199, 199, 199, 199,
199, 199, 199, 198, 198, 198,
385 198, 198, 198, 198, 198, 198, 197, 197, 197, 197, 197, 197, 197,
197, 197, 197, 196, 196, 196, 196, 196, 196, 196, 196, 196, 195,
195, 195, 195, 195, 195, 195,
386 195, 195, 195, 194, 194, 194, 194, 194, 194, 194, 194, 194, 193,
193, 193, 193, 193, 193, 193, 193, 193, 193, 192, 192, 192, 192,
192, 192, 192, 192, 192, 191,
387 191, 191, 191, 191, 191, 191, 191, 191, 191, 190, 190, 190, 190,
190, 190, 190, 190, 190, 189, 189, 189, 189, 189, 189, 189, 189,
189, 189, 188, 188, 188, 188,
388 188, 188, 188, 188, 188, 188, 187, 187, 187, 187, 187, 187, 187,
187, 187, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 185,
185, 185, 185, 185, 185, 185,
389 185, 185, 185, 184, 184, 184, 184, 184, 184, 184, 184, 184, 184,
183, 183, 183, 183, 183, 183, 183, 183, 183, 183, 182, 182, 182,
182, 182, 182, 182, 182, 182,
390 181, 181, 181, 181, 181, 181, 181, 181, 181, 181, 180, 180, 180,
180, 180, 180, 180, 180, 180, 180, 179, 179, 179, 179, 179, 179,
179, 179, 179, 179, 178, 178,
391 178, 178, 178, 178, 178, 178, 178, 178, 177, 177, 177, 177, 177,
177, 177, 177, 177, 177, 176, 176, 176, 176, 176, 176, 176, 176,
176, 176, 176, 175, 175, 175,
392 175, 175, 175, 175, 175, 175, 175, 174, 174, 174, 174, 174, 174,
174, 174, 174, 174, 173, 173, 173, 173, 173, 173, 173, 173, 173,
173, 172, 172, 172, 172, 172,
393 172, 172, 172, 172, 172, 171, 171, 171, 171, 171, 171, 171, 171,
171, 171, 171, 170, 170, 170, 170, 170, 170, 170, 170, 170, 170,
169, 169, 169, 169, 169, 169,

```
        169, 169, 169, 169, 168, 168, 168, 168, 168, 168, 168, 168, 168,
        168, 168, 167, 167, 167, 167, 167, 167, 167, 167, 167, 167, 166,
        166, 166, 166, 166, 166, 166,
        166, 166, 166, 166, 165, 165, 165, 165, 165, 165, 165, 165, 165,
        165, 164, 164, 164, 164, 164, 164, 164, 164, 164, 164, 164, 163,
        163, 163, 163, 163, 163, 163,
        163, 163, 163, 163, 162, 162, 162, 162, 162, 162, 162, 162, 162,
        162, 161, 161, 161, 161, 161, 161, 161, 161, 161, 161, 161, 160,
        160, 160, 160, 160, 160, 160,
        160, 160, 160, 160, 159, 159, 159, 159, 159, 159, 159, 159, 159,
        159, 159, 158, 158, 158, 158, 158, 158, 158, 158, 158, 158, 157,
        157, 157, 157, 157, 157, 157,
        157, 157, 157, 157, 156, 156, 156, 156, 156, 156, 156, 156, 156,
        156, 156, 155, 155, 155, 155, 155, 155, 155, 155, 155, 155, 155,
        154, 154, 154, 154, 154, 154,
        154, 154, 154, 154, 154, 153, 153, 153, 153, 153, 153, 153, 153,
        153, 153, 153, 152, 152, 152, 152, 152, 152, 152, 152, 152, 152,
        152, 151, 151, 151, 151, 151,
        151, 151, 151, 151, 151, 151, 151, 150, 150, 150, 150, 150, 150,
        150, 150, 150, 150, 150, 149, 149, 149, 149, 149, 149, 149, 149,
        149, 149, 149, 148, 148, 148,
        148, 148, 148, 148, 148, 148, 148, 148, 147, 147, 147, 147, 147,
        147, 147, 147, 147, 147, 147, 147, 146, 146, 146, 146, 146, 146,
        146, 146, 146, 146, 146, 145,
        145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 144, 144, 144,
        144, 144, 144, 144, 144, 144, 144, 144, 144, 143, 143, 143, 143,
        143, 143, 143, 143, 143, 143,
        143, 142, 142, 142, 142, 142, 142, 142, 142, 142, 142, 142, 142,
        141, 141, 141, 141, 141, 141, 141, 141, 141, 141, 141, 140, 140,
        140, 140, 140, 140, 140, 140,
        140, 140, 140, 140, 139, 139, 139, 139, 139, 139, 139, 139, 139,
        139, 139, 139, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138,
        138, 137, 137, 137, 137, 137,
        137, 137, 137, 137, 137, 137, 137, 136, 136, 136, 136, 136, 136,
        136, 136, 136, 136, 136, 136, 135, 135, 135, 135, 135, 135, 135,
        135, 135, 135, 135, 135, 134,
        134, 134, 134, 134, 134, 134, 134, 134, 134, 134, 133, 133, 133,
        133, 133, 133, 133, 133, 133, 133, 133, 133, 132, 132, 132, 132,
        132, 132, 132, 132, 132, 132,
        132, 132, 131, 131, 131, 131, 131, 131, 131, 131, 131, 131, 131,
        131, 130, 130, 130, 130, 130, 130, 130, 130, 130, 130, 130, 130,
        129, 129, 129, 129, 129, 129,
        129, 129, 129, 129, 129, 129, 128, 128, 128, 128, 128, 128, 128,
        128, 128, 128, 128, 128, 127, 127, 127, 127, 127, 127, 127, 127,
        127, 127, 127, 127, 127, 126,
        126, 126, 126, 126, 126, 126, 126, 126, 126, 126, 126, 125, 125,
        125, 125, 125, 125, 125, 125, 125, 125, 125, 125, 124, 124, 124,
        124, 124, 124, 124, 124, };
    //wrap adc value
    if (adc > 4095)
        adc = 4095;
    //return the angle value from the correct LUT
    switch (board) {
        case 1:
            return cpLUT_1[adc];
        case 2:
            return cpLUT_2[adc];
```

```
        case 3:
            return cpLUT_3[adc];
        default:
            return 180;
    }
}
```

A.4.11 activeLUT.c

```c
/*
 * HARA_LUTs.c
 *
 *  Created on: May 28, 2019
 *      Author: MichaelBolt
 */

#include "HARA_LUTs.h"

//****************************************************************************
//
//! Finds linear phase gradient corresponding to a point
//!
//! \param point is the direction you'd like to steer the beam towards (0-90)
//!
//! This function returns an integer that corresponds to the linear phase
//! gradient needed to steer the beam of a 4-element, half wavelength spaced
//! linear antenna array
//!
//! \return integer phase gradient between elements
//
//****************************************************************************
uint8_t activeSteer_LUT(unsigned int point) {
    static const uint8_t activeSteerLUT[91] = {
         0,  2,  3,  5,  6,  8,  9, 11, 13, 14,
        16, 17, 19, 20, 22, 23, 25, 26, 28, 29,
        31, 32, 34, 35, 37, 38, 39, 41, 42, 44,
        45, 46, 48, 49, 50, 52, 53, 54, 55, 57,
        58, 59, 60, 61, 63, 64, 65, 66, 67, 68,
        69, 70, 71, 72, 73, 74, 75, 75, 76, 77,
        78, 79, 79, 80, 81, 82, 82, 83, 83, 84,
        85, 85, 86, 86, 87, 87, 87, 88, 88, 88,
        89, 89, 89, 89, 90, 90, 90, 90, 90, 90,
        90};
    if (point > 90)
        return activeSteerLUT[90];
    return activeSteerLUT[point];
}
```

A.5 Python GUI Source Code

A.5.1 HARA_app.py

```
from tkinter import *
from tkinter import ttk
import matplotlib
matplotlib.use("TkAgg")
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from matplotlib.figure import Figure
import matplotlib.animation as animation
from matplotlib import style
style.use('ggplot')
from random import *
#my packages
from ConfigTab import PLL
from ConfigTab import PFD
from ADCgraph import ADCgraph
from CMD_tab import CMD_tab
from Phase_tab import Phase_tab
from ManualTab import Manual_tab
from SerialComms import comSelectWindow
from SerialComms import HARAuartObj
import SerialComms

VER_NUMBER = '5.3'  #version number for application

#global HARA object, stored in SerialComms module
SerialComms.HARA = HARAuartObj()

#create root window
root = Tk()
root.title("H.A.R.A. Interface")


#create a title frame and populate
title_frame = ttk.Frame(root)
title_frame['padding'] = (10, 10)
title_frame.columnconfigure(2, weight=1)
#logo
logo = PhotoImage(file='STORMlab.png')
ttk.Label(title_frame, image = logo, anchor = 'e').grid(column=2, row=0, sticky = (N, S, E, W))
#title
ttk.Label(title_frame, text = 'H.A.R.A.', foreground = "#03244d", anchor = 'sw', font = ('TkDefaultFont', 40)).grid(column=0, row=0, sticky=(N, S, E, W), pady = 10)
ttk.Label(title_frame, text = 'v' + VER_NUMBER, foreground = '#dd550c', anchor = 'sw', font = ('TkDefaultFont', 10)).grid(column=1, row=0, sticky = (N, S, E, W), pady = 20)


#create notebook for middle section
notebook_frame = ttk.Frame(root)
notebook_frame['padding'] = (10,5)
```

```
notebook_frame.rowconfigure(0, weight = 1)
notebook_frame.columnconfigure(0, weight = 1)
notebook = ttk.Notebook(notebook_frame)
notebook.grid(row=0,column=0,sticky=(N,S,E,W))
#frames for notebook tabs
Config_frame = ttk.Frame(notebook)
CMD_frame = ttk.Frame(notebook)
Manual_frame = ttk.Frame(notebook)
Phase_frame = ttk.Frame(notebook)
notebook.add(Config_frame, text = 'Config')
notebook.add(CMD_frame, text = 'Commands')
notebook.add(Phase_frame, text = 'Manual (Phase)')
notebook.add(Manual_frame, text = 'Manual (DAC)')
#PLL tab
PLL1 = PLL(Config_frame, 1, 'Downconversion')
PLL2 = PLL(Config_frame, 2, 'Transmission')
PFD = PFD(Config_frame)
#CMD tab
CMD = CMD_tab(CMD_frame)
#Phase tab
PHASE = Phase_tab(Phase_frame)
#DAC Manual tab
MAN = Manual_tab(Manual_frame)

#create frames for ADC graphing and csv logging
ADC_frame = ttk.Frame(root, padding = (10,5))
ADC_frame.rowconfigure(0, weight = 1)
ADC_frame.columnconfigure(0, weight = 1)
LOG_frame = ttk.Frame(root, padding = (10,5))
LOG_frame.rowconfigure(0, weight = 1)
LOG_frame.columnconfigure(0, weight = 1)
ADCgraphs = ADCgraph(ADC_frame, LOG_frame, VER_NUMBER)

#Packet display frame
pkt_frame = ttk.Frame(root, padding = (10,5))
SerialComms.PKTdisp = SerialComms.pktDisplay(pkt_frame)

#place frames
title_frame.grid(column=0, row=0, sticky = (N,S,E,W))
notebook_frame.grid(column=0, row=1, sticky = (N, S, E, W))
LOG_frame.grid(column=0, row=2, sticky = (N,S,E,W))
pkt_frame.grid(column=0, row=3, sticky = (N,S,E,W))
ADC_frame.grid(column=1, row=0, rowspan = 4, sticky = (N,S,E,W))
root.columnconfigure(0,weight = 0)
root.rowconfigure(0, weight = 0)
root.rowconfigure(1, weight = 0)
root.rowconfigure(2, weight = 0)
root.rowconfigure(3, weight = 1)


######  Pop-up COM selection window ######
comWindow = Toplevel(root)
comWindow.title('COM Port Select')
comFrame = ttk.Frame(comWindow, padding = (10,5))
comFrame.grid(column = 0, row = 0, sticky = (N,S,E,W))
comSelectWindow = comSelectWindow(comFrame, comWindow)
```

```

#main loop
root.mainloop()
```

A.5.2 SerialComms.py

```
from tkinter import *
from tkinter import ttk
import serial
import serial.tools.list_ports
import threading

HARA = []
PKTdisp = []

class pktDisplay:
    adc = [0,1,2,3]
    theta = [0,1,2,3]
    psi = [0,1,2,3]
    xi = [0,1,2,3]
    delta_i = [0,1,2,3]
    delta_j = [0,1,2,3]
    sw = [0,1,2,3]
    mux_val = [0,1,2,3]
    width = 4

    def __init__(self, parent):
        #set up titles
        ttk.Label(parent, text = 'Ele', anchor = CENTER).grid(row = 0, column = 0, sticky = (N,S,E,W))
        ttk.Label(parent, text = 'PFD', anchor = CENTER).grid(row = 0, column = 1, sticky = (N,S,E,W))
        ttk.Label(parent, text = 'ADC', anchor = CENTER).grid(row = 0, column = 2, sticky = (N,S,E,W))
        ttk.Label(parent, text = u'\u03B8', anchor = CENTER).grid(row = 0, column = 3, sticky = (N,S,E,W))          #theta
        ttk.Label(parent, text = u'\u03BE', anchor = CENTER).grid(row = 0, column = 4, sticky = (N,S,E,W))          #xi
        ttk.Label(parent, text = u'\u03C8', anchor = CENTER).grid(row = 0, column = 5, sticky = (N,S,E,W))          #psi
        ttk.Label(parent, text = u'\u0394' + 'i', anchor = CENTER).grid(row = 0, column = 6, sticky = (N,S,E,W))     #delta_i
        ttk.Label(parent, text = u'\u0394' + 'j', anchor = CENTER).grid(row = 0, column = 7, sticky = (N,S,E,W))     #delta_j
        ttk.Label(parent, text = 'SW', anchor = CENTER).grid(row = 0, column = 8, sticky = (N,S,E,W))               #SW (CP/DAC)

        #set up information
        for i in [0,1,2,3]:
            self.adc[i] = StringVar()
            self.theta[i] = StringVar()
            self.psi[i] = StringVar()
            self.xi[i] = StringVar()
            self.delta_i[i] = StringVar()
            self.delta_j[i] = StringVar()
            self.sw[i] = StringVar()
            self.mux_val[i] = StringVar()
            ttk.Label(parent, text = '#' + str(i), anchor = CENTER).grid(row = 1+i, column = 0, sticky = (N,S,E,W))
```

```
            ttk.Label(parent, textvariable = self.mux_val[i], anchor = CENTER, width = self.width).grid(row = 1+i, column = 1, sticky = (N,S,E,W))
            ttk.Label(parent, textvariable = self.adc[i], anchor = CENTER, width = self.width).grid(row = 1+i, column = 2, sticky = (N,S,E,W))
            ttk.Label(parent, textvariable = self.theta[i], anchor = CENTER, width = self.width).grid(row = 1+i, column = 3, sticky = (N,S,E,W))
            ttk.Label(parent, textvariable = self.xi[i], anchor = CENTER, width = self.width).grid(row = 1+i, column = 4, sticky = (N,S,E,W))
            ttk.Label(parent, textvariable = self.psi[i], anchor = CENTER, width = self.width).grid(row = 1+i, column = 5, sticky = (N,S,E,W))
            ttk.Label(parent, textvariable = self.delta_i[i], anchor = CENTER, width = self.width).grid(row = 1+i, column = 6, sticky = (N,S,E,W))
            ttk.Label(parent, textvariable = self.delta_j[i], anchor = CENTER, width = self.width).grid(row = 1+i, column = 7, sticky = (N,S,E,W))
            ttk.Label(parent, textvariable = self.sw[i], anchor = CENTER, width = self.width).grid(row = 1+i, column = 8, sticky = (N,S,E,W))
        parent.columnconfigure(0, weight = 1)
        parent.columnconfigure(1, weight = 1)
        parent.columnconfigure(2, weight = 1)
        parent.columnconfigure(3, weight = 1)
        parent.columnconfigure(4, weight = 1)
        parent.columnconfigure(5, weight = 1)
        parent.columnconfigure(6, weight = 1)
        parent.columnconfigure(7, weight = 1)
        parent.columnconfigure(8, weight = 1)

    def update(self, HARA):
        for i in [0,1,2,3]:
            self.adc[i].set(str(HARA.adc[i]))
            self.theta[i].set(str(HARA.theta[i]))
            self.psi[i].set(str(HARA.psi[i]))
            self.xi[i].set(str(HARA.xi[i]))
            self.delta_i[i].set(str(HARA.delta_i[i]))
            self.delta_j[i].set(str(HARA.delta_j[i]))
            if HARA.sw[i] == 0:
                self.sw[i].set('DAC')
            else:
                self.sw[i].set('CP')
            self.mux_val[i].set(str(HARA.mux_val[i]))

#pop-up window to establish a serial connection
class comSelectWindow:
    ports = []  #list of available COM ports
    port = []   #name of selected port
    ser = []    #serial object for connecting

    def __init__(self, parent, window):
        self.port = StringVar()
        self.window = window
```

```
        #initial ports search
        for comport in serial.tools.list_ports.comports():
            self.ports.append(comport.device)
        #search for ports button
        ttk.Button(parent, text = 'Search for Ports', command = self.searchForPorts).grid(row = 0, column = 0, sticky = (N,S,E,W))
        #dropdown list of ports
        self.portSelect = ttk.Combobox(parent, textvariable = self.port, state = 'readonly', values = self.ports, width = 30)
        self.portSelect.grid(row = 0, column = 1, columnspan = 2, sticky = (N,S,E,W))
        self.portSelect.bind('<<ComboboxSelected>>',self.portChange)
        #connect button
        ttk.Button(parent, text = 'Connect', command = self.connectBtn).grid(row = 1, column = 1, sticky =(N,S,E,W))
        #error message label
        self.errMessage = ttk.Label(parent, text = '', foreground = 'red', anchor = 'center', width = 20)
        self.errMessage.grid(row = 1, column = 2, sticky = (N,S,E,W))

    def searchForPorts(self):
        self.ports = []
        for comport in serial.tools.list_ports.comports():
            self.ports.append(comport.device)
        self.portSelect['values'] = self.ports

    def portChange(self, val):
        self.portSelect.selection_clear()
        self.errMessage['text'] = ''

    def connectBtn(self):
        temp = HARA.connect(self.port.get())
        #successful connection
        if temp == 0:
            self.errMessage['text'] = 'Connected!'
            self.errMessage['foreground'] = 'green'
            #close parent window on success after 1 sec
            t = threading.Timer(1, self.suicide)
            t.start()
        #could not connect to port or invalid connect word
        elif temp == 1:
            self.errMessage['text'] = 'ERR: Wrong Port'
            self.errMessage['foreground'] = 'red'
        #empty buffer on connected port
        elif temp == 2:
            self.errMessage['text'] = 'ERR: Empty Buffer'
            self.errMessage['foreground'] = 'red'

    def suicide(self):
        self.window.destroy()


#HARA UART serial communication class
class HARAuartObj:
    ser = []                    #serial object
    state = 0                   #current state
    adc = [0,0,0,0]             #current adc data
```

```
    theta = [0,0,0,0]       #current theta value
    psi = [0,0,0,0]         #current psi value
    xi = [0,0,0,0]          #current xi value
    delta_i = [0,0,0,0]     #current delta_i value
    delta_j = [0,0,0,0]     #current delta_j value
    sw = [0,0,0,0]          #current CP/DAC switch states
    mux_val = [False, False, False, False]      #current mux pin values
    newData = False         #boolean to indicate new data has been received

    def __init__(self):
        pass

    #function to make a connection
    #   returns:
    #       0 = success
    #       1 = wrong port
    #       2 = empty buffer
    def connect(self,port):
        try:
            #try to open the specified port
            self.ser = serial.Serial(port = port, baudrate = 115200, timeout = 1)
            #if it was a success, listen for our connect message: 0xBA
            rec = self.ser.readline()
            try:
                #if it's not our message, it's the wrong port
                if rec[0] != 0xBA:
                    raise serial.SerialException
                #if it was our message..
                else:
                    #send UART connect message: 0xBA
                    self.ser.write(bytes([0xBA]))
                    #flush UART rx buffer
                    self.ser.reset_input_buffer()
                    #kick off receive thread
                    rxThread = threading.Thread(target = self.receive, daemon = True)
                    rxThread.start()
                    return 0
            except IndexError:
                return 2
        except serial.SerialException:
            return 1


    dict_cmd = {
        # 'CHANGE_CP_SW'          : 0x30,
        'PLL_RECONFIGURE'       : 0x40,
        'PLL_RECAL'             : 0x50,
        # 'FIND_PHASE_OFFSET'     : 0x60,
        # 'FIND_RX_PHASE_OFFSET'  : 0x70,
        'ADJUST_DELTA_MANUALLY' : 0x70,
        'SET_DELTAS_BY_CP'      : 0x80,
        'PFD_RECONFIGURE'       : 0x90,
        'ACTIVE_STEER'          : 0xA0,
        'STOP_ACTIVE_STEER'     : 0xB0,
        'RETRODIRECT'           : 0xC0,
```

```
        'STOP_RETRODIRECT'      : 0xD0,
        'DAC_WRITE'             : 0xE0,
        'PHASE_WRITE'           : 0xF0,
    }
    #function to send a UART command (4 bytes long)
    def command(self, cmd, payload = (0,0,0)):
        try:
            #if PFD_RECONFIGURE, 6 byte message
            if (cmd is 'PFD_RECONFIGURE'):
                msg = [0, 0, 0, 0, 0, 0]
                msg[0] = self.dict_cmd[cmd] | (payload[0] & 0x000F)
                msg[1] = ((payload[1] & 0x000F) << 4) | (payload[2] & 0x000F)
                msg[2] = (payload[3] & 0x000F)
                msg[3] = ((payload[4] & 0x000F) << 4) | (payload[5] & 0x000F)
                msg[4] = (payload[6] & 0x000F)
                msg[5] = ((payload[7] & 0x000F) << 4) | (payload[8] & 0x000F)
                self.ser.write(bytes(msg))

            #if PLL_RECONGIGURE, 4 byte message
            elif (cmd is 'PLL_RECONFIGURE') or (cmd is 'DAC_WRITE'):
                msg = [0, 0, 0, 0]
                msg[0] = (self.dict_cmd[cmd]) | (payload[0] & 0x000F)
                msg[1] = ((payload[1] & 0x000F) << 4) | (payload[2] & 0x000F)
                msg[2] = ((payload[3] & 0x000F) << 4) | (payload[4] & 0x000F)
                msg[3] = ((payload[5] & 0x000F) << 4) | (payload[6] & 0x000F)
                self.ser.write(bytes(msg))
            #if not PLL_RECONFIGURE, 2 bytes
            else:
                msg = [0, 0]
                msg[0] = (self.dict_cmd[cmd]) | (payload[0] & 0x000F)
                msg[1] = ((payload[1] & 0x000F) << 4) | (payload[2] & 0x000F)
                self.ser.write(bytes(msg))
        except:
            print("HEY DUMMY, connect the board :)")

    #function to read the UART transmissions in a separate thread
    def receive(self):
        while True:
            try:
                pkt = self.ser.read(21)
                self.state       = pkt[0]
                #determine mux value
                j = 1
                for i in [0x08, 0x10, 0x20]:
                    if self.state & i:
                        self.mux_val[j] = True
                    else:
                        self.mux_val[j] = False
                    j = j + 1
                #clear mux values
```

```
            self.state = self.state & ~0x38
            #every 50 packets, it's the offsets
            if self.state & 0x80:
                self.state      = self.state & 0x3F
                self.adc[0]     = (pkt[1 ] << 8) | pkt[2]
                self.adc[1]     = (pkt[3 ] << 8) | pkt[4]
                self.adc[2]     = (pkt[5 ] << 8) | pkt[6]
                self.adc[3]     = (pkt[7 ] << 8) | pkt[8]
                self.delta_i[1] = self.sign_extend(((pkt[9] << 8)  | pkt[10]), 16)
                self.delta_i[2] = self.sign_extend(((pkt[11] << 8) | pkt[12]), 16)
                self.delta_i[3] = self.sign_extend(((pkt[13] << 8) | pkt[14]), 16)
                self.delta_j[1] = self.sign_extend(((pkt[15] << 8) | pkt[16]), 16)
                self.delta_j[2] = self.sign_extend(((pkt[17] << 8) | pkt[18]), 16)
                self.delta_j[3] = self.sign_extend(((pkt[19] << 8) | pkt[20]), 16)
            #every other packet is xi and sw values
            elif self.state & 0x40:
                self.state      = self.state & 0x3F
                self.adc[0]     = (pkt[1 ] << 8) | pkt[2]
                self.adc[1]     = (pkt[3 ] << 8) | pkt[4]
                self.adc[2]     = (pkt[5 ] << 8) | pkt[6]
                self.adc[3]     = (pkt[7 ] << 8) | pkt[8]
                self.sw[1]      = self.sign_extend(((pkt[9 ] << 8) | pkt[10]), 16)
                self.sw[2]      = self.sign_extend(((pkt[11] << 8) | pkt[12]), 16)
                self.sw[3]      = self.sign_extend(((pkt[13] << 8) | pkt[14]), 16)
                self.xi[1]      = self.sign_extend(((pkt[15] << 8) | pkt[16]), 16)
                self.xi[2]      = self.sign_extend(((pkt[17] << 8) | pkt[18]), 16)
                self.xi[3]      = self.sign_extend(((pkt[19] << 8) | pkt[20]), 16)
            #otherwise, it's psi and theta
            else:
                self.adc[0]     = (pkt[1 ] << 8) | pkt[2]
                self.adc[1]     = (pkt[3 ] << 8) | pkt[4]
                self.adc[2]     = (pkt[5 ] << 8) | pkt[6]
                self.adc[3]     = (pkt[7 ] << 8) | pkt[8]
                self.theta[1]   = self.sign_extend(((pkt[9 ] << 8) | pkt[10]), 16)
                self.theta[2]   = self.sign_extend(((pkt[11] << 8) | pkt[12]), 16)
                self.theta[3]   = self.sign_extend(((pkt[13] << 8) | pkt[14]), 16)
                self.psi[1]     = self.sign_extend(((pkt[15] << 8) | pkt[16]), 16)
                self.psi[2]     = self.sign_extend(((pkt[17] << 8) | pkt[18]), 16)
                self.psi[3]     = self.sign_extend(((pkt[19] << 8) | pkt[20]), 16)
```

```
                    self.newData         = True
                    #update the sreen
                    PKTdisp.update(self)
                #if there was a timeout, just don't worry about it
                except IndexError:
                    pass

    def sign_extend(self,value, bits):
        sign_bit = 1 << (bits - 1)
        return (value & (sign_bit - 1)) - (value & sign_bit)
```

A.5.3 ADCgraph.py

```
from tkinter import *
from tkinter import ttk
#for plotting
import matplotlib
matplotlib.use("TkAgg")
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from matplotlib.figure import Figure
import matplotlib.animation as animation
from matplotlib import style
style.use('ggplot')
#for numbers
from random import random
from numpy import linspace
from numpy import zeros
import datetime
import SerialComms


class ADCgraph:
    titleFontSize = 8   #font size for subplot titles
    axesFontSize = 6
    refreshRate = 10    #number of frames per second
    VER_NUMBER = 0

    def __init__(self, parent, log_parent, ver):
        #store version number
        self.VER_NUMBER = ver
        #####   ADC graphs  #####
        #create figure for graphing
        self.f = Figure(figsize = (4,5), dpi = 100, tight_layout = 'true')
        #adc subplot initialization
        self.adcsub = [0,1,2,3]
        self.adcln = [0,1,2,3]
        self.xdata = [0,1,2,3]
        self.ydata = [0,1,2,3]
        loc = [1,2,3,4]
        for i in [0, 1, 2, 3]:
            self.adcsub[i] = self.f.add_subplot(4,1,loc[i])
            self.adcsub[i].set_title('ADC ' + str(i),fontsize = self.titleFontSize)
            self.adcsub[i].set_xlabel('Time (s)',fontsize = self.axesFontSize)
            self.adcsub[i].set_ylabel('V',fontsize = self.axesFontSize)
            self.adcsub[i].set_xticklabels([-5, -4, -3, -2, -1, 0], fontsize = self.axesFontSize)
            self.adcsub[i].set_yticklabels([0, 2048, 4096], fontsize = self.axesFontSize)
            self.xdata[i] = linspace(-5, 0, 51)
            self.ydata[i] = zeros(51)
            self.adcln[i], = self.adcsub[i].plot(self.xdata[i],self.ydata[i])
        #create canvas widget with figure embedded in it
        self.canvas = FigureCanvasTkAgg(self.f, parent)
        self.grid(column=0, row=0, sticky =(N,S,E,W), columnspan=4)
```

```
        #configure animation
        self.ani = animation.FuncAnimation(self.f, self.animateFcn, interval = (1000/self.refreshRate), init_func=self.animateInit, blit=True)
        #####   LOG area    #####
        #create button for logging data
        self.logging = False
        self.logBtn = ttk.Button(log_parent, text = 'Log', command = self.logBtnPress,width=10)
        self.logBtn.grid(column=2,row=0,sticky=(N,S,E,W))
        #create text entry for log file
        ttk.Label(log_parent, text = 'Filename:',anchor='w').grid(column=0, row=0, sticky=(N,S,E,W))
        self.logFileName = StringVar()
        self.logFileName.set(datetime.datetime.now().strftime('%Y_%m_%d_%H%M%S') + '.csv')
        ttk.Entry(log_parent, textvariable=self.logFileName, width = 20).grid(column=1,row=0,sticky=(N,S,E,W))
        #create button for logging a flag
        self.logFlag = False
        ttk.Button(log_parent, text = 'Log Flag', command = self.logFlagBtnPress, width = 10).grid(column = 3, row = 0, sticky = (N,S,E,W))

    #call to grid function for GUI placement
    def grid(self, **keyword_params):
        self.canvas.get_tk_widget().grid(keyword_params)

    #initialization for animation (sets up backgrounds)
    def animateInit(self):
        for i in [0,1,2,3]:
            self.adcsub[i].set_xlim(-5, 0)
            self.adcsub[i].set_ylim(0, 4096)
        return tuple(self.adcln,)

    #animation function called once per frame
    def animateFcn(self,frame_num):
        #if we've got new data to plot/log
        if SerialComms.HARA.newData:
            #update graphs
            for i in [0,1,2,3]:
                self.ydata[i][:len(self.ydata[i])-1] = self.ydata[i][1:]
                self.ydata[i][len(self.ydata[i])-1] = SerialComms.HARA.adc[i]
                self.adcln[i].set_data(self.xdata[i],self.ydata[i])
            #if logging, populate row
            if self.logging:
                self.logfile.write(str(SerialComms.HARA.state) + ', ')
                self.logfile.write(str(SerialComms.HARA.adc[0]) + ', ')
                for i in [1,2,3]:
                    self.logfile.write(str(SerialComms.HARA.mux_val[i]) + ', ')
                    self.logfile.write(str(SerialComms.HARA.adc[i]) + ', ')
                    self.logfile.write(str(SerialComms.HARA.psi[i]) + ', ')
                    self.logfile.write(str(SerialComms.HARA.theta[i]) + ',')
```

```
                    self.logfile.write(str(SerialComms.HARA.xi[i])+ ',')
                    self.logfile.write(str(SerialComms.HARA.delta_i[i]) + ', ')
                    self.logfile.write(str(SerialComms.HARA.delta_j[i]) + ',')
                    self.logfile.write(str(SerialComms.HARA.sw[i]) + ', ')
                self.logfile.write('%d,' % self.logFlag)
                self.logFlag = False
                now = datetime.datetime.now()
                self.logfile.write('%d, %d, %d, %d\n' % (now.hour, now.minute, now.second, now.microsecond))
            #clear new data flag
            SerialComms.HARA.newData = False
        return tuple(self.adcln,)

    def logBtnPress(self):
        #if we aren't logging, start
        if not self.logging:
            self.logBtn['text'] = 'Stop logging'
            #open file and write header
            self.logfile = open(self.logFileName.get(),'w')
            self.logfile.write('Version:,' + self.VER_NUMBER + '\n')
            self.logfile.write('state, adc0, mux1, adc1, psi1, theta1, xi1, di1, dj1, sw1, mux2, adc2, psi2, theta2, xi2, di2, dj2, sw2, mux3, adc3, psi3, theta3, xi3, di3, dj3, sw3, Flag, SystemHour, SystemMinute, SystemSecond, SystemuS\n')
            #update flag to start logging in animation loop
            self.logging = True
        #if we are logging, stop
        else:
            #update button and flag
            self.logging = False
            self.logBtn['text'] = 'Log'
            #close file
            self.logfile.close()
            temp = self.logFileName.get()
            #if we were just using the datetime as a log name, just use the new datetime
            if datetime.datetime.now().strftime('%Y_%m_%d') in temp:
                self.logFileName.set(datetime.datetime.now().strftime('%Y_%m_%d_%H%M%S') +
                '.csv')
            #if our file ends in _x, increment it
            elif temp[len(temp) - 6] is '_':
                self.logFileName.set(temp[:len(temp)-5] + str(int(temp[len(temp)-5])+1) + temp[len(temp)-4:])
            #if our file ends in _xx, increment it
            elif temp[len(temp) - 7] is '_':
                self.logFileName.set(temp[:len(temp)-6] + str(int(temp[(len(temp)-6):(len(temp)-4)])+1) + temp[len(temp)-4:])
            #if using some other format, add a b!
            else:
                self.logFileName.set(temp[:len(temp)-4] + 'b' + temp[len(temp)-4:])

```

```
    def logFlagBtnPress(self):
        self.logFlag = True
```

A.5.4 ConfigTab.py

```
from tkinter import *
from tkinter import ttk
import SerialComms


class PLL:
    #dictionary definitions for message construction
    dict_power = {  -9 : 0x00,
                    -6 : 0x01,
                    -3 : 0x02,
                    0 : 0x03,
                }
    dict_startFreq = {  1 : 2410,
                        2 : 2500,
                        3 : 909
                    }

    def __init__(self, parent, number, msg = '???'):
        self.number = number                            #PLL.number is identifier
        self.parent = parent                            #parent frame
        self.freq = self.dict_startFreq[self.number]    #frequency (integer multiple of 1 MHz)
        self.power = 0                                  #power level (in dBm)
        self.configured = 'not configured'              #indicates whether or not it is configured
        #set up U.I. based on number:
        #labels
        # ttk.Label(parent, text = 'PLL ' + str(number) + ': ' + msg).grid(column = 0, row = (2*(number-1)), sticky = (W), columnspan =7)
        ttk.Label(parent, text = 'PLL ' + str(number) + ':').grid(column = 0, row = (2*(number-1)), sticky = (N,S,E))
        ttk.Label(parent, text = msg).grid(column = 1, row = (2*(number-1)), columnspan = 6, sticky = (N,S,W))
        ttk.Label(parent, text = 'MHz',).grid(column = 1, row = 1 + 2*(number-1), sticky = W)
        ttk.Label(parent, text = 'dBm').grid(column = 3, row = 1 + 2*(number-1), sticky = W)
        #configured label
        self.configMsg = ttk.Label(parent, text = self.configured, foreground = 'red',  anchor = E)
        self.configMsg.grid(column = 4, row = 1 + 2*(number-1), sticky=(E,W))
        #configure button
        self.configBtn = ttk.Button(parent, text='configure', command = self.configureButtonPress)
        self.configBtn.grid(column = 5, row = 1 + 2*(number-1))
        #re-cal button
        self.recalBtn = ttk.Button(parent, text = 're-cal', command = self.recalButtonPress)
        self.recalBtn.grid(column = 6, row = 1 + 2*(number-1))
        #frequency entry
```

```
        self.freqEntry = ttk.Entry(parent, width = 6, validate = 'focusout', validatecommand = self.freqStringValidate)
        self.freqEntry.insert(0,str(self.freq))
        self.freqEntry.grid(column = 0, row = 1 + 2*(number-1), sticky = (W))
        #power level (spinbox)
        self.powerEntry = Spinbox(parent, value = ('-9', '-6', '-3', '0'), width=2, command = self.powerLevelChange)
        self.powerEntry.delete(0,'end')
        self.powerEntry.insert(0,'0')
        self.powerEntry.grid(column = 2, row = 1 + 2*(number-1), sticky = E)
        #configure row and column weights
        parent.rowconfigure(2 * (number-1), weight = 1)
        parent.rowconfigure(1 + 2 * (number-1), weight = 1)
        parent.columnconfigure(0, weight = 0)
        parent.columnconfigure(1, weight = 0)
        parent.columnconfigure(2, weight = 0)
        parent.columnconfigure(3, weight = 0)
        parent.columnconfigure(4, weight = 1)
        parent.columnconfigure(5, weight = 0)
        parent.columnconfigure(6, weight = 0)

    #config button pressed
    def configureButtonPress(self):
        #determine Odiv and N
        if self.freq >= 4200:
            Odiv = 1
        elif self.freq >= 2100:
            Odiv = 2
        elif self.freq >= 1400:
            Odiv = 3
        elif self.freq >= 1050:
            Odiv = 4
        elif self.freq >= 840:
            Odiv = 5
        elif self.freq >= 700:
            Odiv = 6
        N = self.freq * Odiv
        #print message to console
        print('PLL #' + str(self.number))
        print('N: ' + str(N))
        print('Odiv: ' + str(Odiv))
        print('dBm: ' + str(self.dict_power[self.power]))
        #construct message (Bytes):
        #[0] = CMD, PLL#
        #[1] = dBm, Odiv
        #[2] = N (upper)
        #[4] = N (lower)
        payload = [0, 0, 0, 0, 0, 0, 0] #7 nibbles long
        payload[0] = self.number
        payload[1] = self.dict_power[self.power]
        payload[2] = Odiv
        payload[3] = (N & 0xF000) >> 12
        payload[4] = (N & 0x0F00) >> 8
        payload[5] = (N & 0x00F0) >> 4
        payload[6] = (N & 0x000F)
```

```
        SerialComms.HARA.command(cmd = 'PLL_RECONFIGURE', payload = payload)

        #update configured message
        self.configured = 'configured'
        self.configMsg['text'] = self.configured
        self.configMsg['foreground'] = 'green'

    #recalibrate button pressed
    def recalButtonPress(self):
        #TODO: UART comms
        print('*** PLL Recal ***')
        print('PLL #' + str(self.number))
        SerialComms.HARA.command(cmd = 'PLL_RECAL', payload = (0, 0, self.number))


    #makes sure the frequency is set to an integer multiple of 1 MHz
    def freqStringValidate(self):
        freq = int(float(self.freqEntry.get()))
        if freq > 6390:
            freq = 6390
        if freq < 700:
            freq = 700
        self.freqEntry.delete(0,'end')
        self.freqEntry.insert(0,str(freq))
        #check if new value was entered
        if freq != self.freq:
            self.configured = 'not configured'
            self.configMsg['text'] = self.configured
            self.configMsg['foreground'] = 'red'
            self.freq = freq
        return 1

    #power level was changed
    def powerLevelChange(self):
        self.powerLevel = int(self.powerEntry.get())
        if self.powerLevel != self.power:
            self.configured = 'not configured'
            self.configMsg['text'] = self.configured
            self.configMsg['foreground'] = 'red'
            self.power = self.powerLevel


#to create PLL tab, simply instantiate 3 PLLs within a frame!

class PFD:
    #dictionary of mux switch values
    dict_mux = {     '3-State'                              : 0,
                     'Digital Lock Detect'                  : 1,
                     'N Divider Output'                     : 2,
                     'DVdd'                                 : 3,
                     'R Divider Output'                     : 4,
                     'N-Channel Open Drain Lock Detect'     : 5,
                     'Serial Data Out'                      : 6,
                     'DGnd'                                 : 7
                     }
```

```python
    dict_apbw = {   '2.9 ns' : 0,
                    '6.0 ns' : 2
                    }
    dict_icp = {    '0.625 mA' : 0,
                    '1.250 mA' : 1,
                    '1.875 mA' : 2,
                    '2.500 mA' : 3,
                    '3.125 mA' : 4,
                    '3.750 mA' : 5,
                    '4.375 mA' : 6,
                    '5.000 mA' : 7
                    }
    dict_polarity = {'+' : 1,
                     '-' : 0
                    }
    dict_ele = {
        '1' : [1],
        '2' : [2],
        '3' : [3],
        'all' : [1, 2, 3],
    }

    def __init__(self, parent):
        #separator
        ttk.Separator(parent, orient = HORIZONTAL).grid(row = 4, column = 0, columnspan = 7, sticky = (E,W))
        #Label:
        ttk.Label(parent, text = "PFD:").grid(row = 5, column = 0, columnspan = 1, sticky = (N,S,E))
        self.configMsg = ttk.Label(parent, text = 'not configured', foreground = 'red', anchor = E)
        self.configMsg.grid(row = 5, column = 4, columnspan = 1, sticky = (N,S,E,W))
        #PFD element selector (spinbox)
        self.elementSelect = Spinbox(parent, value = ('1', '2', '3', 'all'), width=2)
        self.elementSelect.delete(0,'end')
        self.elementSelect.insert(0,'all')
        self.elementSelect.grid(row = 5, column = 1, sticky = (N,S,W))
        #configure button
        ttk.Button(parent, text = 'Send Message', command = self.PFDmsg).grid(row = 5, column = 5, columnspan = 2, sticky = (E,W))
        #Mux out:
        ttk.Label(parent, text = "Mux:").grid(row = 6, column = 0, columnspan = 1, sticky = (E))
        self.muxSelectCombobox = ttk.Combobox(parent, values = ('N Divider Output', 'R Divider Output', 'Digital Lock Detect', 'N-Channel Open Drain Lock Detect', 'DVdd', 'DGnd', '3-State', 'Serial Data Out'))
        self.muxSelectCombobox.bind('<<ComboboxSelected>>',self.ComboBoxChange)
        self.muxSelectCombobox.set('Digital Lock Detect')
        self.muxSelectCombobox.grid(row = 6, column = 1, columnspan = 4, sticky = (N,S,E,W))
        #Polarity
```

```
        ttk.Label(parent, text = "CP Polarity:").grid(row = 6, column = 5, sticky = (N,S,E))
        self.polaritySelectCombobox = ttk.Combobox(parent, values = ('+','-'), width = 2)
        self.polaritySelectCombobox.bind('<<ComboboxSelected>>',self.ComboBoxChange)
        self.polaritySelectCombobox.set('-')
        self.polaritySelectCombobox.grid(row = 6, column = 6, sticky = W)
        #Charge Pump Current
        ttk.Label(parent, text = "Icp:").grid(row = 7, column = 0, sticky = (E))
        self.icpSelectCombobox = ttk.Combobox(parent, values = ('0.625 mA','1.250 mA','1.875 mA','2.500 mA','3.125 mA','3.750 mA','4.375 mA','5.000 mA'), width = 8)
        self.icpSelectCombobox.bind('<<ComboboxSelected>>',self.ComboBoxChange)
        self.icpSelectCombobox.set('0.625 mA')
        self.icpSelectCombobox.grid(row = 7, column = 1, columnspan = 2, sticky = (N,S,E,W))
        #Anti-Backlask Pulse Width
        ttk.Label(parent, text = 'Anti-Backlash Pulse Width:').grid(row = 7, column = 3, columnspan = 3, sticky = (N,S,E))
        self.apbwSelectCombobox = ttk.Combobox(parent, values = ('2.9 ns','6.0 ns'), width = 5)
        self.apbwSelectCombobox.bind('<<ComboboxSelected>>',self.ComboBoxChange)
        self.apbwSelectCombobox.set('6.0 ns')
        self.apbwSelectCombobox.grid(row = 7, column = 6, columnspan = 1, sticky = (N,S,E,W))
        #R divider selection
        ttk.Label(parent, text = 'R Div:').grid(row = 8, column = 0, columnspan = 1, sticky = (N,S,E))
        self.rDivEntry = ttk.Entry(parent, width = 6)
        self.rDivEntry.insert(0,str(50))
        self.rDivEntry.grid(row = 8, column = 1, columnspan = 2, sticky = (W))
        #N divider selection
        ttk.Label(parent, text = 'N Div:').grid(row = 8, column = 3, columnspan = 1, sticky = (N,S,E))
        self.nDivEntry = ttk.Entry(parent, width = 6)
        self.nDivEntry.insert(0,str(50))
        self.nDivEntry.grid(row = 8, column = 4, columnspan = 1, sticky = (W))


    def PFDmsg(self):
        #construct and send message
        for ele in self.dict_ele[self.elementSelect.get()]:
            # ele = int(self.elementSelect.get())
            mux = self.dict_mux[self.muxSelectCombobox.get()]
            icp = self.dict_icp[self.icpSelectCombobox.get()]
            apbw = self.dict_apbw[self.apbwSelectCombobox.get()]
            pol = self.dict_polarity[self.polaritySelectCombobox.get()]
            msg = [0,0,0,0,0,0,0,0,0]
            msg[0] = (ele << 1) | ((mux & 0x04) >> 2)
            msg[1] = ((mux & 0x03) << 2) | ((icp & 0x06) >> 1)
            msg[2] = ((icp & 0x01) << 3) | (apbw << 1) | (pol)
```

```
            #R divider - 14 bit
            r = int(float(self.rDivEntry.get()))
            msg[3] = (r & (0x0300)) >> 8
            msg[4] = (r & (0x00F0)) >> 4
            msg[5] = (r & (0x000F))
            #N divider - 13 bit
            n = int(float(self.nDivEntry.get()))
            msg[6] = (n & (0x0100)) >> 8
            msg[7] = (n & (0x00F0)) >> 4
            msg[8] = (n & (0x000F))
            SerialComms.HARA.command('PFD_RECONFIGURE',msg)
        #update configured message
        self.configMsg['text'] = 'configured'
        self.configMsg['foreground'] = 'green'
        #update entry fields
        self.nDivEntry.delete(0,'end')
        self.nDivEntry.insert(0,str(n))
        self.rDivEntry.delete(0,'end')
        self.rDivEntry.insert(0,str(r))

    def ComboBoxChange(self, other):
        self.configMsg['text'] = 'not configured'
        self.configMsg['foreground'] = 'red'
        self.muxSelectCombobox.selection_clear()
        self.icpSelectCombobox.selection_clear()
        self.apbwSelectCombobox.selection_clear()
        self.polaritySelectCombobox.selection_clear()
```

A.5.5 CMD_tab.py

```python
from tkinter import *
from tkinter import ttk
from functools import partial
import threading
import time
import SerialComms

class CMD_tab:
    #variables:
    elementSelect = 1               # currently selected element (int)
    phaseOffset = []                # StringVar containing currently selected phase offset
    point = []                      # StringVar containing currently selected point angle
    #flags
    activeSteer = False                 # flag to indicate if we are in active steer mode
    retrodirect = False                 # flag to indicate if we are in retrodirect mode

    dict_ele = {
        '1' : [1],
        '2' : [2],
        '3' : [3],
        'all' : [1,2,3],
        }

    def __init__(self, parent):
        ########### calibration commands    ##########
        #labels
        ttk.Label(parent, text = 'Calibration:').grid(column = 0, row = 0, sticky = W)

        #####   Set delta_i/j by CP   #####
        #button
        self.setDeltaBtn = ttk.Button(parent, text = 'Set ' + u'\u0394' + 'i by CP', command = self.setDeltaByCP)
        self.setDeltaBtn.grid(row = 1, column = 0, columnspan = 2, sticky = (E,W))

        #####   spin (element select list)   #####
        ttk.Label(parent, text = 'Element:',anchor = E).grid(column = 2, row = 1, sticky = E)
        self.elementSelectCombobox = Spinbox(parent, values = ('1','2','3','all'), width = 4)
        self.elementSelectCombobox.delete(0,'end')
        self.elementSelectCombobox.insert(0,'all')
        self.elementSelectCombobox.grid(column = 3, row = 1, columnspan = 1, sticky = (E,W))
#         self.elementSelectCombobox.bind('<<ComboboxSelected>>',self.elementSelectChange)

        #####   Adjust delta_i manually #####
```

```
        ttk.Label(parent, text = 'Adjust ' + u'\u0394' + 'i:').grid(row=2, column=0)
        f = ttk.Frame(parent)
        f.grid(row=2, column=1, columnspan=3, sticky=(E,W))
        adjustments = [-10, -5, -1, 1, 5, 10]
        for i in [0, 1, 2, 3, 4, 5]:
            action = partial(self.adjustDelta, 'i', adjustments[i])
            ttk.Button(f, text=str(adjustments[i])+u'\u00b0', command=action, width=3.5).grid(row=0, column=i, sticky=(N,S,E,W))
            f.columnconfigure(i, weight=1)

        #####   Adjust delta_j manually ######
        ttk.Label(parent, text = 'Adjust ' + u'\u0394' +'j:').grid(row=3, column=0)
        f = ttk.Frame(parent)
        f.grid(row=3, column=1, columnspan=3, sticky=(E,W))
        for i in [0, 1, 2, 3, 4, 5]:
            action = partial(self.adjustDelta, 'j', adjustments[i])
            ttk.Button(f, text = str(adjustments[i]) + u'\u00b0', command = action, width=3.5).grid(row=0, column=i, sticky=(N,S,E,W))
            f.columnconfigure(i, weight=1)

        ########### Operational commands    ##########
        ttk.Separator(parent, orient = HORIZONTAL).grid(row = 4, column = 0, columnspan = 4, sticky = (E,W))
        ttk.Label(parent,text = 'Operation:', anchor = W).grid(row = 5, column = 0, columnspan = 4, sticky = (E,W))

        #####   Active Steer    #####
        self.point = StringVar()    #angle to point array at
        self.point.set('0')         #initialize point to 0 degrees
        #button
        self.activeSteerBtn = ttk.Button(parent, text = 'Active Steer', command = self.activeSteerBtnPress)
        self.activeSteerBtn.grid(row = 6, column = 0, sticky = (E,W))
        #label
        ttk.Label(parent, text = 'Point:', anchor = E).grid(row = 6, column = 1, sticky = (E,W))
        #scale (slider bar)
        self.pointScale = ttk.Scale(parent, orient = HORIZONTAL, length = 180, from_ = -90, to = 90, variable = self.point, command = self.pointScaleChange)
        self.pointScale.state(['disabled'])
        self.pointScale.grid(row = 6, column = 2, sticky = (E,W))
        #spinbox
        self.pointSpinbox = Spinbox(parent, from_ = -90, to = 90, increment = 1, width = 4, textvariable = self.point, state = 'disabled')
        self.pointSpinbox.grid(row = 6, column = 3, sticky = (E,W))

        #####   Retrodirect #####
        #button
        self.retrodirectBtn = ttk.Button(parent, text = 'Retrodirect', command = self.retrodirectBtnPress)
        self.retrodirectBtn.grid(row = 7, column = 0, sticky = (E,W))

```

```
        #####   Current State Label ######
        ttk.Label(parent, text = 'State:', anchor = E).grid(row = 7, column = 1, sticky = (N,S,E,W))
        self.stateLabel = ttk.Label(parent, text = 'None')
        self.stateLabel.grid(row = 7, column = 2, columnspan = 2, sticky = (N,S,E,W))

        ######  Thread to send commands ######
        cmdThread = threading.Thread(target = self.cmd, daemon = True)
        cmdThread.start()

        #####   Configure row/column weights    #####
        parent.columnconfigure(2, weight = 1)
        parent.rowconfigure(4, weight = 1)


    # #elementSelectChange: updates selected element pointer
    # def elementSelectChange(self, val):
        # #update elementSelect and clear highlight in box
        # self.elementSelect = self.elementSelectCombobox.get()
        # self.elementSelectCombobox.selection_clear()

    #adjustDelta: sends command to update delta_i/j by a given value
    def adjustDelta(self, delta, value):
        for ele in self.dict_ele[self.elementSelectCombobox.get()]:
            msg = [0, 0, 0]
            msg[0] = (ele << 2)
            if delta == 'j':
                print('j')
                msg[0] = msg[0] | 0x01
            else:
                print('i')
            msg[1] = (value & 0x00F0) >> 4
            msg[2] = (value & 0x000F)
            SerialComms.HARA.command('ADJUST_DELTA_MANUALLY', msg)


    #setDeltaByCP: sends message to use current CP value to set delta_i and delta_j
    def setDeltaByCP(self):
        for ele in self.dict_ele[self.elementSelectCombobox.get()]:
            print("HARA #" + str(ele) + " = 0")
            msg = [0,0,0]
            msg[0] = (ele << 2)
            SerialComms.HARA.command('SET_DELTAS_BY_CP',msg)

    #activeSteerBtnPress: disables other buttons and begins sending active steer messages
    def activeSteerBtnPress(self):
        #if we are not currently in active steer mode, enter it
        if not self.activeSteer:
            #enable the point selection widgets
            self.pointScale.state(['!disabled'])
            self.pointSpinbox['state'] = 'normal'
            #change button text
            self.activeSteerBtn['text'] = 'Stop Steering'
            #disable all other buttons/widgets on screen
```

```
            self.elementSelectCombobox['state'] = 'disabled'
            self.setDeltaBtn['state'] = 'disabled'
            self.retrodirectBtn['state'] = 'disabled'
            #change flag
            self.activeSteer = True
        else:
            #disable the point selection widgets
            self.pointScale.state(['disabled'])
            self.pointSpinbox['state'] = 'disabled'
            self.point.set('0')
            #change button text
            self.activeSteerBtn['text'] = 'Active Steer'
            #reenable all other buttons/widgets on screen
            self.elementSelectCombobox['state'] = 'readonly'
            self.setDeltaBtn['state'] = 'normal'
            self.retrodirectBtn['state'] = 'normal'
            #change flag
            self.activeSteer = False
            #send 'stop active steer' command
            SerialComms.HARA.command('STOP_RETRODIRECT')

    #pointScaleChange: rounds value from Active Steer slider when it changes
    def pointScaleChange(self, val):
        #round the string to an integer
        self.point.set(int(float(val)))

    def retrodirectBtnPress(self):
        #if we aren't retrodirecting, start
        if not self.retrodirect:
            #change button text
            self.retrodirectBtn['text'] = 'Stop Retrodirection'
            #disable all other buttons/widgets on screen
            self.elementSelectCombobox['state'] = 'disabled'
            self.setDeltaBtn['state'] = 'disabled'
            self.activeSteerBtn['state'] = 'disabled'
            #send 'Retrodirect' command
            SerialComms.HARA.command('RETRODIRECT')
            #change flag
            self.retrodirect = True
        else:
            #change button text
            self.retrodirectBtn['text'] = 'Retrodirect'
            #reenable all other buttons/widgets on screen
            self.elementSelectCombobox['state'] = 'readonly'
            self.setDeltaBtn['state'] = 'normal'
            self.activeSteerBtn['state'] = 'normal'
            #send 'Stop Retrodirect' command
            SerialComms.HARA.command('STOP_RETRODIRECT')
            #change flag
            self.retrodirect = False

    #flags
    # activeSteer = False         # flag to indicate if we are in active steer mode
    # retrodirect = False         # flag to indicate if we are in retrodirect mode
```

```
    def cmd(self):
        while True:
            state = SerialComms.HARA.state
            #if 'Active steer' was pressed...
            if self.activeSteer:
                point = int(float(self.point.get()))
                msg = [0,0,0]
                msg[0] = (point & 0x0F00) >> 8
                msg[1] = (point & 0x00F0) >> 4
                msg[2] = (point & 0x000F)
                SerialComms.HARA.command('ACTIVE_STEER',msg)
            #NOTE: 'Stop Active Steer' command is handled in its button handler
            #NOTE: 'Retrodirect'/'Stop Retrodirecting' command is handled in its button handler
            #update the stateLabel
            self.stateLabel['text'] = {
                    0x00 : 'None',
                    0x01 : 'Finding Phase Offset',
                    0x04 : 'Active Steering',
                    0x05 : 'Retrodirecting',
                }.get(state, 'ERR')
            #sleep for half a second
            time.sleep(0.5)
```

A.5.6 Phase_tab.py

```python
from tkinter import *
from tkinter import ttk
from functools import partial
import SerialComms

class Phase_tab:
    RxSpinBox = [0,0,0,0]
    TxSpinBox = [0,0,0,0]

    def __init__(self, parent):
        f1 = ttk.Frame(parent, padding = (5,5))
        f1.grid(row = 0, column = 0, columnspan = 3, rowspan = 3, sticky = (N,S,E,W))
        f1.rowconfigure(0, weight = 1)
        f1.columnconfigure(0, weight = 1)
        f = ttk.Frame(f1, borderwidth = 1, relief = 'solid', padding = (5, 5))
        f.grid(row = 0, column = 0, columnspan = 3, rowspan = 3, sticky = (N,S,E,W))
        f.rowconfigure(0, weight = 1)
        f.columnconfigure(0, weight = 1)
        ttk.Label(f, text = '0'+ u'\u00b0'+' - 359'+ u'\u00b0', anchor = 'center').grid(row = 0, column = 0, sticky = (N,S,E,W))
        for i in [1,2,3]:
            #labels
            ttk.Label(parent, text = 'HARA #' + str(i)).grid(row = 3 * int(i / 2), column = 3 * int(i % 2), columnspan = 3, sticky = (N,S,E,W))
            ttk.Label(parent, text = 'Rx:').grid(row = 1 + (3*int(i/2)), column = 3*int(i%2), sticky = (N,S,E,W))
            ttk.Label(parent, text = 'Tx:').grid(row = 2 + (3*int(i/2)), column = 3*int(i%2), sticky = (N,S,E,W))
            #spinboxes
            self.RxSpinBox[i] = Spinbox(parent, from_ = 0, to = 359, increment = 1, width = 4)
            self.RxSpinBox[i].grid(row = 1 + (3*int(i/2)), column = 1 + (3*int(i%2)), sticky = (N,S,E,W))
            self.TxSpinBox[i] = Spinbox(parent, from_ = 0, to = 359, increment = 1, width = 4)
            self.TxSpinBox[i].grid(row = 2 + (3*int(i/2)), column = 1 + (3*int(i%2)), sticky = (N,S,E,W))
            #write btutons
            ttk.Button(parent, text = 'Send', command = partial(self.sendCommand, i, 'Rx')).grid(row = 1 + (3*int(i/2)), column = 2 + (3*int(i%2)), sticky = (N,S,E,W))
            ttk.Button(parent, text = 'Send', command = partial(self.sendCommand, i, 'Tx')).grid(row = 2 + (3*int(i/2)), column = 2 + (3*int(i%2)), sticky = (N,S,E,W))
        for i in [0,1,2,3,4,5]:
            parent.rowconfigure(i, weight = 1)
            parent.columnconfigure(i, weight = 1)


    def sendCommand(self, num, ps):
```

```
        if ps is 'Rx':
            phaseShifter = 0
            phase = int(self.RxSpinBox[num].get())
        else:
            phaseShifter = 1
            phase = int(self.TxSpinBox[num].get())
        payload = [0,0,0] #3 nibbles long
        payload[0] = ((num & 0x03) << 2) | (phaseShifter << 1) | ((phase & 0x100) >> 8)
        payload[1] = (phase & 0x0F0) >> 4
        payload[2] = (phase & 0x00F)
        print('HARA #' + str(num))
        print('Phase Shifter: ' + ps)
        print(str(phase))
        SerialComms.HARA.command(cmd = 'PHASE_WRITE', payload = payload)
```

A.5.7 ManualTab.py

```
from tkinter import *
from tkinter import ttk
from functools import partial
import threading
import time
import SerialComms

class Manual_tab:

    VcalSpinBox = [0,0,0,0]
    ThetaSpinBox = [0,0,0,0]

    def __init__(self, parent):
        f1 = ttk.Frame(parent, padding = (5,5))
        f1.grid(row = 0, column = 0, columnspan = 3, rowspan = 3, sticky = (N,S,E,W))
        f1.rowconfigure(0, weight = 1)
        f1.columnconfigure(0, weight = 1)
        f = ttk.Frame(f1, borderwidth = 1, relief = 'solid', padding = (5, 5))
        f.grid(row = 0, column = 0, columnspan = 3, rowspan = 3, sticky = (N,S,E,W))
        f.rowconfigure(0, weight = 1)
        f.columnconfigure(0, weight = 1)
        ttk.Label(f, text = '0 - 4095', anchor = 'center').grid(row = 0, column = 0, sticky = (N,S,E,W))
        for i in [1,2,3]:
            #labels
            ttk.Label(parent, text = 'HARA #' + str(i)).grid(row = 3 * int(i / 2), column = 3 * int(i % 2), columnspan = 3, sticky = (N,S,E,W))
            ttk.Label(parent, text = 'Rx:').grid(row = 1 + (3*int(i/2)), column = 3*int(i%2), sticky = (N,S,E,W))
            ttk.Label(parent, text = 'Tx:').grid(row = 2 + (3*int(i/2)), column = 3*int(i%2), sticky = (N,S,E,W))
            #spinboxes
            self.VcalSpinBox[i] = Spinbox(parent, from_ = 0, to = 4095, increment = 100, width = 4)
            self.VcalSpinBox[i].grid(row = 1 + (3*int(i/2)), column = 1 + (3*int(i%2)), sticky = (N,S,E,W))
            self.ThetaSpinBox[i] = Spinbox(parent, from_ = 0, to = 4095, increment = 100, width = 4)
            self.ThetaSpinBox[i].grid(row = 2 + (3*int(i/2)), column = 1 + (3*int(i%2)), sticky = (N,S,E,W))
            #write buttons
            ttk.Button(parent, text = 'Send', command = partial(self.sendCommand, i, 'Rx')).grid(row = 1 + (3*int(i/2)), column = 2 + (3*int(i%2)), sticky = (N,S,E,W))
            ttk.Button(parent, text = 'Send', command = partial(self.sendCommand, i, 'Tx')).grid(row = 2 + (3*int(i/2)), column = 2 + (3*int(i%2)), sticky = (N,S,E,W))
        for i in [0,1,2,3,4,5]:
            parent.rowconfigure(i, weight = 1)
            parent.columnconfigure(i, weight = 1)
```

```python


    def sendCommand(self, num, dac):
        if dac is 'Rx':
            dacType = 0
            dacVal = int(self.VcalSpinBox[num].get())
        else:
            dacType = 1
            dacVal = int(self.ThetaSpinBox[num].get())
        payload = [0,0,0,0,0,0,0] #7 nibbles long
        payload[0] = num
        payload[1] = dacType    #0 for Vcal, 1 for theta
        payload[2] = 0
        payload[3] = (dacVal & 0xF000) >> 12
        payload[4] = (dacVal & 0x0F00) >> 8
        payload[5] = (dacVal & 0x00F0) >> 4
        payload[6] = (dacVal & 0x000F)
        SerialComms.HARA.command(cmd = 'DAC_WRITE', payload = payload)
```

Appendix B

Phase Shifter Characterization Files

B.1 Characterization Test Source Code

B.1.1 PhaseShifterCharacterization.py

```
import pyautogui
import time
import datetime
import sys, getopt


stepsize = 40
freq = 2410
mode = 'Rx'
phaseShifter = 'NOSW50ohm'
ref = 100

#constants
mouseTime = 0.5
keyTime = 0.15
sleepTime = 1.0
size = pyautogui.size()
#locations (config tab)
config_tab = (980, 235)
freq_rx = (990, 280)
config_rx = (1305, 280)
freq_tx = (990, 330)
config_tx = (1305, 330)
#locations (manual phase tab)
manual_phase_tab = (1210, 235)
rx_1 = (1285, 285)
tx_1 = (1285, 325)
rx_2 = (1055, 385)
tx_2 = (1055, 425)
rx_3 = (1285, 385)
tx_3 = (1285, 425)


#move GUI window to the right spot
pyautogui.click(x=420, y=1060, duration=mouseTime)
```

```python
pyautogui.keyDown('alt')
pyautogui.press('space')
pyautogui.keyUp('alt')
pyautogui.press('x')
pyautogui.moveTo(size[0]/2, 5, mouseTime)
pyautogui.dragTo(1400, 50, mouseTime, button='left')
time.sleep(sleepTime)

#set frequency
pyautogui.click(x = config_tab[0], y = config_tab[1], duration=mouseTime) #config tab
if (mode is 'Rx'):
    pyautogui.click(x=freq_rx[0], y=freq_rx[1], duration=mouseTime) #Downconversion freq
else:
    pyautogui.click(x=freq_tx[0], y=freq_tx[1], duration=mouseTime) #Downconversion freq
pyautogui.press(['backspace','backspace','backspace','backspace','backspace'], interval = keyTime)
pyautogui.typewrite('%0.0d' % freq, interval = keyTime)
if (mode is 'Rx'):
    pyautogui.click(x=config_rx[0], y=config_rx[1], duration=mouseTime) #configure
else:
    pyautogui.click(x=config_tx[0], y=config_tx[1], duration=mouseTime) #configure
time.sleep(sleepTime)

#select Manual (DAC) tab
pyautogui.click(x=manual_phase_tab[0], y=manual_phase_tab[1], duration=mouseTime)
#write 0 to all DACs
if (mode is 'Rx'):
    loop = [rx_1, rx_2, rx_3]
else:
    loop = [tx_1, tx_2, tx_3]
for loc in loop:
    pyautogui.click(x=loc[0], y=loc[1], duration = mouseTime)
    pyautogui.press(['backspace','backspace','backspace','backspace','backspace'], interval = keyTime)
    pyautogui.typewrite('%0.0d' % 0, interval = keyTime)
    pyautogui.click(x = loc[0] + 100, y = loc[1], duration = mouseTime)

time.sleep(sleepTime)

#create folder
folder = mode + 'Tester_' + str(phaseShifter) + '_' + str(freq) + 'MHz_' + str(ref) + 'MHz_' + str(stepsize) + '_big'
print(folder)
pyautogui.click(x=15, y=295, duration=mouseTime)    #File
pyautogui.click(x=90, y=385, duration=mouseTime)    #File Utilities
pyautogui.click(x=515, y=455, duration=mouseTime)   #Look in:
time.sleep(0.5)
```

```
pyautogui.press(['backspace','backspace','backspace','backspace','backspace'], interval = keyTime)
pyautogui.typewrite('E:/', interval = keyTime)      #E:/
pyautogui.press('enter')
pyautogui.click(x=710, y=455, duration=mouseTime)   #New Folder
time.sleep(0.5)
pyautogui.typewrite(folder, interval = keyTime)     #Folder Name
pyautogui.click(x=505, y=590, duration=mouseTime)   #OK
pyautogui.click(x=810, y=750, duration=mouseTime)   #deselect window
time.sleep(sleepTime)

#test loop
num = 1
if mode is 'Rx':
    loop = [rx_1, rx_2, rx_3]
else:
    loop = [tx_1, tx_2, tx_3]

#iterate up (1)
num = 1
for loc in loop:
    #iterate through DAC values
    for dac in range(0, 4096, stepsize):
        #write new DAC vlaue
        pyautogui.click(x=loc[0], y=loc[1], duration = mouseTime)
        pyautogui.press(['backspace','backspace','backspace','backspace','backspace'], interval = keyTime)
        pyautogui.typewrite('%0.0d' % dac, interval = keyTime)
        pyautogui.click(x=loc[0]+100, y=loc[1], duration=mouseTime)
        #wait 2 seconds for settling out
        time.sleep(2)
        #change horizontal scale twice to start the 10k measurements
        pyautogui.click(x = 690, y = 810, duration = mouseTime)
        time.sleep(0.5)
        pyautogui.click(x = 665, y = 790, duration = mouseTime)
        time.sleep(0.5)
        pyautogui.click(x = 715, y = 790, duration = mouseTime)
        #wait for measurements
        time.sleep(25)
        #save file
        pyautogui.click(x = 100, y = 550, duration = mouseTime, button = 'right')
        pyautogui.click(x = 200, y = 585, duration = mouseTime)
        pyautogui.click(x = 200, y = 600, duration = mouseTime)
        time.sleep(0.5)
        pyautogui.typewrite(('E:/'+folder), interval = keyTime)
        pyautogui.click(x = 200, y = 625, duration = mouseTime)
        time.sleep(0.5)
        pyautogui.typewrite('DAC%0.0d_%0.0d' % (num, dac), interval = keyTime)
        pyautogui.click(x = 490, y = 765, duration = mouseTime)
        time.sleep(sleepTime)
    #set back to 0
    pyautogui.click(x=loc[0], y=loc[1], duration = mouseTime)
```

```
    pyautogui.press(['backspace','backspace','backspace','backspace','backspace'], interval = keyTime)
    pyautogui.typewrite('%0.0d' % 0, interval = keyTime)
    pyautogui.click(x=loc[0]+100, y=loc[1], duration=mouseTime)
    num = num + 1
print(datetime.datetime.now())
print("Hey look mom, I made it (1) ")

#iterate down (2)
num = 1
for loc in loop:
    #iterate through DAC values
    for dac in range(0, 4095, stepsize):
        #write new DAC vlaue
        pyautogui.click(x=loc[0], y=loc[1], duration = mouseTime)
        pyautogui.press(['backspace','backspace','backspace','backspace','backspace'], interval = keyTime)
        value_dac = 4095 - dac
        pyautogui.typewrite('%0.0d' % value_dac, interval = keyTime)
        pyautogui.click(x=loc[0]+100, y=loc[1], duration=mouseTime)
        #wait 2 seconds for settling out
        time.sleep(2)
        #change horizontal scale twice to start the 10k measurements
        pyautogui.click(x = 690, y = 810, duration = mouseTime)
        time.sleep(0.5)
        pyautogui.click(x = 665, y = 790, duration = mouseTime)
        time.sleep(0.5)
        pyautogui.click(x = 715, y = 790, duration = mouseTime)
        #wait for measurements
        time.sleep(25)
        #save file
        pyautogui.click(x = 100, y = 550, duration = mouseTime, button = 'right')
        pyautogui.click(x = 200, y = 585, duration = mouseTime)
        pyautogui.click(x = 200, y = 600, duration = mouseTime)
        time.sleep(0.5)
        pyautogui.typewrite(('E:/'+folder), interval = keyTime)
        pyautogui.click(x = 200, y = 625, duration = mouseTime)
        time.sleep(0.5)
        pyautogui.typewrite('DAC%0.0d_%0.0d' % (num, value_dac), interval = keyTime)
        pyautogui.click(x = 490, y = 765, duration = mouseTime)
        time.sleep(sleepTime)
    #set back to 0
    pyautogui.click(x=loc[0], y=loc[1], duration = mouseTime)
    pyautogui.press(['backspace','backspace','backspace','backspace','backspace'], interval = keyTime)
    pyautogui.typewrite('%0.0d' % 0, interval = keyTime)
    pyautogui.click(x=loc[0]+100, y=loc[1], duration=mouseTime)
    num = num + 1
print(datetime.datetime.now())
print("Hey look mom, I made it (2) ")

#iterate up (3)
```

```
num = 1
for loc in loop:
    #iterate through DAC values
    for dac in range(2, 4095, stepsize):
        #write new DAC vlaue
        pyautogui.click(x=loc[0], y=loc[1], duration = mouseTime)
        pyautogui.press(['backspace','backspace','backspace','backspace','backspace'], interval = keyTime)
        pyautogui.typewrite('%0.0d' % dac, interval = keyTime)
        pyautogui.click(x=loc[0]+100, y=loc[1], duration=mouseTime)
        #wait 2 seconds for settling out
        time.sleep(2)
        #change horizontal scale twice to start the 10k measurements
        pyautogui.click(x = 690, y = 810, duration = mouseTime)
        time.sleep(0.5)
        pyautogui.click(x = 665, y = 790, duration = mouseTime)
        time.sleep(0.5)
        pyautogui.click(x = 715, y = 790, duration = mouseTime)
        #wait for measurements
        time.sleep(25)
        #save file
        pyautogui.click(x = 100, y = 550, duration = mouseTime, button = 'right')
        pyautogui.click(x = 200, y = 585, duration = mouseTime)
        pyautogui.click(x = 200, y = 600, duration = mouseTime)
        time.sleep(0.5)
        pyautogui.typewrite(('E:/'+folder), interval = keyTime)
        pyautogui.click(x = 200, y = 625, duration = mouseTime)
        time.sleep(0.5)
        pyautogui.typewrite('DAC%0.0d_%0.0d' % (num, dac), interval = keyTime)
        pyautogui.click(x = 490, y = 765, duration = mouseTime)
        time.sleep(sleepTime)
    #set back to 0
    pyautogui.click(x=loc[0], y=loc[1], duration = mouseTime)
    pyautogui.press(['backspace','backspace','backspace','backspace','backspace'], interval = keyTime)
    pyautogui.typewrite('%0.0d' % 0, interval = keyTime)
    pyautogui.click(x=loc[0]+100, y=loc[1], duration=mouseTime)
    num = num + 1
print(datetime.datetime.now())
print("Hey look mom, I made it (3) ")

#iterate down (4)
num = 1
for loc in loop:
    #iterate through DAC values
    for dac in range(0, 4087, stepsize):
        #write new DAC vlaue
        pyautogui.click(x=loc[0], y=loc[1], duration = mouseTime)
        pyautogui.press(['backspace','backspace','backspace','backspace','backspace'], interval = keyTime)
        value_dac = 4087 - dac
        pyautogui.typewrite('%0.0d' % value_dac, interval = keyTime)
```

```
        pyautogui.click(x=loc[0]+100, y=loc[1], duration=mouseTime)
        #wait 2 seconds for settling out
        time.sleep(2)
        #change horizontal scale twice to start the 10k measurements
        pyautogui.click(x = 690, y = 810, duration = mouseTime)
        time.sleep(0.5)
        pyautogui.click(x = 665, y = 790, duration = mouseTime)
        time.sleep(0.5)
        pyautogui.click(x = 715, y = 790, duration = mouseTime)
        #wait for measurements
        time.sleep(25)
        #save file
        pyautogui.click(x = 100, y = 550, duration = mouseTime, button = 'right')
        pyautogui.click(x = 200, y = 585, duration = mouseTime)
        pyautogui.click(x = 200, y = 600, duration = mouseTime)
        time.sleep(0.5)
        pyautogui.typewrite(('E:/'+folder), interval = keyTime)
        pyautogui.click(x = 200, y = 625, duration = mouseTime)
        time.sleep(0.5)
        pyautogui.typewrite('DAC%0.0d_%0.0d' % (num, value_dac), interval = keyTime)
        pyautogui.click(x = 490, y = 765, duration = mouseTime)
        time.sleep(sleepTime)
    #set back to 0
    pyautogui.click(x=loc[0], y=loc[1], duration = mouseTime)
    pyautogui.press(['backspace','backspace','backspace','backspace','backspace'], interval = keyTime)
    pyautogui.typewrite('%0.0d' % 0, interval = keyTime)
    pyautogui.click(x=loc[0]+100, y=loc[1], duration=mouseTime)
    num = num + 1
print(datetime.datetime.now())
print("Hey look mom, I made it (4) ")
```

B.1.2 DataReader.m

```
%DataReader.m
%
%reads all the oscope logs and creates 3 spreadsheets of data
clc
clear all
close all

header = {'DAC','Val','Mean','Min','Max','PkPk','Std'};
std_lim = 10;
folder = 'TxTester_F_2500MHz_100MHz_3';
%read in tests for DAC1
T1 = cell2table(cell(0,7),'VariableNames',header);
files = dir([folder '/DAC1*.csv']);
for f = files'
    i1 = strfind(f.name,'_');
    i2 = strfind(f.name,'.');
    val = str2double(f.name(i1+1:i2-1));
    [num,dat1] = xlsread([folder '/' f.name], 'K6:O6');
    for n = 1:size(dat1,2)
        dat1{n} = str2double(dat1{n}(1:size(dat1{n},2)-4));
    end
    [num,dat2] = xlsread([folder '/' f.name], 'K8:O8');
    for n = 1:size(dat2,2)
        dat2{n} = str2double(dat2{n}(1:size(dat2{n},2)-4));
    end
    [num,dat3] = xlsread([folder '/' f.name], 'K10:O10');
    for n = 1:size(dat3,2)
        dat3{n} = str2double(dat3{n}(1:size(dat3{n},2)-4));
    end
    T1 = [T1; 1 val dat1];
    T1 = [T1; 2 val dat2];
    T1 = [T1; 3 val dat3];
end

%read in tests for DAC2
T2 = cell2table(cell(0,7),'VariableNames',header);
files = dir([folder '/DAC2*.csv']);
for f = files'
    i1 = strfind(f.name,'_');
    i2 = strfind(f.name,'.');
    val = str2double(f.name(i1+1:i2-1));
    [num,dat1] = xlsread([folder '/' f.name], 'K6:O6');
    for n = 1:size(dat1,2)
        dat1{n} = str2double(dat1{n}(1:size(dat1{n},2)-4));
    end
    [num,dat2] = xlsread([folder '/' f.name], 'K8:O8');
    for n = 1:size(dat2,2)
        dat2{n} = str2double(dat2{n}(1:size(dat2{n},2)-4));
    end
    [num,dat3] = xlsread([folder '/' f.name], 'K10:O10');
    for n = 1:size(dat3,2)
        dat3{n} = str2double(dat3{n}(1:size(dat3{n},2)-4));
    end
    T2 = [T2; 1 val dat1];
```

```
    T2 = [T2; 2 val dat2];
    T2 = [T2; 3 val dat3];
end

%read in tests for DAC3
T3 = cell2table(cell(0,7),'VariableNames',header);
files = dir([folder '/DAC3*.csv']);
for f = files'
    i1 = strfind(f.name,'_');
    i2 = strfind(f.name,'.');
    val = str2double(f.name(i1+1:i2-1));
    [num,dat1] = xlsread([folder '/' f.name], 'K6:O6');
    for n = 1:size(dat1,2)
        dat1{n} = str2double(dat1{n}(1:size(dat1{n},2)-4));
    end
    [num,dat2] = xlsread([folder '/' f.name], 'K8:O8');
    for n = 1:size(dat2,2)
        dat2{n} = str2double(dat2{n}(1:size(dat2{n},2)-4));
    end
    [num,dat3] = xlsread([folder '/' f.name], 'K10:O10');
    for n = 1:size(dat3,2)
        dat3{n} = str2double(dat3{n}(1:size(dat3{n},2)-4));
    end
    T3 = [T3; 1 val dat1];
    T3 = [T3; 2 val dat2];
    T3 = [T3; 3 val dat3];
end

%remove any bad table rows
T1(T1.Std >= std_lim, :) = [];
T2(T2.Std >= std_lim, :) = [];
T3(T3.Std >= std_lim, :) = [];
%sort by DAC numbers
T1 = sortrows(sortrows(T1, 2), 1);
T2 = sortrows(sortrows(T2, 2), 1);
T3 = sortrows(sortrows(T3, 2), 1);
%write new files
write(T1, [folder '/DAC1.xls']);
write(T2, [folder '/DAC2.xls']);
write(T3, [folder '/DAC3.xls']);
%make one super table
T{1} = T1;
T{2} = T2;
T{3} = T3;
TT = T;
```

B.1.3 Plotter.m

```
%Plotter.m
%
%Reads in measured phase shifter data, finds a curve fit
%and plots the reults

clear all
folder = 'TxTester_F_2500MHz_100MHz_3';
order = [7,7,7];
% dac_cutoff = [3000, 2800, 3000];
dac_cutoff = [4095, 4095, 4095];
T1 = readtable([folder '/DAC1.xls']);
T2 = readtable([folder '/DAC2.xls']);
T3 = readtable([folder '/DAC3.xls']);
%make one super table
T{1} = T1;
T{2} = T2;
T{3} = T3;

% %remove outliers
%RxTester_2500MHz_10
if strcmp(folder,'RxTester_2500MHz_10')
    T{1}(T{1}.Val == 2050,:) = [];
    T{1}(T{1}.Val == 2610,:) = [];
    T{2}(T{2}.Val == 2910,:) = [];
elseif strcmp(folder,'TX_10')
    T{2}(T{2}.Val == 2950,:) = [];
elseif strcmp(folder, 'RxTester_BB639_2400MHz_100MHz_10')
    T{1}(T{1}.Val == 3990,:) = [];
    T{2}(T{2}.Val == 1870,:) = [];
    T{3}(T{3}.Val == 2680,:) = [];
elseif strcmp(folder, 'TxTester_BB639_2500MHz_100MHz_10')
    T{1}(T{1}.Val == 1460,:) = [];
    T{2}(T{2}.Val == 3310,:) = [];
    T{3}(T{3}.Val == 2100,:) = [];
elseif strcmp(folder, 'RxTester_NOSW_2410MHz_100MHz_10')
    T{1}(T{1}.Val == 3320,:) = [];
    T{1}(T{1}.Val == 3640,:) = [];
    T{3}(T{3}.Val == 2940,:) = [];
elseif strcmp(folder, 'RxTester_Terminated_2410MHz_100MHz_3')
    T{1}(T{1}.Val == 3285,:) = [];
    T{1}(T{1}.Val == 2751,:) = [];
    T{1}(T{1}.Val == 2385,:) = [];
    T{2}(T{2}.Val == 1668,:) = [];
    T{2}(T{2}.Val == 2259,:) = [];
    T{2}(T{2}.Val == 2538,:) = [];
    T{2}(T{2}.Val == 3132,:) = [];
    T{2}(T{2}.Val == 4011,:) = [];
    T{2}(T{2}.Val == 477,:)  = [];
    T{3}(T{3}.Val == 3987,:) = [];
    T{3}(T{3}.Val == 1200,:) = [];
    T{3}(T{3}.Val == 2148,:) = [];
    T{3}(T{3}.Val == 2373,:) = [];
    T{3}(T{3}.Val == 1872,:) = [];
    T{3}(T{3}.Val == 3990,:) = [];
```

```
    T{3}(T{3}.Val == 1179,:) = [];
    T{3}(T{3}.Val == 4071,:) = [];
    T{3}(T{3}.Val == 3186,:) = [];
    T{3}(T{3}.Val == 3447,:) = [];
    T{3}(T{3}.Val == 3132,:) = [];
elseif strcmp(folder, 'TxTester_F_2500MHz_100MHz_3')
    T{1}(T{1}.Val == 1392,:) = [];
    T{3}(T{3}.Val == 2679,:) = [];
end

%unwrap
for j = [1, 2, 3]
    offset = 0;
    i1 = find(T{j}.DAC==j, 1, 'first');
    i2 = find(T{j}.DAC==j, 1, 'last');
    for n = i1:i2-1
        f = 0;
        if (T{j}.Mean(n+1) - T{j}.Mean(n)) > 180
            f = 1;
        end
        T{j}.Mean(n) = T{j}.Mean(n) - offset;
        if f == 1
            offset = offset + 360;
        end
    end
    T{j}.Mean(i2) = T{j}.Mean(i2) - offset;
end
% %find a 4095 point curve for the DAC and write it out
% LineFit = cell(1,3);
% LineFit_data = cell(1,3);
% for j = [1, 2, 3]
%     %preallocate zeros
%     LineFit{j} = zeros(4096,1);
%     %select only data for the relevant DAC
%     a = T{j}(T{j}.DAC == j,:);
%     %interpolate linearly between points
%     dac = 0:1:4095;
%     i = 1;
%     for n = 1:size(dac,2)
%         if dac(n) < a.Val(i)
%             LineFit{j}(n) = a.Mean(i-1) + m * (dac(n) - a.Val(i-1));
%         else
%             if i == size(a,1)
%                 LineFit{j}(n) = a.Mean(i);
%             else
%                 i = i + 1;
%                 m = (a.Mean(i) - a.Mean(i-1)) / (a.Val(i) - a.Val(i-1));
%                 LineFit{j}(n) = a.Mean(i-1);
%             end
%         end
%     end
%     %keep sample to plot against
%     LineFit_data{j} = a;
% end
%find a polynomial and find fit
for j = [1, 2, 3]
    a = T{j}(T{j}.DAC==j,:);
```

```
    a = a(a.Val <= dac_cutoff(j),:);
    p{j} = polyfit(a.Val, a.Mean, order(j));
    x{j} = 0:1:dac_cutoff(j);
    y{j} = polyval(p{j},x{j});
end
%rewrap the polynomial and mean data
for j = [1, 2, 3]
    for n = find(T{j}.DAC==j, 1, 'first'):find(T{j}.DAC==j, 1, 'last')
        while T{j}.Mean(n) < 0
            T{j}.Mean(n) = T{j}.Mean(n) + 360;
        end
        while T{j}.Mean(n) >= 360
            T{j}.Mean(n) = T{j}.Mean(n) - 360;
        end
    end
    for n = 1:size(y{j},2)
        while y{j}(n) < 0
            y{j}(n) = y{j}(n) + 360;
        end
        while y{j}(n) >= 360
            y{j}(n) = y{j}(n) - 360;
        end
    end
end
%add NaNs for polynomial line plotting
for j = [1, 2, 3]
    m = size(y{j},2);
    n = 2;
    while n ~= m
        if (y{j}(n) - y{j}(n-1)) > 180
            y{j}(n+1:m+1) = y{j}(n:m);
            y{j}(n) = NaN;
            x{j}(n+1:m+1) = x{j}(n:m);
            x{j}(n) = NaN;
            m = m + 1;
        end
        n = n + 1;
    end
end

%write coefficients to file
header = cell(1,max(order)+1);
for j = 1:size(header,2)
    header{j} = sprintf('poly%d',max(order)+1 - j);
end
Tp = cell2table(cell(3,size(header,2)), 'VariableNames', header);
for j = [1,2,3]
    for i = 1:size(p{j},2)
        Tp{j,i} = {p{j}(i)};
    end
end
write(Tp, [folder '/coeff.xls']);


%% graph results

f = figure(1);
```

```
f.Units = 'inches';
f.Position = [0, 1, 12, 8];

%sub1
for j = [1, 2, 3]
    s = subplot(2,3,j);
    for i = [1, 2, 3]
        scatter(T{j}.Val(T{j}.DAC == i), T{j}.Mean(T{j}.DAC == i),
        'DisplayName', sprintf('DAC %0.0d', i));
        hold on
    end
    plot(x{j}, y{j}, 'DisplayName', sprintf('Fit %0.0d',i));
    hold off
    l = legend('show');
    l.Location = 'Best';
    grid on
    xlim([0 4095])
    ylim([0 360])
    s.YTick = [0 90 180 270 360];
    ylabel('degrees')
    xlabel('DAC')
    title(sprintf('DAC %0.0d Mean', j));

    s = subplot(2,3,j+3);
    for i = [1, 2, 3]
        stem(T{j}.Val(T{j}.DAC == i), T{j}.Std(T{j}.DAC == i),
        'DisplayName', sprintf('DAC %0.0d', i));
        hold on
    end
    hold off
    l = legend('show');
    l.Location = 'Best';
    grid on
    xlim([0 4095])
    ylim([0 10]);
    s.YTick = [0 2.5 5 7.5 10];
    ylabel('degrees')
    xlabel('DAC')
    title(sprintf('DAC %0.0d Std. Dev', j));
end
%add label to pic
% label = folder;
% label(label=='_') = ' ';
% suplabel(label,'t');

%  3/6/20
% print([folder '/' folder],'-djpeg');
```

B.1.4 LUTs.m

```
% LUT generator.m

clear all
dac_rx_max = 4000;
dac_tx_max = 4000;
% read in polynomials
folder_rx = 'RxTester_Terminated_2410MHz_100MHz_3';
folder_tx = 'TxTester_F_2500MHz_100MHz_3';
folder_dest = '../../../Software/TM4C/';
T = readtable([folder_rx '/coeff.xls']);
for i = 1:size(T,1)
    p_rx{i} = T{i,:};
end
T = readtable([folder_tx '/coeff.xls']);
for i = 1:size(T,1)
    p_tx{i} = T{i,:};
end

%generate values
dac = 0:1:4095;
for i = 1:3
    %generate
    rx{i} = polyval(p_rx{i},dac);
    tx{i} = polyval(p_tx{i},dac);
    %shift so they start at 0
    rx{i} = rx{i} - rx{i}(1);
    tx{i} = tx{i} - tx{i}(1);
end

%% TX LUT (generate txLUTs.c)
% for TX LUT, the input is the angle to write (0-359) and the output is a
% DAC value (0-4095)

%only keep TX values for dac <= 2000
dac_tx = 0:1:dac_tx_max;
wrap_tx = zeros(1,3);
for i = 1:3
    %only keep DAC <= 2000
    tx{i}((size(dac_tx,2)+1):size(tx{i},2)) = [];
    %shift so we end at 0 degrees
    tx{i} = tx{i} - tx{i}(size(tx{i},2));
    %count back from end until we find first wrap point
    for j = size(tx{i},2):-1:1
        if tx{i}(j) >= 360
            wrap_tx(i) = j;
            break;
        end
    end
    %now, wrap them values!
    for j = 1:size(tx{i},2)
        while tx{i}(j) >= 360
            tx{i}(j) = tx{i}(j) - 360;
        end
        while tx{i}(j) < 0
```

```
                tx{i}(j) = tx{i}(j) + 360;
            end
        end
        %actually, just delete everyone who don't wrap
        tx{i}(1:wrap_tx(i)-1) = [];
        %now, find the average DAC value for each discrete angle
        for j = 0:359
            m = [];
            for n = 2:size(tx{i},2)
                if round(tx{i}(n)) == j
                    m = [m, (wrap_tx(i) - 1 + n)];
                end
            end
            LUT_tx{i}(j+1) = round(mean(m));
        end
    end

    % now plot!
    f = figure(2);
    % sub 1 - angle vs. dac
    subplot(3,1,1);
    for i = 1:3
        plot(dac_tx(wrap_tx(i):size(dac_tx,2)), tx{i});
        hold on
    end
    for i = 1:3
        scatter(wrap_tx(i), tx{i}(1), 'g');
    end
    hold off
    ylim([0 360]);
    xlim([0 4095]);
    xlabel('DAC');
    ylabel('Angle');
    title('TX Graphs');
    % sub 2 - dac vs. angle
    subplot(3,1,2);
    for i = 1:3
        scatter(tx{i},dac_tx(wrap_tx(i):size(dac_tx,2)));
        hold on;
    end
    for i = 1:3
        plot([0 360],[wrap_tx(i) wrap_tx(i)], 'g');
    end
    hold off;
    xlim([0 360]);
    ylim([0 4095]);
    xlabel('Angle');
    ylabel('DAC');
    %sub 3 - discrete DAC vs. discrete angle (LUT_tx)
    subplot(3,1,3)
    for i = 1:3
        scatter(0:1:359, LUT_tx{i});
        hold on
    end
    hold off;
    xlim([0 360])
    ylim([0 4095])
```

```
xlabel('Angle');
ylabel('DAC');

%now, we print the LUT.c file
f = fopen([folder_dest '/txLUT.c'],'w');

fprintf(f, '/*\n');
fprintf(f, ' *    txLUT.c\n');
fprintf(f, ' *\n');
fprintf(f, [' *    Generated with data in: ' folder_tx '\n']);
fprintf(f, ' */\n');
fprintf(f, '\n');
fprintf(f, '#include "HARA_LUTs.h"\n');
fprintf(f, '\n\n');
fprintf(f, '//****************************************************************************\n');
fprintf(f, '//! Finds DAC value to write to achieve a phase difference\n');
fprintf(f, '//!\n');
fprintf(f, '//! \\param phase is the phase (0-359) that you would like to write to the\n');
fprintf(f, '//!               TX DAC in question\n');
fprintf(f, '//! \\param board is the hara element being used for the phase shifter (1-3)\n');
fprintf(f, '//!\n');
fprintf(f, '//! This function returns a 12-bit value from a LUT containing DAC values that\n');
fprintf(f, '//! correspond to decimal degrees (0 - 359).\n');
fprintf(f, '//!\n');
fprintf(f, '//! \\return DAC value to write to achieve the intended phase\n');
fprintf(f, '//****************************************************************************\n');
fprintf(f, 'uint16_t txLUT(unsigned int phase, unsigned int board) {\n');
for i = 1:3
    fprintf(f, '    static const uint16_t txLUT_%0.0d[360] = {\n',i);
    fprintf(f, '        ');
    for j = 1:360
        fprintf(f, '%4d, ', LUT_tx{i}(j));
        if ~mod(j,10)
            fprintf(f, '\n        ');
        end
    end
    fprintf(f, '};\n');
end
fprintf(f, '    //wrap phase\n');
fprintf(f, '    if (phase > 359)\n');
fprintf(f, '        phase = 359;\n');
fprintf(f, '    //return the DAC value from the correct LUT\n');
fprintf(f, '    switch (board) {\n');
fprintf(f, '        case 1:\n');
fprintf(f, '            return txLUT_1[phase];\n');
fprintf(f, '        case 2:\n');
fprintf(f, '            return txLUT_2[phase];\n');
fprintf(f, '        case 3:\n');
fprintf(f, '            return txLUT_3[phase];\n');
fprintf(f, '        default:\n');
fprintf(f, '            return 2048;\n');
```

```
fprintf(f, '    }\n');
fprintf(f, '}\n');

fclose(f);

%% RX LUT (generate rxLUTs.c)
% for RX LUT, the input is the angle to write (0-359) and the output is a
% DAC value (0-4095)

%only keep RX values for dac <= 2000
dac_rx = 0:1:dac_rx_max;
wrap_rx = zeros(1,3);
for i = 1:3
    %only keep DAC <= 2000
    rx{i}((size(dac_rx,2)+1):size(rx{i},2)) = [];
    %shift so we end at 0 degrees
    rx{i} = rx{i} - rx{i}(size(rx{i},2));
    %count back from end until we find first wrap point
    for j = size(rx{i},2):-1:1
        if rx{i}(j) >= 360
            wrap_rx(i) = j;
            break;
        end
    end
    %now, wrap them values!
    for j = 1:size(rx{i},2)
        while rx{i}(j) >= 360
            rx{i}(j) = rx{i}(j) - 360;
        end
        while rx{i}(j) < 0
            rx{i}(j) = rx{i}(j) + 360;
        end
    end
    %actually, just delete everyone who don't wrap
    rx{i}(1:wrap_rx(i)-1) = [];
    %now, find the average DAC value for each discrete angle
    for j = 0:359
        m = [];
        for n = 2:size(rx{i},2)
            if round(rx{i}(n)) == j
                m = [m, (wrap_rx(i) - 1 + n)];
            end
        end
        LUT_rx{i}(j+1) = round(mean(m));
    end
end

% now plot!
f = figure(3);
% sub 1 - angle vs. dac
subplot(3,1,1);
for i = 1:3
    plot(dac_rx(wrap_rx(i):size(dac_rx,2)), rx{i});
    hold on
end
for i = 1:3
    scatter(wrap_rx(i), rx{i}(1), 'g');
```

```
end
hold off
ylim([0 360]);
xlim([0 4095]);
xlabel('DAC');
ylabel('Angle');
title('RX Graphs');
% sub 2 - dac vs. angle
subplot(3,1,2);
for i = 1:3
    scatter(rx{i},dac_rx(wrap_rx(i):size(dac_rx,2)));
    hold on;
end
for i = 1:3
    plot([0 360],[wrap_rx(i) wrap_rx(i)], 'g');
end
hold off;
xlim([0 360]);
ylim([0 4095]);
xlabel('Angle');
ylabel('DAC');
%sub 3 - discrete DAC vs. discrete angle (LUT_rx)
subplot(3,1,3)
for i = 1:3
    scatter(0:1:359, LUT_rx{i});
    hold on
end
hold off;
xlim([0 360])
ylim([0 4095])
xlabel('Angle');
ylabel('DAC');

%now, we print the LUT.c file
f = fopen([folder_dest '/rxLUT.c'],'w');

fprintf(f, '/*\n');
fprintf(f, ' *    rxLUT.c\n');
fprintf(f, ' *\n');
fprintf(f, [' *    Generated with data in: ' folder_rx '\n']);
fprintf(f, ' */\n');
fprintf(f, '\n');
fprintf(f, '#include "HARA_LUTs.h"\n');
fprintf(f, '\n\n');
fprintf(f, '//*****************************************************************************\n');
fprintf(f, '//! Finds DAC value to write to achieve a phase difference\n');
fprintf(f, '//!\n');
fprintf(f, '//! \\param phase is the phase (0-359) that you would like to write to the\n');
fprintf(f, '//!              RX DAC in question\n');
fprintf(f, '//! \\param board is the hara element being used for the phase shifter (1-3)\n');
fprintf(f, '//!\n');
fprintf(f, '//! This function returns a 12-bit value from a LUT containing DAC values that\n');
fprintf(f, '//! correspond to decimal degrees (0 - 359).\n');
```

```
fprintf(f, '//!\n');
fprintf(f, '//! \\return DAC value to write to achieve the intended phase\n');
fprintf(f, '//******************************************************************************\n');
fprintf(f, 'uint16_t rxLUT(unsigned int phase, unsigned int board) {\n');
for i = 1:3
    fprintf(f, '    static const uint16_t rxLUT_%0.0d[360] = {\n',i);
    fprintf(f, '        ');
    for j = 1:360
        fprintf(f, '%4d, ', LUT_rx{i}(j));
        if ~mod(j,10)
            fprintf(f, '\n        ');
        end
    end
    fprintf(f, '};\n');
end
fprintf(f, '    //wrap phase\n');
fprintf(f, '    if (phase > 359)\n');
fprintf(f, '        phase = 359;\n');
fprintf(f, '    //return the DAC value from the correct LUT\n');
fprintf(f, '    switch (board) {\n');
fprintf(f, '        case 1:\n');
fprintf(f, '            return rxLUT_1[phase];\n');
fprintf(f, '        case 2:\n');
fprintf(f, '            return rxLUT_2[phase];\n');
fprintf(f, '        case 3:\n');
fprintf(f, '            return rxLUT_3[phase];\n');
fprintf(f, '        default:\n');
fprintf(f, '            return 2048;\n');
fprintf(f, '    }\n');
fprintf(f, '}\n');

fclose(f);

%% CP LUT (generate cpLUTs.c)
% for CP LUT, the input is the ADC value (0 - 4095) and the output is an
% angle value (0 - 359)

%generate values
adc_max = 3.0;
dac_max = 3.3;
dac_cp = linspace(0, adc_max * 4095 / dac_max, 4095);
for i = 1:3
    %generate
    rx_cp{i} = polyval(p_rx{i},dac_cp);
    %shift so they start at 0
    rx_cp{i} = rx_cp{i} - rx_cp{i}(1);
    %now wrap
    for n = 1:size(rx_cp{i},2)
        while rx_cp{i}(n) > 359
            rx_cp{i}(n) = rx_cp{i}(n) - 360;
        end
        while rx_cp{i}(n) < 0
            rx_cp{i}(n) = rx_cp{i}(n) + 360;
        end
    end
```

```
    %now round
    rx_cp{i} = round(rx_cp{i});
end

% now plot!
f = figure(4);
% sub 1 - angle vs. voltage
subplot(2,1,1);
for i = 1:3
    plot(dac_cp * dac_max / 4095, rx_cp{i});
    hold on
end
hold off
ylim([0 360]);
xlabel('ADC (V)');
ylabel('Angle');
title('CP Graphs');
% sub 2 - ADC vs. angle
subplot(2,1,2);
for i = 1:3
    scatter(rx_cp{i},dac_cp);
    hold on;
end
hold off;
xlim([0 360]);
ylim([0 4095]);
xlabel('Angle');
ylabel('ADC');

%now, we print the LUT.c file
f = fopen([folder_dest '/cpLUT.c'],'w');

fprintf(f, '/*\n');
fprintf(f, ' *    cpLUT.c\n');
fprintf(f, ' *\n');
fprintf(f, [' *    Generated with data in: ' folder_rx '\n']);
fprintf(f, ' */\n');
fprintf(f, '\n');
fprintf(f, '#include "HARA_LUTs.h"\n');
fprintf(f, '\n\n');
fprintf(f, '//*****************************************************************************\n');
fprintf(f, '//! Finds angle corresponding to measured ADC voltage\n');
fprintf(f, '//!\n');
fprintf(f, '//! \\param adc is the adc value (0-4095) that you want to find the corresponding\n');
fprintf(f, '//!              angle for\n');
fprintf(f, '//! \\param board is the hara element being used (1-3)\n');
fprintf(f, '//!\n');
fprintf(f, '//! This function returns a 12-bit value from a LUT containing angle values that\n');
fprintf(f, '//! correspond to decimal degrees (0 - 359).\n');
fprintf(f, '//!\n');
fprintf(f, '//! \\return angle value corresponding to input adc value\n');
fprintf(f, '//*****************************************************************************\n');
fprintf(f, 'uint16_t cpLUT(unsigned int adc, unsigned int board) {\n');
```

```
for i = 1:3
    fprintf(f, '    static const uint16_t cpLUT_%0.0d[4096] = {\n',i);
    fprintf(f, '        ');
    for j = 1:size(rx_cp{i},2)
        fprintf(f, '%3d, ', rx_cp{i}(j));
        if ~mod(j,32)
            fprintf(f, '\n        ');
        end
    end
    fprintf(f, '};\n');
end
fprintf(f, '    //wrap adc value\n');
fprintf(f, '    if (adc > 4095)\n');
fprintf(f, '        adc = 4095;\n');
fprintf(f, '    //return the angle value from the correct LUT\n');
fprintf(f, '    switch (board) {\n');
fprintf(f, '        case 1:\n');
fprintf(f, '            return cpLUT_1[adc];\n');
fprintf(f, '        case 2:\n');
fprintf(f, '            return cpLUT_2[adc];\n');
fprintf(f, '        case 3:\n');
fprintf(f, '            return cpLUT_3[adc];\n');
fprintf(f, '        default:\n');
fprintf(f, '            return 180;\n');
fprintf(f, '    }\n');
fprintf(f, '}\n');

fclose(f);
```

B.2 Sample Test Results

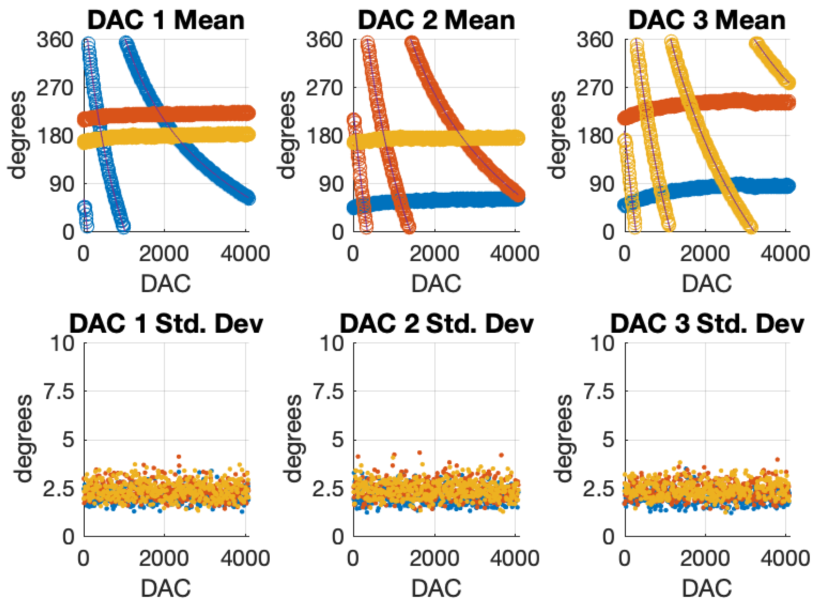


Figure B.1: Downconversion signal chain phase shifter characterization for a 2.410 GHz signal set 1

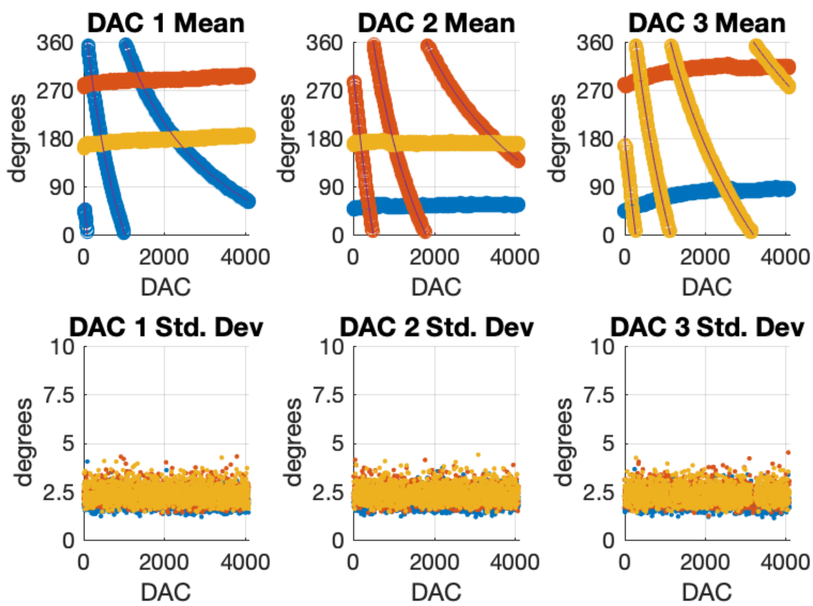


Figure B.2: Downconversion signal chain phase shifter characterization for a 2.410 GHz signal set 2

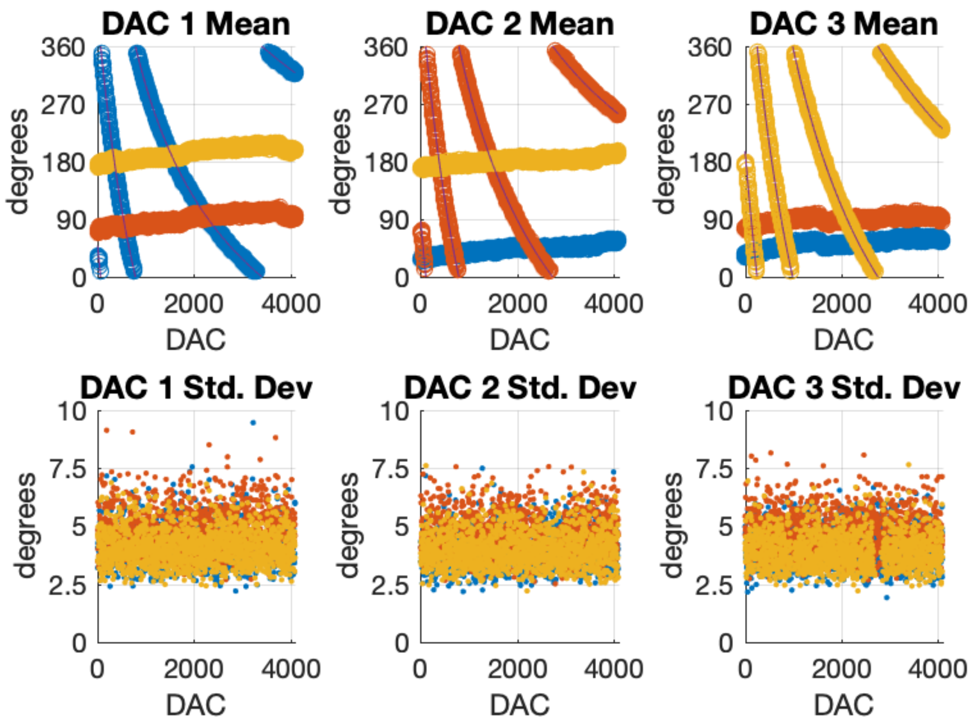


Figure B.3: Carrier generation phase shifter characterization for a 2.500 GHz signal set 1

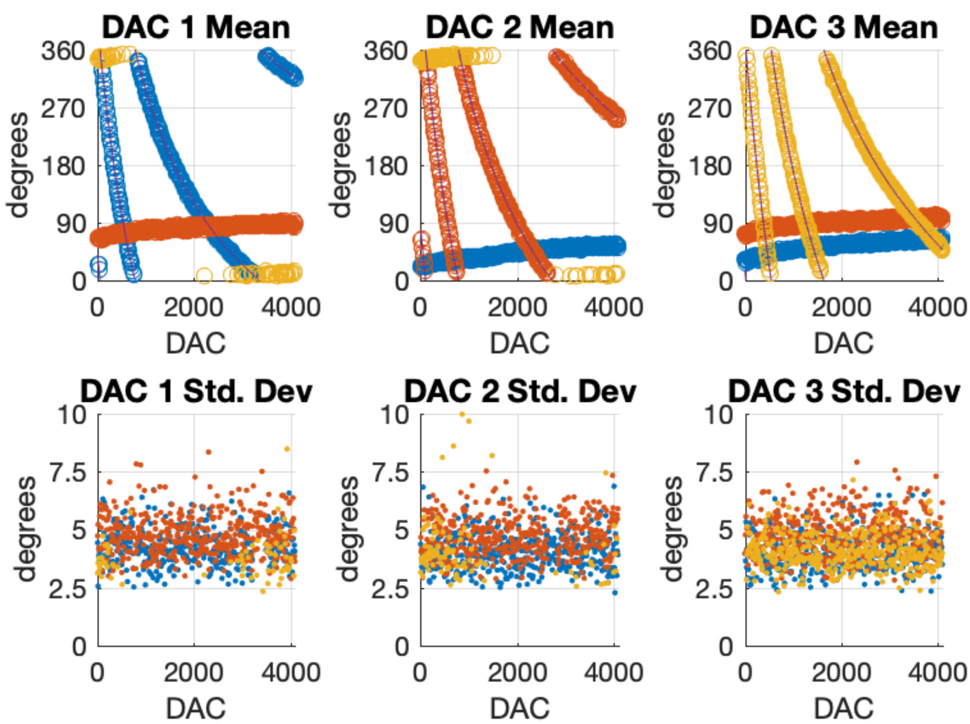


Figure B.4: Carrier generation phase shifter characterization for a 2.500 GHz signal set 2

Appendix C

Anechoic Chamber Test Files

C.1 Software

C.1.1 AnechoicChamberTest.py

```
import pyautogui
import time
import numpy

TestName = 'Y_'

#class for Oscope
#   Ensure Chrome is the FIRST ICON IN THE TRAY
class KeysightOscope :
    Icon = (415, 2060)
    CommandBox = (565, 298)
    SendCommand = (570,340)
    SaveCommand = "SAVE:WAVEFORM ALL, \"" #SAVE:WAVEFORM ALL, "<file.csv>"
    SaveDirectory = "E:/AnechoicTests/"
    SaveFile = 'ScopeIF_'

    def record(self, angle):
        pyautogui.click(self.Icon, duration=0.5)
        pyautogui.click(self.CommandBox, duration=1.0, clicks=3)
        pyautogui.typewrite((self.SaveCommand + self.SaveDirectory +
        self.SaveFile + TestName + angle + '.csv\"'), interval=0.1)
        pyautogui.click(self.SendCommand, duration=0.5)
        time.sleep(1.0)
        pyautogui.click(self.Icon, duration=0.5)

#class for Tektronix SignalVu buttons
#   Ensure SingalVu is the SECOND ICON IN THE TRY
class TektronixSignalVu :
    Icon = (465, 1060)
    File = (75, 35)
    SaveAs = (75, 115)
    Location = (660, 73)
    FileName = (520, 460)
    clearTraces = (1895, 97)
    SaveDirectory = "C:/Users/storm/Desktop/AnechoicTesting"
```

```
    SaveFile = 'SignalVu_'
    mouseTime = 0.5

    #record('angle')
    # saves the current SignalVu data as a .csv
    def record(self, angle):
        pyautogui.click(self.Icon, duration=self.mouseTime)
        pyautogui.click(self.clearTraces, duration=self.mouseTime)
        time.sleep(3.0)
        #pyautogui.click(self.File, duration=self.mouseTime)
        #pyautogui.click(self.SaveAs, duration=self.mouseTime)
        pyautogui.keyDown('ctrl')
        pyautogui.press('s')
        pyautogui.keyUp('ctrl')
        time.sleep(0)
        pyautogui.click(self.Location, duration=self.mouseTime)
        pyautogui.typewrite(self.SaveDirectory, interval=0.1)
        pyautogui.press('enter')
        pyautogui.click(self.FileName, duration=self.mouseTime)
        pyautogui.typewrite((self.SaveFile + TestName + angle + '.csv'),
        interval=0.1)
        pyautogui.press('enter')
        time.sleep(1.0)
        pyautogui.click(self.Icon, duration=self.mouseTime)

#class for HARA GUI
#   Ensure HARA GUI is the THIRD ICON IN THE TRAY
class HARAGui :
    Icon = (515,1060)
    Filename = (1240, 450)
    Log = (1315, 450)
    FileName = 'HARAlog_'

    #move the GUI to the right location
    def position(self) :
        size = pyautogui.size()
        pyautogui.click(self.Icon, duration=0.5)
        pyautogui.keyDown('alt')
        pyautogui.press('space')
        pyautogui.keyUp('alt')
        pyautogui.press('x')
        pyautogui.moveTo(size[0]/2, 5, duration=0.5)
        pyautogui.dragTo(1400, 50, duration=0.5, button='left')
        time.sleep(1.0)
        pyautogui.click(self.Icon, duration=0.5)

    #record sample
    def record(self, angle):
        pyautogui.click(self.Icon, duration=0.5)
        pyautogui.click(self.Filename, duration=0.5, clicks=3)
        pyautogui.typewrite((self.FileName + TestName + angle + '.csv'),
        interval=0.1)
        pyautogui.click(self.Log, duration=0.5)
        time.sleep(5.0)
        pyautogui.click(self.Log, duration=0.5)
        time.sleep(1.0)

```

```
#class for DAMS system
#   Emsure DAMS is the FOURTH ICON IN THE TRAY
class DAMSGui :
    Icon = (565, 1060)
    MoveToPosition = (110, 730)
    MoveToAzimuth = (275, 514)
    MoveTo = (370, 525)

    def moveTo(self, azimuth) :
        pyautogui.click(self.Icon, duration=0.5)
        pyautogui.moveTo(self.MoveToPosition, duration=0.5)
        pyautogui.mouseDown(); time.sleep(0.1); pyautogui.mouseUp()
        pyautogui.moveTo(self.MoveToAzimuth, duration=0.5)
        pyautogui.mouseDown(); time.sleep(0.1); pyautogui.mouseUp()
        pyautogui.mouseDown(); time.sleep(0.1); pyautogui.mouseUp()
        pyautogui.typewrite(str(azimuth), interval=0.1)
        pyautogui.moveTo(self.MoveTo, duration=0.5)
        pyautogui.mouseDown(); time.sleep(0.1); pyautogui.mouseUp()
        time.sleep(1.0)
        pyautogui.click(self.Icon, duration=0.5)





#front matter
print("ICON 1: Chrome")
print("ICON 2: SignalVu")
print("ICON 3: GUI")
print("ICON 4: DAMS")
flag = input("\nSPACE begins, anything else quits.\n")
if flag[0] == ' ':
    #create objects
    Scope = KeysightOscope()
    SignalVu = TektronixSignalVu()
    Hara = HARAGui()
    Hara.position()
    Dams = DAMSGui()
    #main loop
    for angle in numpy.linspace(0, 90, 16):
        Dams.moveTo(angle)
        #wait for user input to confirm
        flag = input("\nSpacebar saves sample, anything else quits\nCurrent angle: %d\n" % angle)
        if flag[0] == ' ':
            #1. log Scope data
            Scope.record(str(int(angle)))
            #2. log SignalVu data
            SignalVu.record(str(int(angle)))
            #3. log GUI data
            Hara.record(str(int(angle)))
        else:
            break
```

C.1.2 DataReader.m

```
clc
clear all

%Table Variables:
%   ViewAngle
%   SignalVu Rx Strength (dBm)
%   Scope IF Sum strength (dBv)
%   Scope IF Single strength (dBv)
%   Log Theta1 (degrees)
%   Log Xi1 (degrees)
%   Log Theta2 (degrees)
%   Log Xi2 (degrees)
%   Log Theta3 (degrees)
%   Log Xi3 (degrees)
varNames = {'ViewAngle','TxSignalStrength',...
    'IFSum','IFSingle',...
    'Theta1','Xi1','Theta2','Xi2','Theta3','Xi3'};
TestNum = 'V':'Y';
AngleNum = linspace(0,90,16);

%iterate over tests
for t = 1:size(TestNum,2)
    T = cell2table(cell(0,10), 'VariableNames', varNames);
    %iterate over angles
    for a = 1:size(AngleNum,2)
        %   read in HARAlog
        %   keep Psi and theta
        try
            f =
            sprintf('RawData/HARAlog_%c_%0d.csv',TestNum(t),AngleNum(a));
            haraLog = readtable(f);
            theta1 = mean(haraLog.theta1);
            theta2 = mean(haraLog.theta2);
            theta3 = mean(haraLog.theta3);
            xi1 = mean(haraLog.xi1);
            xi2 = mean(haraLog.xi2);
            xi3 = mean(haraLog.xi3);
        catch
            fprintf('%s not found\n', f);
            theta1 = NaN;
            theta2 = NaN;
            theta3 = NaN;
            xi1 = NaN;
            xi2 = NaN;
            xi3 = NaN;
        end
        %   read in SignalVu 2.5 GHz magnitude measured
        try
            f =
            sprintf('RawData/SignalVu_%c_%0d.csv',TestNum(t),AngleNum(a));
            signalVu = xlsread(f, 1, 'B64');
        catch
            fprintf('%s not found\n', f);
            signalVu = NaN;
```

```
        end
        %   read in ScopeIF_ 10 MHz IF signal strengths
        try
            f = sprintf('RawData/ScopeIF_%c_%0d_ALL.csv',TestNum(t),AngleNum(a));
            scopeIF = [xlsread(f,1,'K16') xlsread(f,1,'N16')];
        catch
            fprintf('%s not found\n', f);
            scopeIF = [NaN NaN];
        end
        T1 = array2table([AngleNum(a) signalVu ...
            scopeIF(1) scopeIF(2) ...
            theta1 xi1 theta2 xi2 theta3 xi3], ...
            'VariableNames', varNames);
        T = [T; T1];
    end
    %write table to file
    f = sprintf('Test_%c.xls',TestNum(t));
    writetable(T,f);
end
```

C.1.3 Plotter.m

```
clc
clear all
% Plotter.m
% Reads in data from generated Test_X.xls files and plots several diferent
% visualizations

% choose test set
testSet = 6;
switch testSet
    case 1
        Test = ['A', 'B', 'C', 'D', 'E'];
        antennas = [-3/8 0; -1/8 0; 1/8 0; 3/8 0];
        d = 1/4;     %element spacing
    case 2
        Test = ['F', 'G', 'H', 'I', 'J'];
        antennas = [-3/6 0; -1/6 0; 1/6 0; 3/6 0];
        d = 1/3;     %element spacing
    case 3
        Test = 'K':'O';
        antennas = [-3/4 0; -1/4 0; 1/4 0; 3/4 0];
        d = 1/2;     %element spacing
    case 4
        Test = 'P':'R';
        antennas = [-0.6036 -0.3536; -0.25 0; 0.25 0; 0.6036 -0.3536];
        d = 42;
    case 5
        Test = 'S':'U';
        antennas = [-0.6036 -0.3536; -0.25 0; 0.25 0; 0.6036 -0.3536];
        d = 42;
    case 6
        Test = 'P':'U';
        antennas = [-0.6036 -0.3536; -0.25 0; 0.25 0; 0.6036 -0.3536];
        d = 42;
    case 7
        Test = 'V':'Y';
        antennas = [-0.6036 -0.3536; -0.25 0; 0.25 0; 0.6036 -0.3536];
        d = 42;
end
fprintf('Test set %d, spacing = %0.2f\n',testSet,d);

% calculate ideal angles of arrival for linear array
viewAngle = linspace(90,180,16); %table turned clockwise
if testSet < 4
    phi = zeros(5,size(viewAngle,2));
    for n = 1:4
        phi(n,:) = (n-1)*d*cosd(viewAngle) * 360;
    end
    phi(5,:) = d*cosd(viewAngle) * 360;
    phi(phi>=180) = phi(phi>=180) - 360;
    phi(phi>=180) = phi(phi>=180) - 360;
    phi(phi<-180) = phi(phi<-180)+360;
    phi(phi<-180) = phi(phi<-180)+360;
else
    phi = zeros(5, size(viewAngle,2));
```

```
end



figure(2)
clf
%
subplot(3,1,1)
plot(viewAngle,phi(2,:),'DisplayName','Truth');
ylim([-180 180]); set(gca,'YTick',[-180 -90 0 90 180]);
xlim([90 180]); set(gca,'XTick',linspace(90,180,5));
legend('location','eastoutside'); grid on;
ylabel('Phase'); set(gca,'fontsize',12);
%
subplot(3,1,2)
plot(viewAngle,phi(3,:),'DisplayName','Truth');
ylim([-180 180]); set(gca,'YTick',[-180 -90 0 90 180]);
xlim([90 180]); set(gca,'XTick',linspace(90,180,5));
legend('location','eastoutside'); grid on;
ylabel('Phase'); set(gca,'fontsize',12);
%
subplot(3,1,3)
plot(viewAngle,phi(4,:),'DisplayName','Truth');
ylim([-180 180]); set(gca,'YTick',[-180 -90 0 90 180]);
xlim([90 180]); set(gca,'XTick',linspace(90,180,5));
legend('location','eastoutside'); grid on;
ylabel('Phase'); xlabel('View Angle'); set(gca,'fontsize',12);
%
figure(1); clf;
%
figure(3); clf;


% read in data and plot
errorAbsolute = cell2table(cell(0,5),'VariableNames',{'ViewAngle','TestName','Theta1','Theta2','Theta3'});
errorRelative = cell2table(cell(0,5),'VariableNames',{'ViewAngle','TestName','Theta1','Theta2','Theta3'});
ite = 1;
TxSignalStrength_cpy = zeros(size(Test,2),16);
Theta = zeros(size(Test,2),3,16);
for t = 1:size(Test,2)

    f = sprintf('Test_%c.xls',Test(t));
    T = readtable(f);
    % ViewAngle really goes from 90 to 180
    T.ViewAngle = T.ViewAngle + 90;
    % offset TxSignalStrength the max is at 0dB gain
    T.TxSignalStrength = T.TxSignalStrength - max(T.TxSignalStrength);
    TxSignalStrength_cpy(ite,:) = T.TxSignalStrength;
    % calculate IFgain for plotting, set so max is 6dB gain
    IFgain = T.IFSum - T.IFSingle;
    IFgain = IFgain - max(IFgain) + 6;
    % give everyone a 0 point to start at
    T.Theta1(1) = 0;
    T.Theta2(1) = 0;
    T.Theta3(1) = 0;
```

```
% wrap data
T.Theta1(T.Theta1 > 180) = T.Theta1(T.Theta1 > 180) - 360;
T.Theta2(T.Theta2 > 180) = T.Theta2(T.Theta2 > 180) - 360;
T.Theta3(T.Theta3 > 180) = T.Theta3(T.Theta3 > 180) - 360;
% store theta values
Theta(ite,1,:) = T.Theta1;
Theta(ite,2,:) = T.Theta2;
Theta(ite,3,:) = T.Theta3;
ite = ite + 1;

% calculate the estimated DoA based on the reported phases
max_gain=ones(size(T.Theta1))*-1000;
max_angle=zeros(size(T.Theta1));
for i = 1:size(T.Theta1,1)
    for fov = 90:1:180
        k = ones(4,1) * 2*pi*[cosd(fov) sind(fov)];
        w(1) = exp(1i * -1 * 0 * pi/180);
        w(2) = exp(1i * -1 * T.Theta1(i) * pi/180);
        w(3) = exp(1i * -1 * T.Theta2(i) * pi/180);
        w(4) = exp(1i * -1 * T.Theta3(i) * pi/180);
        Y = abs(sum( w' .* exp(-1i * dot(k,antennas,2)) ));
        if Y > max_gain(i)
            max_gain(i) = Y;
            max_angle(i) = fov;
        end
    end
end
% plot values
figure(3)
plot(T.ViewAngle, max_angle);
hold on



%Tx/Rx Patterns
figure(1)
%TX pattern
subplot(2,1,1)
polarplot(T.ViewAngle*pi/180, T.TxSignalStrength,
'DisplayName',sprintf('%c',Test(t)));
hold on
%RX pattern
subplot(2,1,2)
polarplot(T.ViewAngle*pi/180,
IFgain,'DisplayName',sprintf('%c',Test(t)));
hold on

%Measured Phases vs. Truth
figure(2)
%element 1
subplot(3,1,1)
hold on
scatter(T.ViewAngle, T.Theta1, 'DisplayName', sprintf('%c',Test(t)));
%element 2
subplot(3,1,2)
hold on
scatter(T.ViewAngle, T.Theta2, 'DisplayName', sprintf('%c',Test(t)));
```

```
    % element 3
    subplot(3,1,3)
    hold on
    scatter(T.ViewAngle, T.Theta3, 'DisplayName', sprintf('%c',Test(t)));

    TT = array2table(...
                [T.ViewAngle, Test(t)*ones(size(T.ViewAngle)) , T.Theta1 - phi(2,:)',  ...
                T.Theta2 - phi(3,:)',  ...
                T.Theta3 - phi(4,:)'],...
              'VariableNames',{'ViewAngle','TestName','Theta1','Theta2','Theta3'});
    errorAbsolute = [errorAbsolute; TT];

    TT = array2table(...
        [T.ViewAngle,  Test(t)*ones(size(T.ViewAngle)) , T.Theta1 - phi(2,:)', ...
        (T.Theta2 - T.Theta1) - (phi(3,:)' - phi(2,:)'), ...
        (T.Theta3 - T.Theta2) - (phi(4,:)' - phi(3,:)')], ...
          'VariableNames',{'ViewAngle','TestName', 'Theta1','Theta2','Theta3'});
    errorRelative = [errorRelative; TT];
end

% Calculate array gain at each view angle using the average Theta values
for i = 1:length(T.ViewAngle)
    fov = T.ViewAngle(i);
    k = ones(4,1) * 2*pi*[cosd(fov) sind(fov)];
    w(1) = exp(1i * -1 * 0 * pi/180);
    w(2) = exp(1i * -1 * nanmean(Theta(:,1,i)) * pi/180);
    w(3) = exp(1i * -1 * nanmean(Theta(:,2,i)) * pi/180);
    w(4) = exp(1i * -1 * nanmean(Theta(:,3,i)) * pi/180);
    Y(i) = abs(sum( w' .* exp(-1i * dot(k,antennas,2)) ));
end
figure(10)
clf
polarplot((T.ViewAngle)*pi/180,10*log10(Y));
title('Tx Signal Strength Calculated from Theta Values');
rlim([0 6.5])

figure(2)
subplot(3,1,1); legend('location','eastoutside'); title('Reported Angles');
subplot(3,1,2); legend('location','eastoutside');
subplot(3,1,3); legend('location','eastoutside');

figure(3)
xlim([90 180]); set(gca,'XTick',linspace(90,180,5));
ylim([90 180]); set(gca,'YTick',linspace(90,180,5));
ylabel('Calculated DoA'); xlabel('Actual DoA'); set(gca,'fontsize',12);

% fix up figure 1
figure(1)
subplot(2,1,1);
title('Transmitted Signal'); set(gca,'fontsize',12);
legend('location','eastoutside')
thetalim([0 180]); grid on;
rlim([-9 0]); set(gca,'RTick',[-9 -6 -3 0]);
```

```matlab
subplot(2,1,2)
title('Received Signal Gain'); set(gca,'fontsize',12);
legend('location','eastoutside')
thetalim([0 180]); grid on
rlim([0 6]); set(gca,'RTick',[0 3 6]);

% Plot average TxSignalStrength stuff
figure(6)
clf;
polarplot(T.ViewAngle*pi/180, nanmean(TxSignalStrength_cpy,1));
rlim([-9 0]); set(gca,'RTick',[-9 -6 -3 0]);
thetalim([0 180]); grid on;
title('Average Transmitted Signal Strength');



%% wrap & plot absolute errors
errorAbsolute.Theta1(errorAbsolute.Theta1 >= 180) = errorAbsolute.Theta1(errorAbsolute.Theta1 >= 180) - 360;
errorAbsolute.Theta2(errorAbsolute.Theta2 >= 180) = errorAbsolute.Theta2(errorAbsolute.Theta2 >= 180) - 360;
errorAbsolute.Theta3(errorAbsolute.Theta3 >= 180) = errorAbsolute.Theta3(errorAbsolute.Theta3 >= 180) - 360;
errorAbsolute.Theta1(errorAbsolute.Theta1 < -180) = errorAbsolute.Theta1(errorAbsolute.Theta1 < -180) + 360;
errorAbsolute.Theta2(errorAbsolute.Theta2 < -180) = errorAbsolute.Theta2(errorAbsolute.Theta2 < -180) + 360;
errorAbsolute.Theta3(errorAbsolute.Theta3 < -180) = errorAbsolute.Theta3(errorAbsolute.Theta3 < -180) + 360;

figure(4)
clf
for t = 1:size(Test,2)
    subplot(3,1,1)
    hold on
    scatter(errorAbsolute.ViewAngle(errorAbsolute.TestName==Test(t)), errorAbsolute.Theta1(errorAbsolute.TestName==Test(t)), ...
        'DisplayName', sprintf('%c',Test(t)))
    subplot(3,1,2)
    hold on
    scatter(errorAbsolute.ViewAngle(errorAbsolute.TestName==Test(t)),errorAbsolute.Theta2(errorAbsolute.TestName==Test(t)), ...
        'DisplayName', sprintf('%c',Test(t)))
    subplot(3,1,3)
    hold on
    scatter(errorAbsolute.ViewAngle(errorAbsolute.TestName==Test(t)),errorAbsolute.Theta3(errorAbsolute.TestName==Test(t)), ...
        'DisplayName', sprintf('%c',Test(t)))
end
subplot(3,1,1);
title('Absolute Errors');
ylim([-180 180]); set(gca,'YTick',[-180 -90 0 90 180]);
xlim([90 180]); set(gca,'XTick',linspace(90,180,5));
legend('location','eastoutside'); grid on;
ylabel('Phase'); set(gca,'fontsize',12);
```

```
subplot(3,1,2);
ylim([-180 180]); set(gca,'YTick',[-180 -90 0 90 180]);
xlim([90 180]); set(gca,'XTick',linspace(90,180,5));
legend('location','eastoutside'); grid on;
ylabel('Phase'); set(gca,'fontsize',12);
subplot(3,1,3);
ylim([-180 180]); set(gca,'YTick',[-180 -90 0 90 180]);
xlim([90 180]); set(gca,'XTick',linspace(90,180,5));
legend('location','eastoutside'); grid on;
ylabel('Phase'); set(gca,'fontsize',12);

fprintf('Absolute Errors:\n');
fprintf('   Theta1 mean: %+6.2f,   mean||: %+6.2f\n',nanmean(errorAbsolute.Theta1), nanmean(abs(errorAbsolute.Theta1)));
fprintf('   Theta2 mean: %+6.2f,   mean||: %+6.2f\n',nanmean(errorAbsolute.Theta2), nanmean(abs(errorAbsolute.Theta2)));
fprintf('   Theta3 mean: %+6.2f,   mean||: %+6.2f\n',nanmean(errorAbsolute.Theta3), nanmean(abs(errorAbsolute.Theta3)));

%% Relative errors
errorRelative.Theta1(errorRelative.Theta1 >= 180) = errorRelative.Theta1(errorRelative.Theta1 >= 180) - 360;
errorRelative.Theta2(errorRelative.Theta2 >= 180) = errorRelative.Theta2(errorRelative.Theta2 >= 180) - 360;
errorRelative.Theta3(errorRelative.Theta3 >= 180) = errorRelative.Theta3(errorRelative.Theta3 >= 180) - 360;
errorRelative.Theta1(errorRelative.Theta1 < -180) = errorRelative.Theta1(errorRelative.Theta1 < -180) + 360;
errorRelative.Theta2(errorRelative.Theta2 < -180) = errorRelative.Theta2(errorRelative.Theta2 < -180) + 360;
errorRelative.Theta3(errorRelative.Theta3 < -180) = errorRelative.Theta3(errorRelative.Theta3 < -180) + 360;

figure(5)
clf
for t = 1:size(Test,2)
    subplot(3,1,1)
    hold on
    scatter(errorRelative.ViewAngle(errorRelative.TestName==Test(t)), errorRelative.Theta1(errorRelative.TestName==Test(t)), ...
        'DisplayName', sprintf('%c',Test(t)))
    subplot(3,1,2)
    hold on
    scatter(errorRelative.ViewAngle(errorRelative.TestName==Test(t)),errorRelative.Theta2(errorRelative.TestName==Test(t)), ...
        'DisplayName', sprintf('%c',Test(t)))
    subplot(3,1,3)
    hold on
    scatter(errorRelative.ViewAngle(errorRelative.TestName==Test(t)),errorRelative.Theta3(errorRelative.TestName==Test(t)), ...
        'DisplayName', sprintf('%c',Test(t)))
end
```

```
subplot(3,1,1);
title('Relative Errors');
ylim([-180 180]); set(gca,'YTick',[-180 -90 0 90 180]);
xlim([90 180]); set(gca,'XTick',linspace(90,180,5));
legend('location','eastoutside'); grid on;
ylabel('Phase'); set(gca,'fontsize',12);
subplot(3,1,2);
ylim([-180 180]); set(gca,'YTick',[-180 -90 0 90 180]);
xlim([90 180]); set(gca,'XTick',linspace(90,180,5));
legend('location','eastoutside'); grid on;
ylabel('Phase'); set(gca,'fontsize',12);
subplot(3,1,3);
ylim([-180 180]); set(gca,'YTick',[-180 -90 0 90 180]);
xlim([90 180]); set(gca,'XTick',linspace(90,180,5));
legend('location','eastoutside'); grid on;
ylabel('Phase'); set(gca,'fontsize',12);

fprintf('Relative Errors:\n');
fprintf('   Theta1 mean: %+6.2f,   mean||: %+6.2f\n',nanmean(errorRelative.Theta1), nanmean(abs(errorRelative.Theta1)));
fprintf('   Theta2 mean: %+6.2f,   mean||: %+6.2f\n',nanmean(errorRelative.Theta2), nanmean(abs(errorRelative.Theta2)));
fprintf('   Theta3 mean: %+6.2f,   mean||: %+6.2f\n',nanmean(errorRelative.Theta3), nanmean(abs(errorRelative.Theta3)));
```

C.2 3D Printed Array Holders

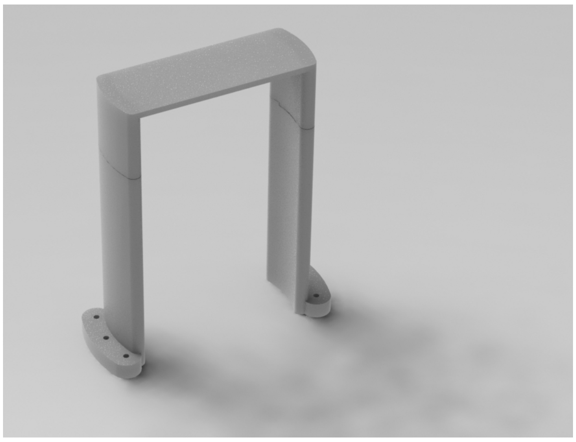


Figure C.1: SolidWorks render of 3D printed stand to attach to DAMS-7000

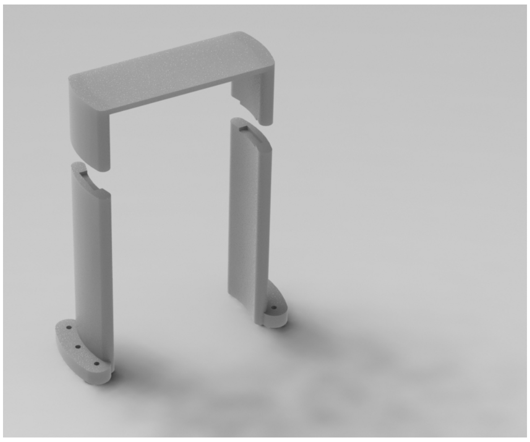


Figure C.2: SolidWorks render of 3D printed stand to attach to DAMS-7000 (exploded view)

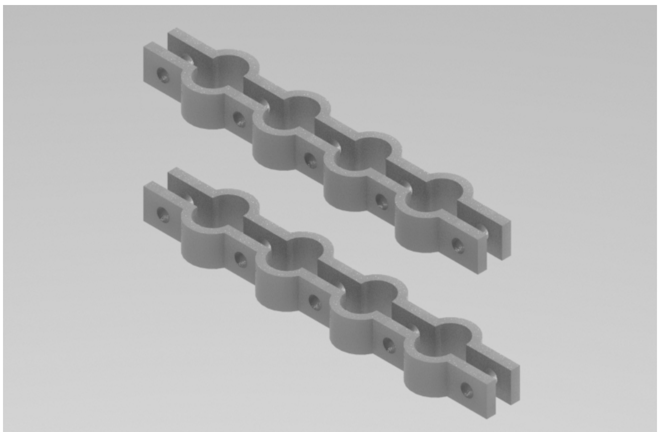


Figure C.3: SolidWorks render of 3D printed $\lambda/4$ linear antenna array holders for 2.400 GHz and 2.500 GHz arrays

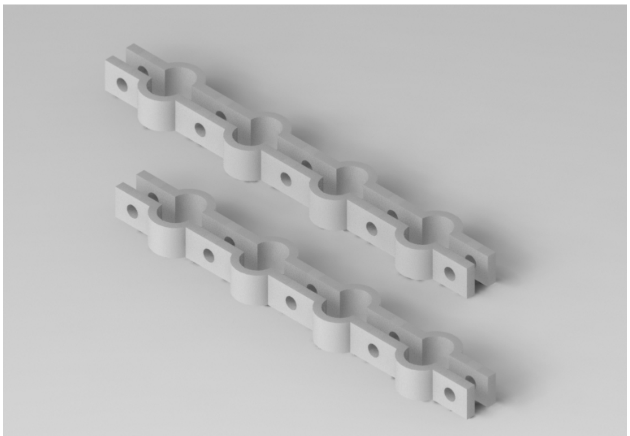


Figure C.4: SolidWorks render of 3D printed $\lambda/3$ linear antenna array holders for 2.400 GHz and 2.500 GHz arrays

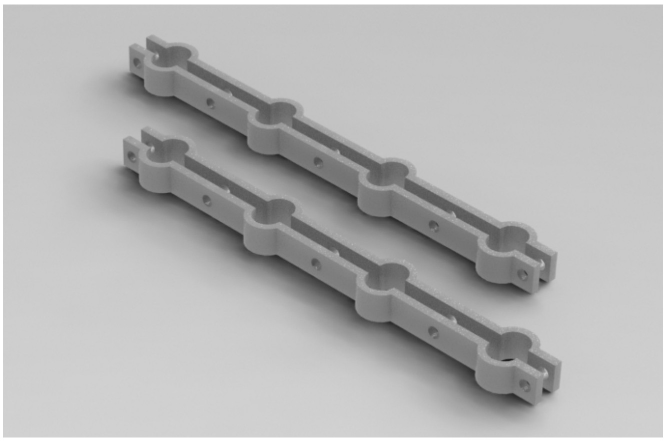


Figure C.5: SolidWorks render of 3D printed $\lambda/2$ linear antenna array holders for 2.400 GHz and 2.500 GHz arrays

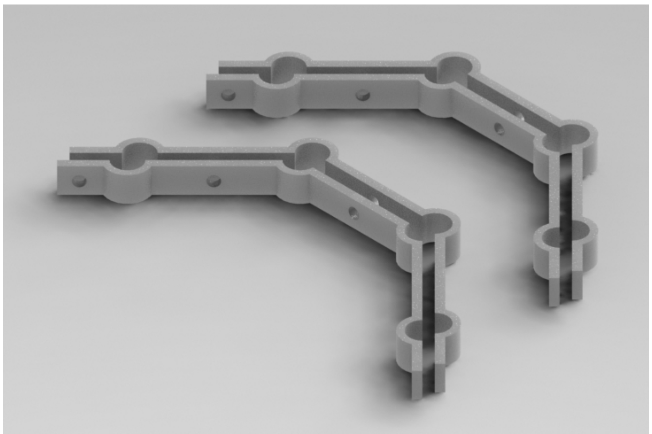
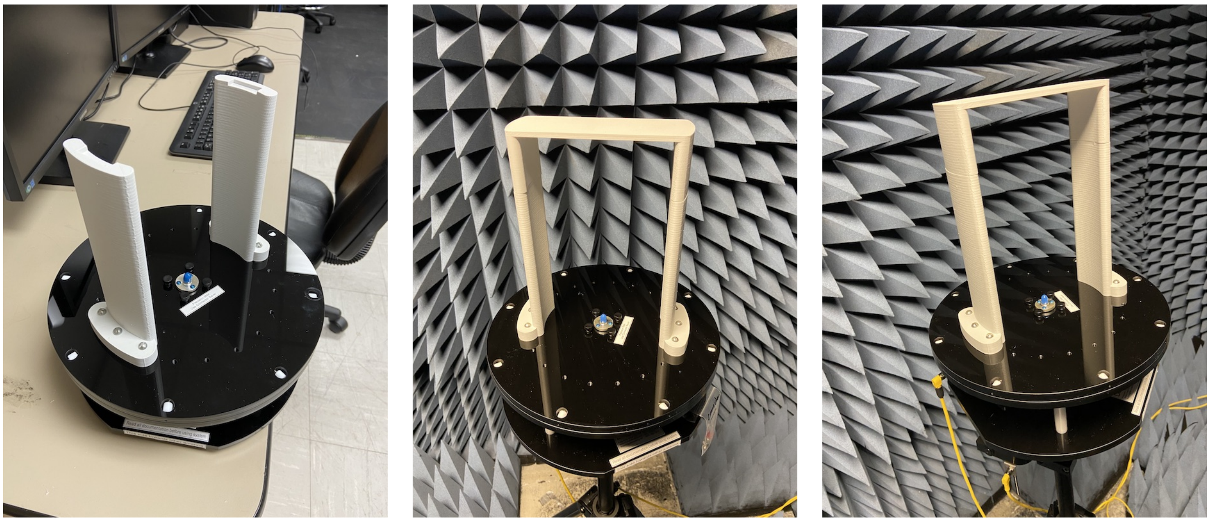


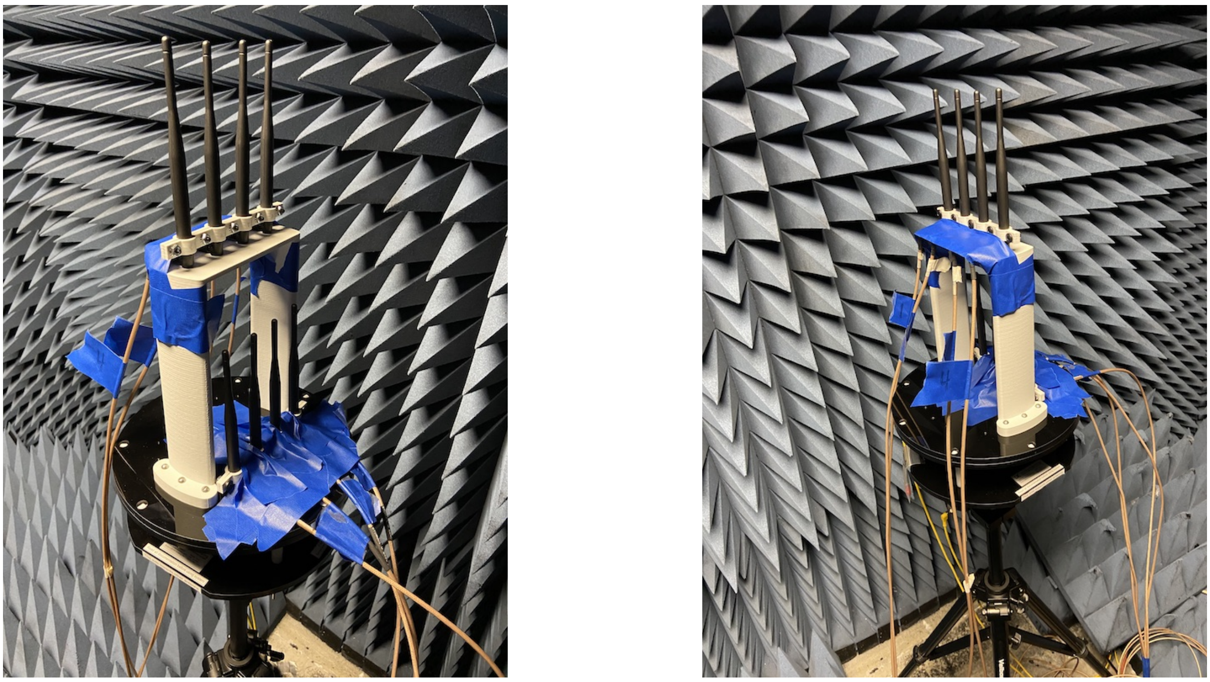
Figure C.6: SolidWorks render of 3D printed irregularly shaped antenna array holders for 2.400 GHz and 2.500 GHz arrays

C.3 Photographs



(a) (b) (c)

Figure C.7: Photographs of the 3D printed DAMS-7000 shelf assembly



(a) (b)

Figure C.8: Photographs of the transmit and receive antenna arrays for the HRA prototype mounted to the DAMS-7000

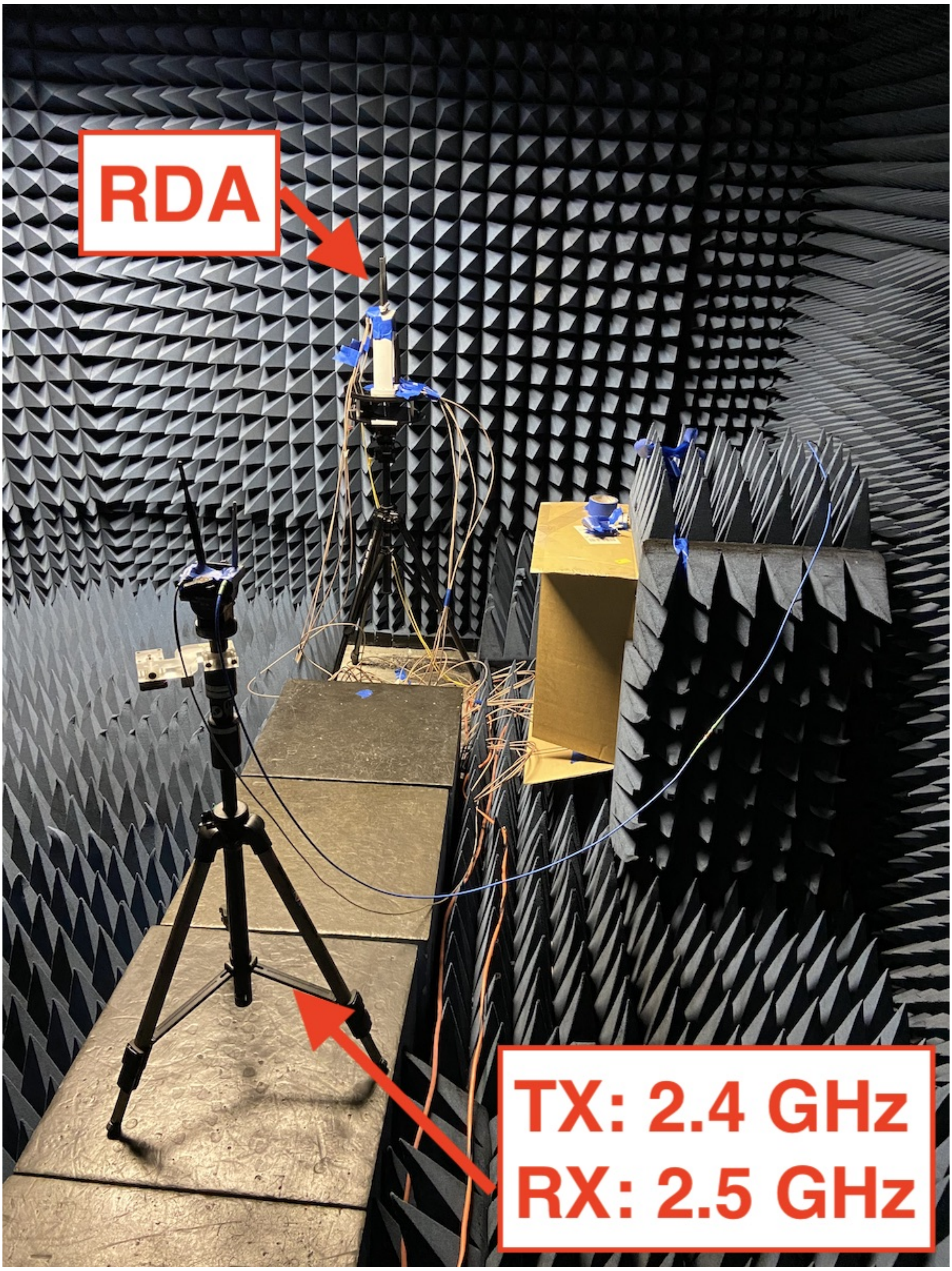


Figure C.9: Photograph of anechoic chamber setup for HRA testing and characterization

Appendix D

DLL-HPC Settling Time Tests

D.1 MATLAB Plotting Scripts

D.1.1 StepResponsePlotter.m

```
clear all
rise_d = 0.0075;      %delta to signal start of rise
fall_d = 0.0075;      %delta to signal end of rise
downsample = 2000;   %downsample factor
bin_size = 2;        %bin size for moving box filter

for testNum = 9
    f = sprintf('180_%d.csv',testNum);

    %read in data
    data = csvread(f,9,0);
    T = array2table(data, 'VariableNames',{'TIME','CH1'});
    %shift time so it starts at 0
    T.TIME = T.TIME - min(T.TIME);

    %create downsampled array by averaging
    for n = 1:((size(T,1)-1)/downsample)
        time(n) = mean(T.TIME((n-1)*downsample + 1 : n*downsample + 1));
        ch1(n) = mean(T.CH1((n-1)*downsample + 1 : n*downsample + 1));
    end
    clear T data
    T = array2table([time' ch1'], 'VariableNames',{'TIME','CH1'});
    clear time
    clear ch1

    %smooth data with moving box filter
    for n = 1:size(T,1)
        cnt = 0;
        total = 0;
        for m = max(1, n-bin_size) : min(n+bin_size, size(T,1))
            total = total + T.CH1(m);
            cnt = cnt + 1;
        end
        T.CH1(n) = total / cnt;
    end
```

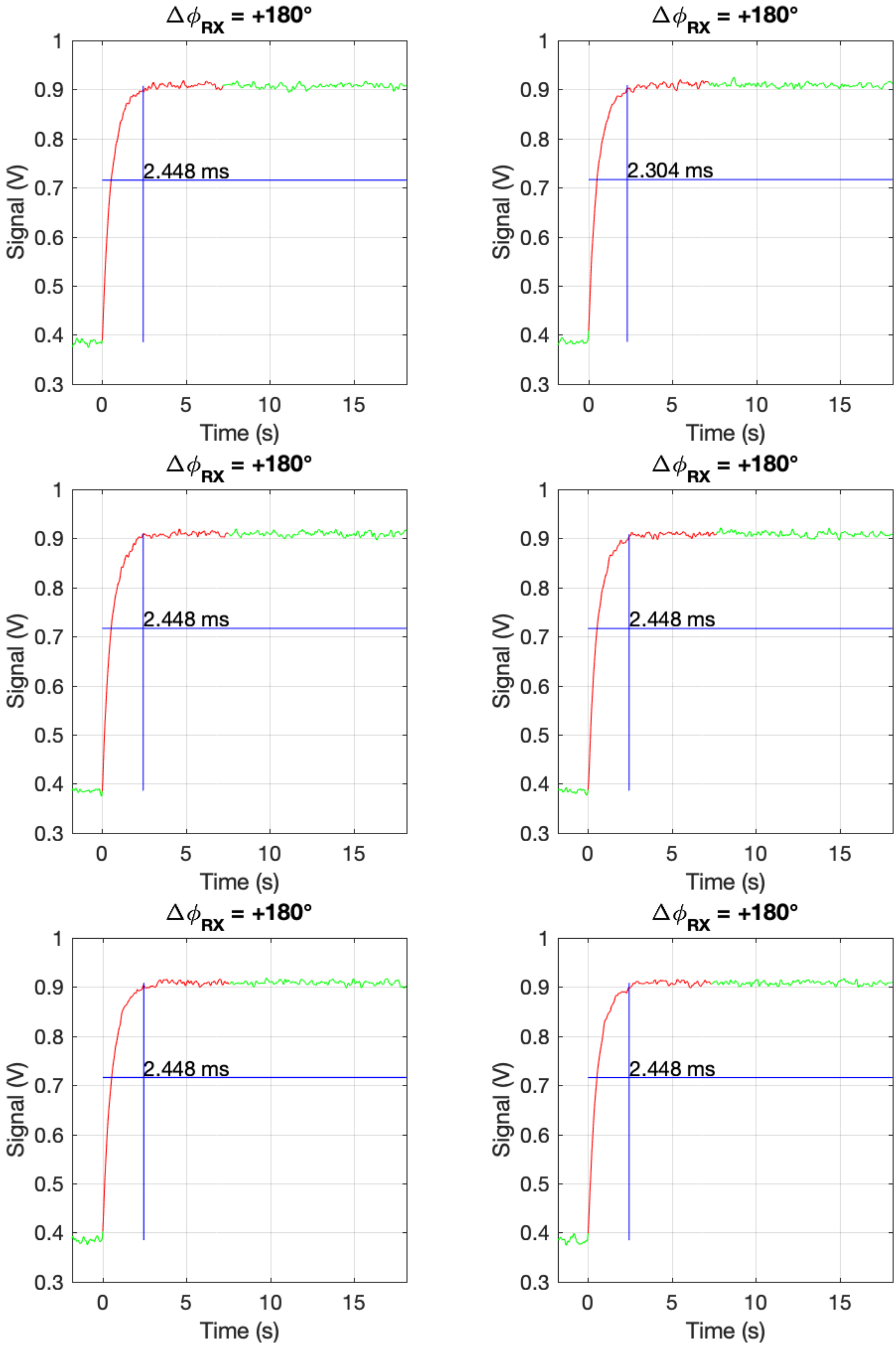
```
%find arrays of deltas
delta = zeros(size(T,1),1);
for n = 2:size(T,1)
    delta(n) = T.CH1(n) - T.CH1(n-1);
end
figure(2)
clf
scatter(T.TIME,abs(delta(:,1)))
grid on

%find end points for start/end values:
beginning = 0;
for n = 1:size(T,1)
    if beginning == 0
        if abs(delta(n)) >= rise_d
            beginning = n;
        end
    end
end
ending = 0;
for n = size(T,1):-1:1
    if ending == 0
        if abs(delta(n)) >= fall_d
            ending = n;
        end
    end
end
%calcualte starting/ending values
start_val = mean(T.CH1(1:beginning-10));
end_val = mean(T.CH1(ending+50:size(T.CH1,1)));
delta_val = end_val - start_val;
%find time constant
for n = 2:size(T.CH1,1)
    if delta_val > 0
        if T.CH1(n) > (start_val + 0.632*delta_val)
            if T.CH1(n-1) < (start_val + 0.632*delta_val)
                crossing = n;
            end
        end
    else
        if T.CH1(n) < (start_val + 0.632*delta_val)
            if T.CH1(n-1) > (start_val + 0.632*delta_val)
                crossing = n;
            end
        end
    end
end
T.TIME = T.TIME - T.TIME(beginning);
T.TIME = T.TIME * 1e3;
tau = T.TIME(crossing) - T.TIME(beginning);
fprintf('settling time: %f\n',4.5*tau);


%plot
fig = figure(1);
clf
```

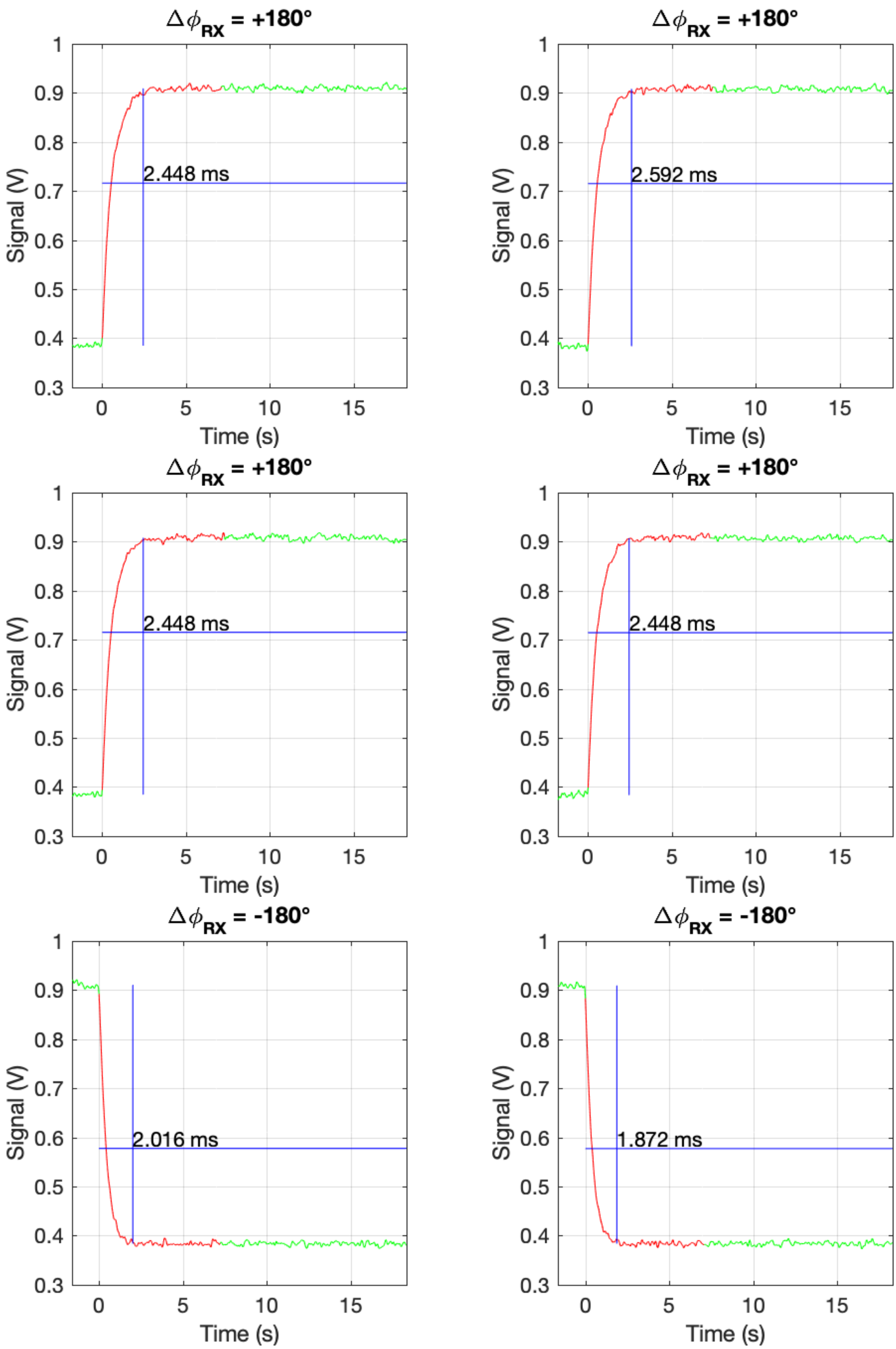
```
    fig.Units = 'inches';
    fig.Position = [1 1 3 3];
    plot(T.TIME, T.CH1, 'r', 'DisplayName', 'Ch. 1');
    hold on
    plot(T.TIME(1:beginning), T.CH1(1:beginning), 'g');
    plot(T.TIME(ending+200:size(T,1)), T.CH1(ending+200:size(T,1)), 'g');
    plot([0 max(T.TIME)],[start_val+0.632*delta_val
    start_val+0.632*delta_val],'b');
    plot([T.TIME(beginning)+4.5*tau T.TIME(beginning)+4.5*tau],[start_val
    end_val],'b');
    text(T.TIME(beginning)+4.5*tau, start_val+0.632*delta_val+0.02,
    sprintf('%1.3f ms',4.5*tau));
    hold off
    xlabel('Time (s)');
    ylabel('Signal (V)');
    xlim([min(T.TIME) max(T.TIME)]);
    grid on
    if f(4) == '_'
        title('\Delta\phi_{RX} = +180\circ');
        print(sprintf('StepTestR%d.png',testNum),'-dpng');
    else
        title('\Delta\phi_{RX} = -180\circ');
        print(sprintf('StepTestF%d.png',testNum),'-dpng');
    end
end
```

D.1.2 LinearSweepResponsePlotter.m

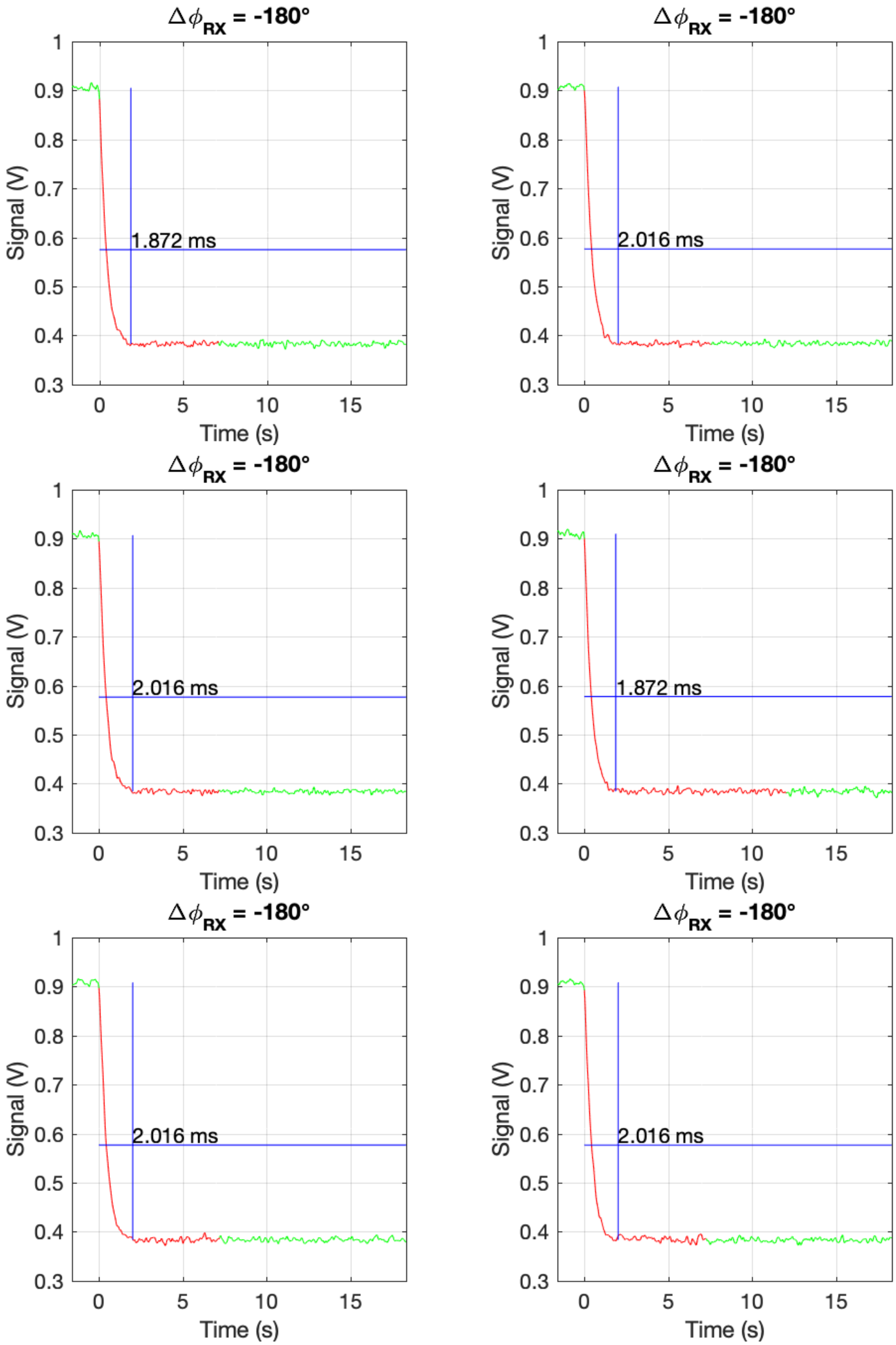
```
clear all
for testNum = 0:18
    % store title
    switch testNum
        case {0,1}
            t = '$\pm50^\circ/sec$';
        case {2,3}
            t = '$\pm120^\circ/sec$';
        case {4,5,8,9}
            t = '$\pm300^\circ/sec$';
        case {6,7}
            t = '$\pm450^\circ/sec$';
        case {10,11}
            t = '$\pm600^\circ/sec$';
        case {12,13}
            t = '$\pm1000^\circ/sec$';
        case {14,15}
            t = '$\pm3000^\circ/sec$';
        case 16
            t = '$\pm6000^\circ/sec$';
        case 17
            t = '$\pm15000^\circ/sec$';
        case 18
            t = '$\pm30000^\circ/sec$';
    end


    % read in data
    f = sprintf('FullSweep_%d.csv',testNum);
    data = csvread(f,9,0);
    T = array2table(data, 'VariableNames',{'TIME','CH1'});
    % shift time so it starts at 0
    T.TIME = T.TIME - min(T.TIME);
    % plot figure
    fig = figure(1);
    clf
    fig.Units = 'inches';
    fig.Position = [1 1 3 3];
    plot(T.TIME, T.CH1)
    xlabel('Time (s)')
    ylabel('V_{ctrl} (V)')
    title(t,'Interpreter','latex','fontsize',14);
    drawnow
    print([f(1:length(f)-4) '.png'],'-dpng');
end
```

D.2 DLL-HPC measured closed-loop step response

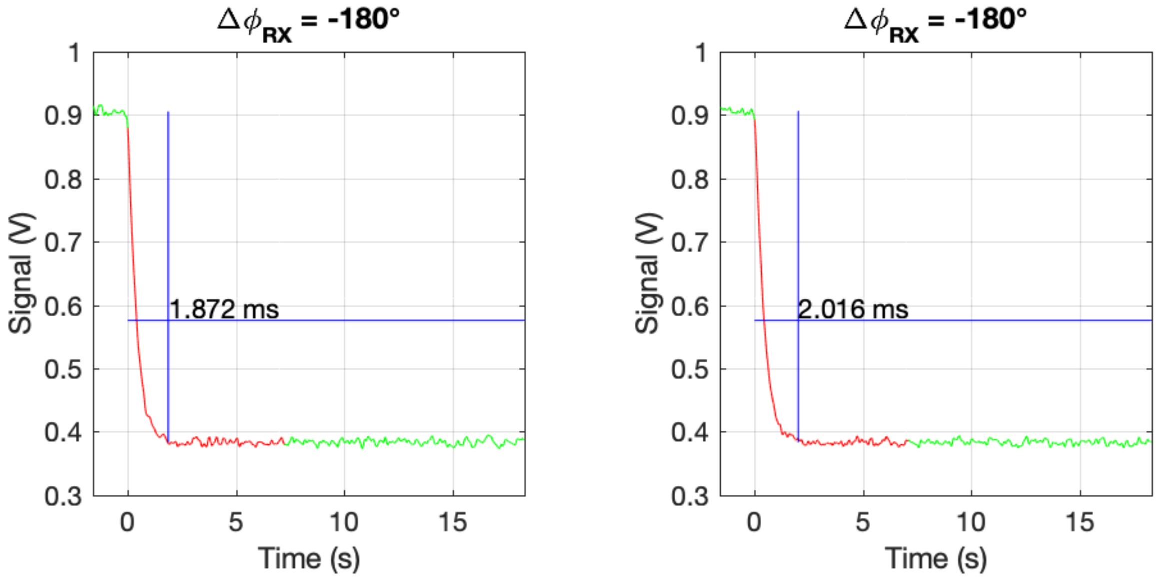




Δϕ RX = +180°
Signal (V)
Time (s)
2.448 ms
Δϕ RX = +180°
Signal (V)
Time (s)
2.592 ms
Δϕ RX = +180°
Signal (V)
Time (s)
2.448 ms
Δϕ RX = +180°
Signal (V)
Time (s)
2.448 ms
Δϕ RX = -180°
Signal (V)
Time (s)
2.016 ms
Δϕ RX = -180°
Signal (V)
Time (s)
1.872 ms



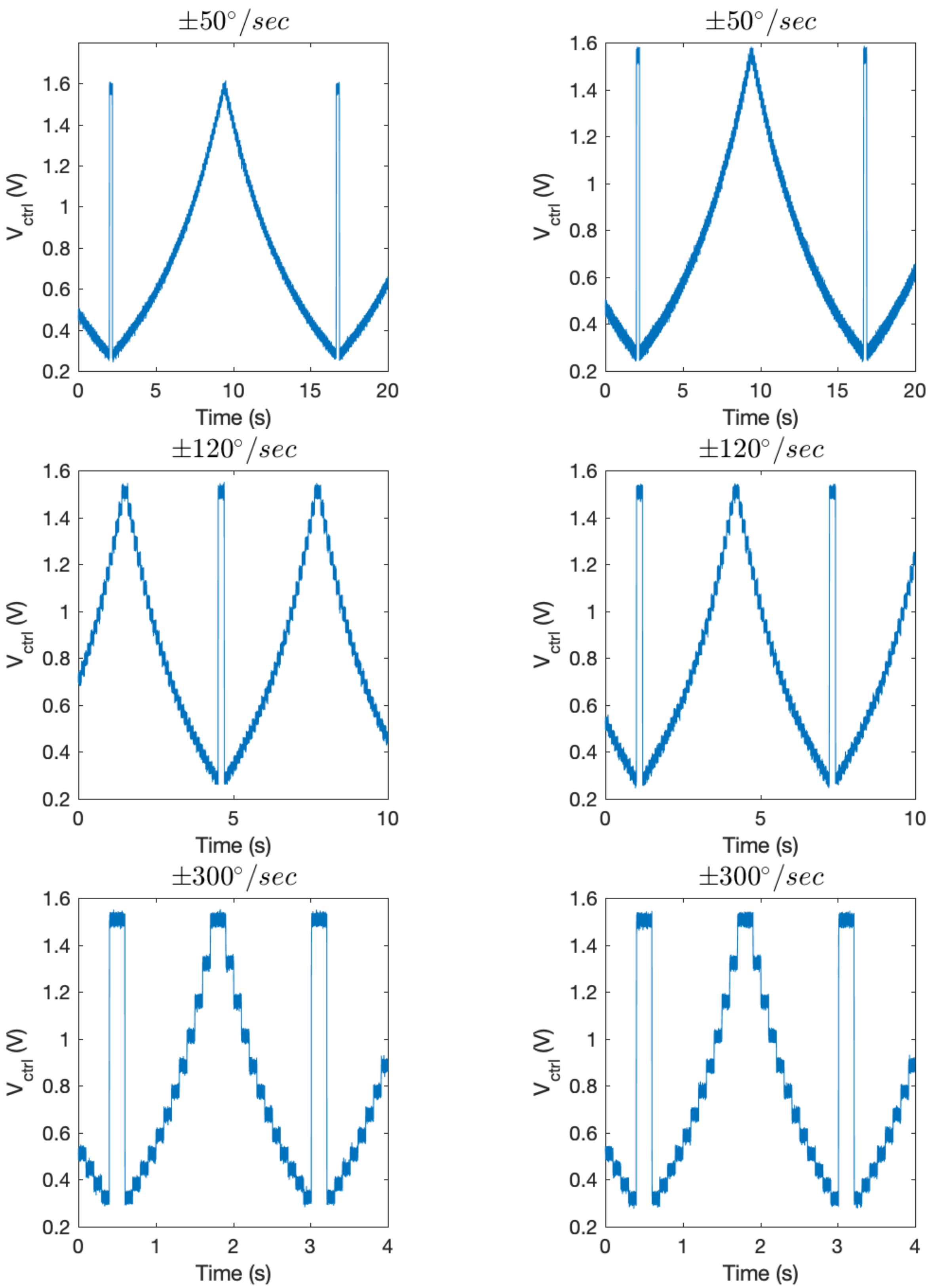
ΔφRX = -180°
Signal (V)
Time (s)
1.872 ms
ΔφRX = -180°
Signal (V)
Time (s)
2.016 ms
ΔφRX = -180°
Signal (V)
Time (s)
2.016 ms
ΔφRX = -180°
Signal (V)
Time (s)
1.872 ms
ΔφRX = -180°
Signal (V)
Time (s)
2.016 ms
ΔφRX = -180°
Signal (V)
Time (s)
2.016 ms

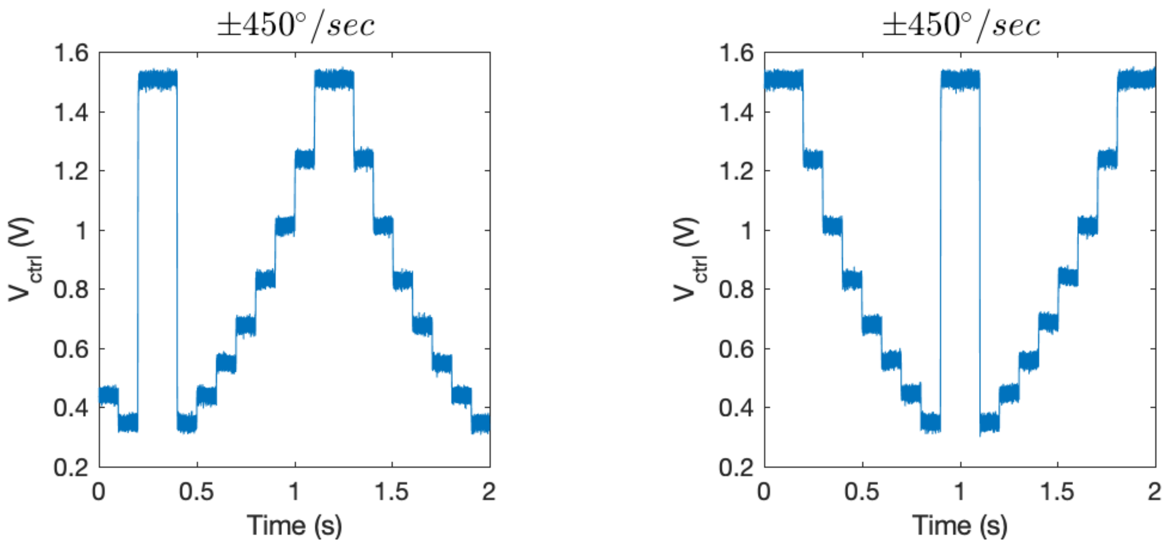


$\Delta\phi_{RX}$ = -180°
1.872 ms
Signal (V)
Time (s)
$\Delta\phi_{RX}$ = -180°
2.016 ms
Signal (V)
Time (s)

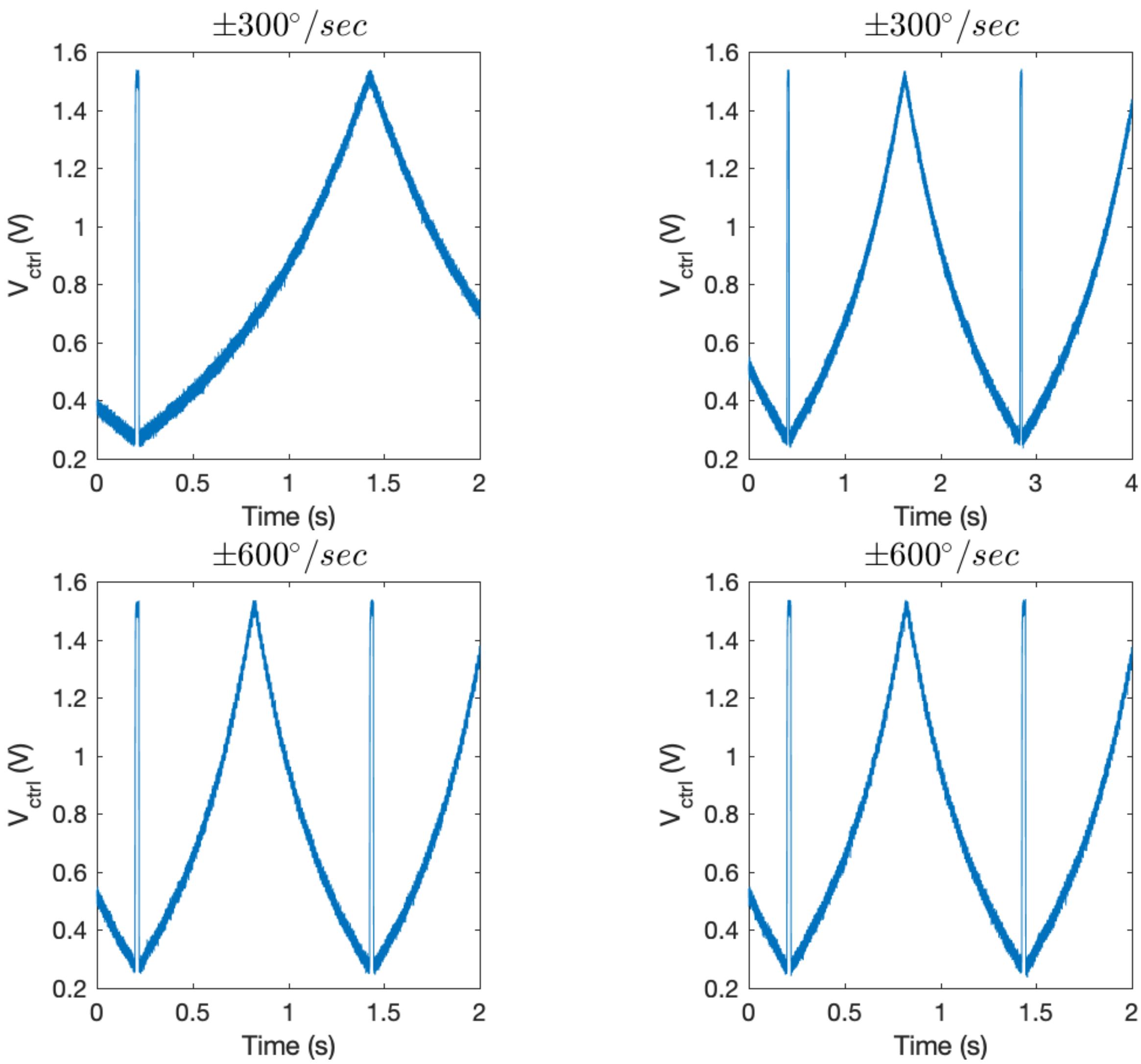
D.3 DLL-HPC measured closed-loop response to linear input changes

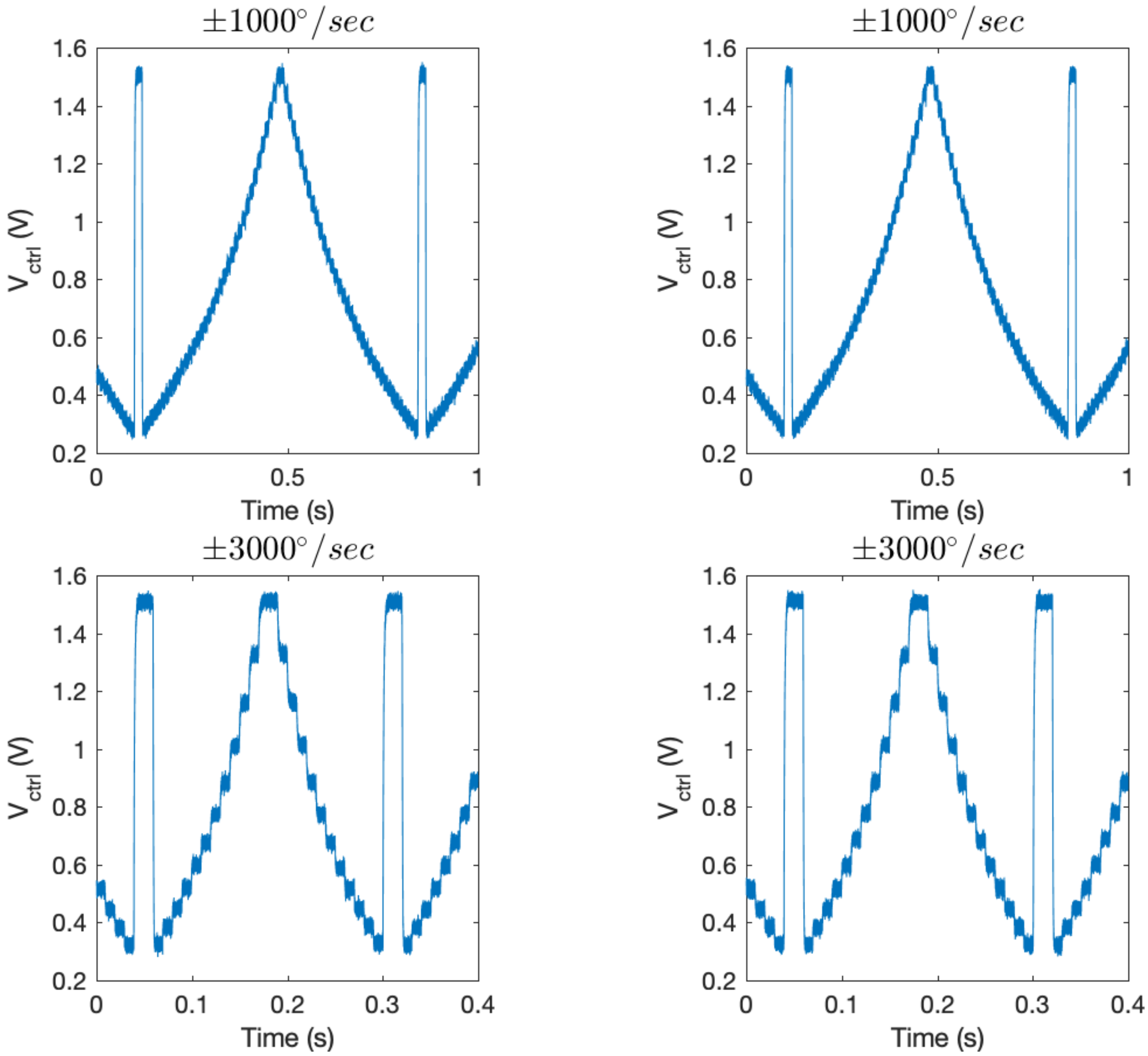
D.3.1 Output sample rate of source set to 10 Hz



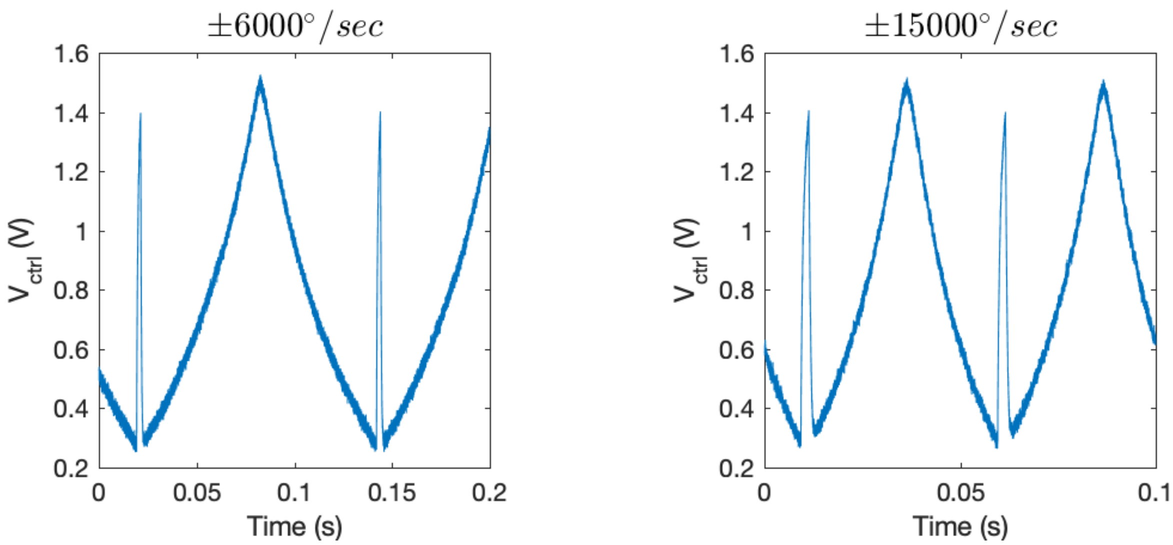


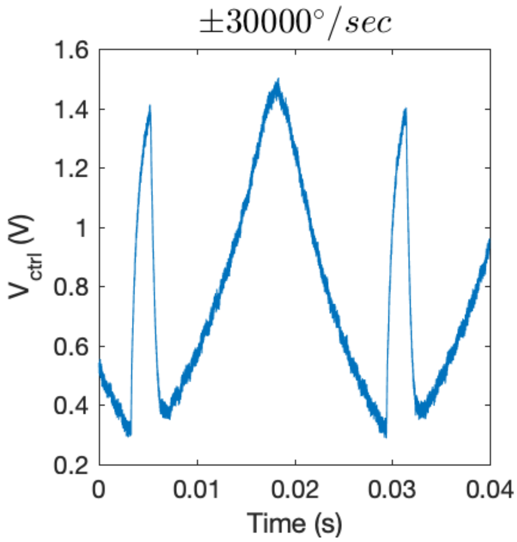
D.3.2 Output sample rate of source set to 100 Hz





D.3.3 Output sample rate of source set to 1 kHz





$\pm 30000°/sec$
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
V_{ctrl} (V)
0
0.01
0.02
0.03
0.04
Time (s)

Appendix E

Additional Integrated Circuit Designs and Simulations

E.1 Digital Logic Elements

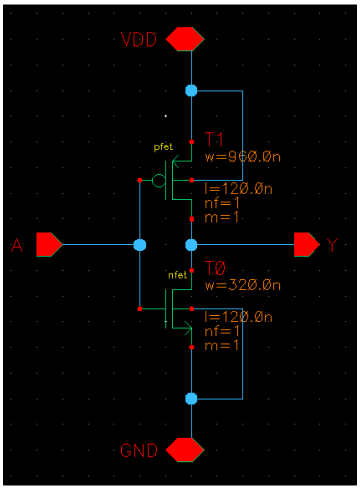


Figure E.1: CMOS inverter transistor-level schematic

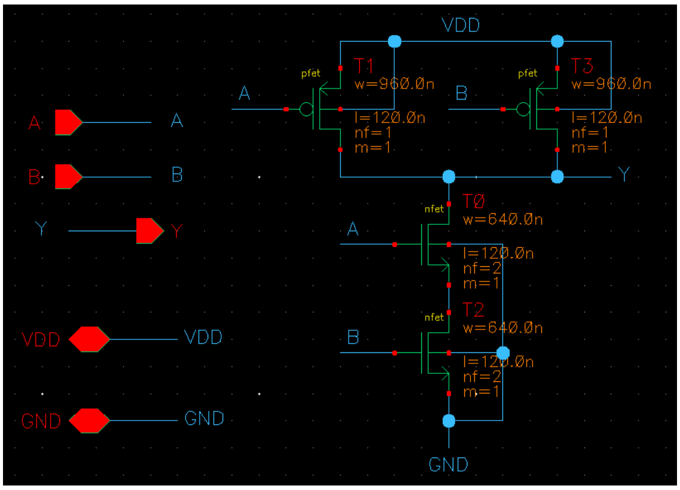


Figure E.2: CMOS NAND2 transistor-level schematic

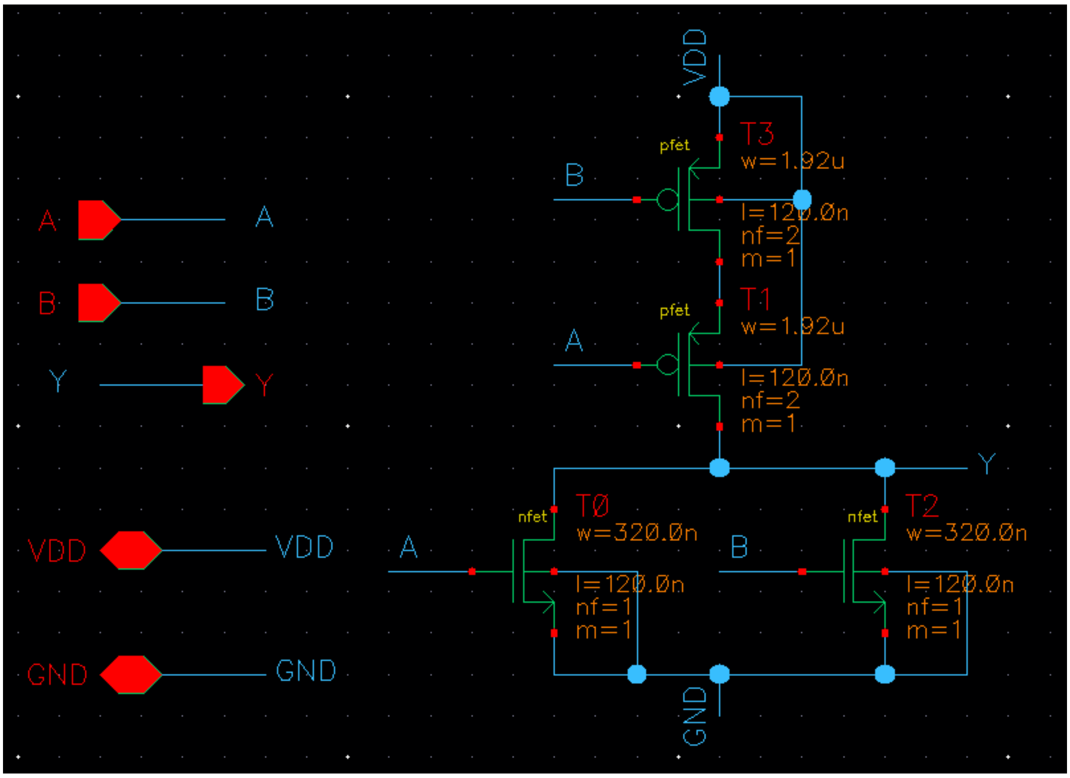


Figure E.3: CMOS NOR2 transistor-level schematic

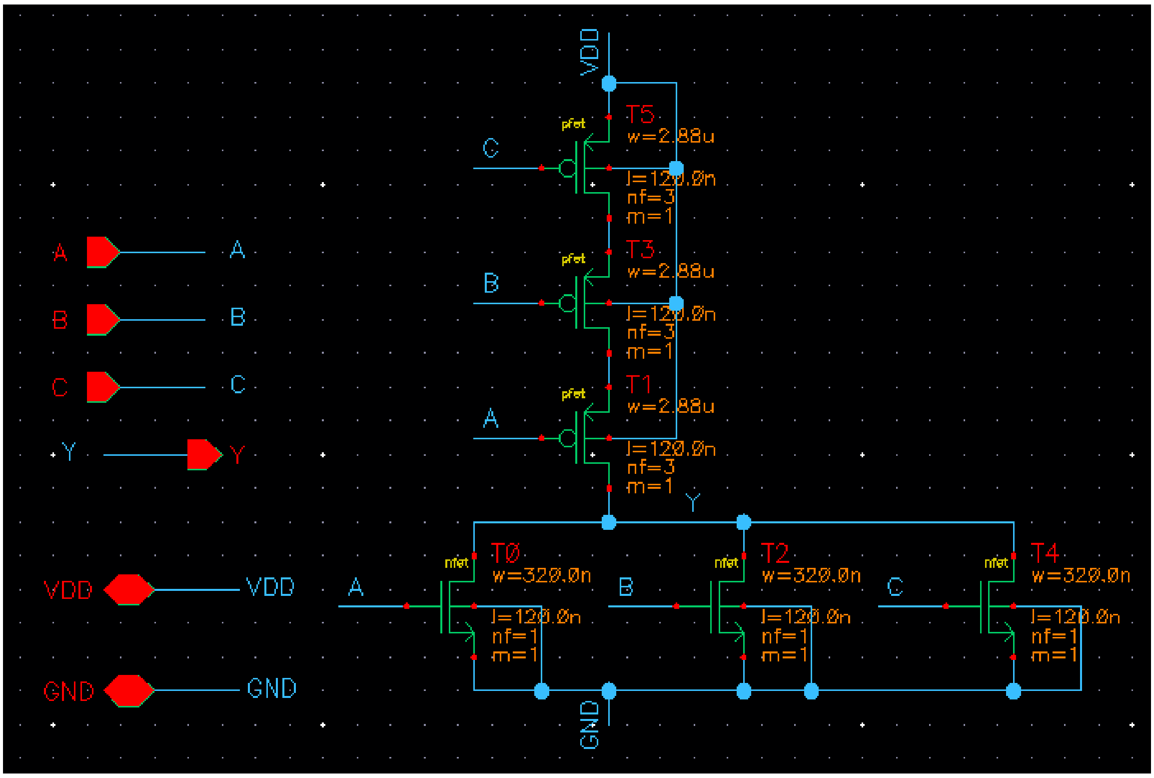


Figure E.4: CMOS NOR3 transistor-level schematic

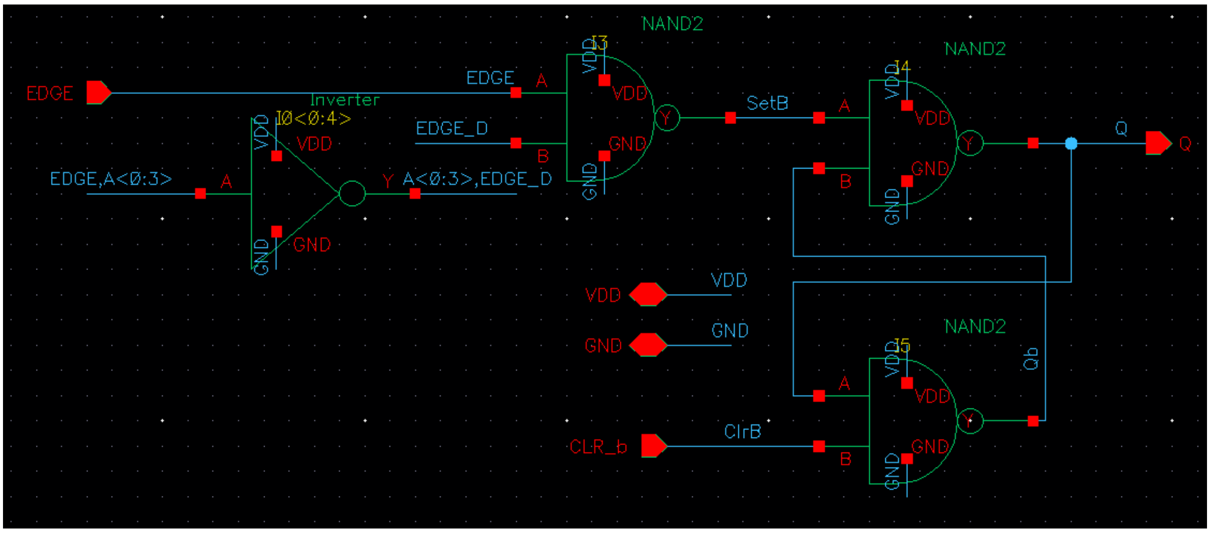


Figure E.5: CMOS active-high edge latch schematic

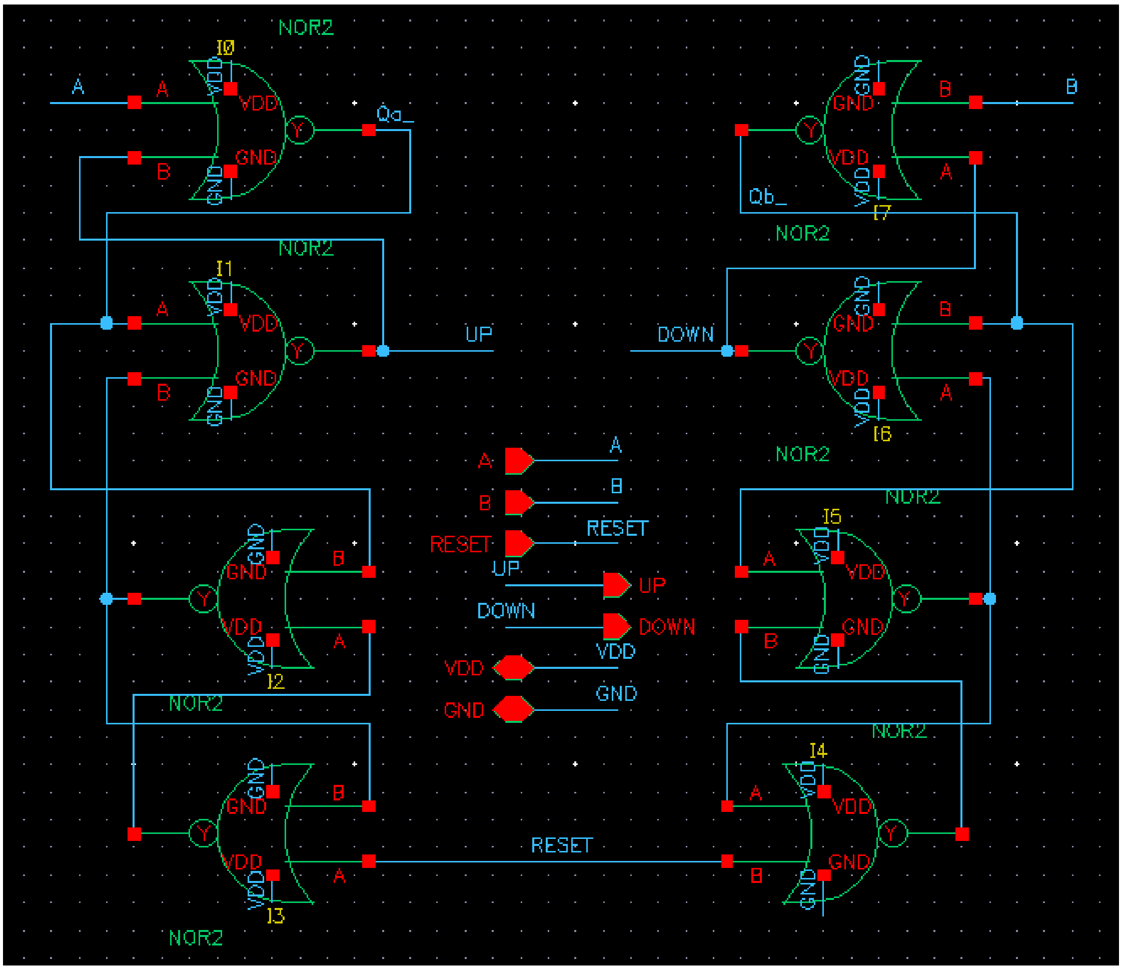


Figure E.6: CMOS NOR based PFD schematic

E.2 Comparator Simulations

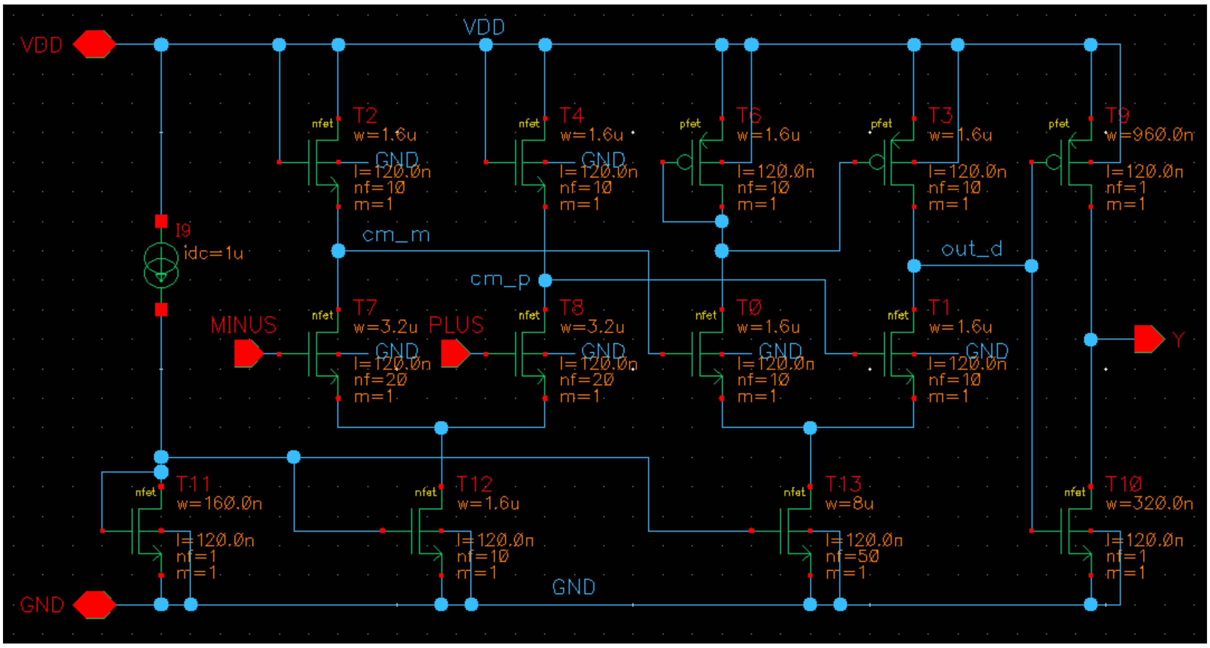


Figure E.7: Cadence Virtuoso transistor-level comparator schematic

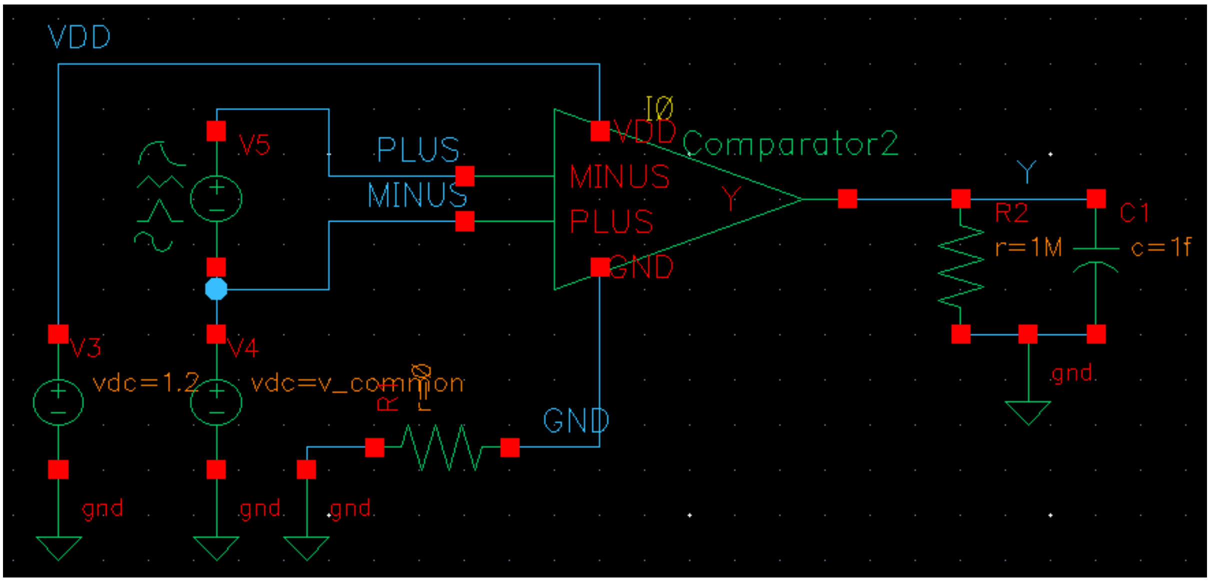


Figure E.8: Cadence Virtuoso schematic for comparator characterization tests

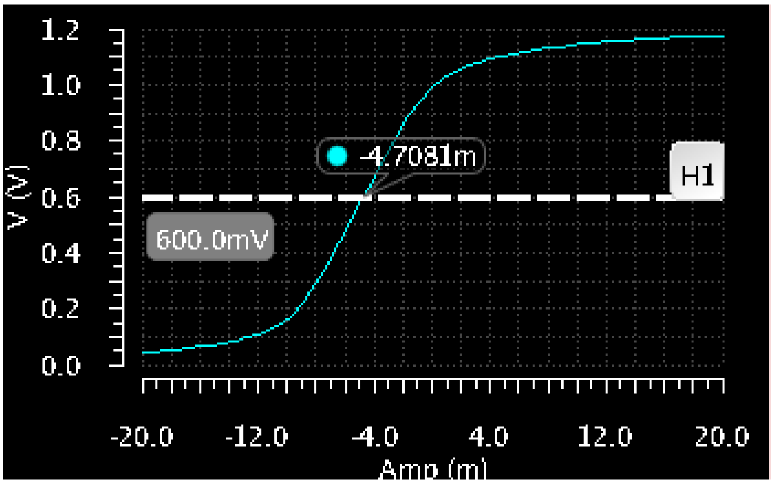


Figure E.9: Comparator simulation showing output voltage vs. input differential voltage $V_{PLUS} - V_{MINUS}$ with balance point marked

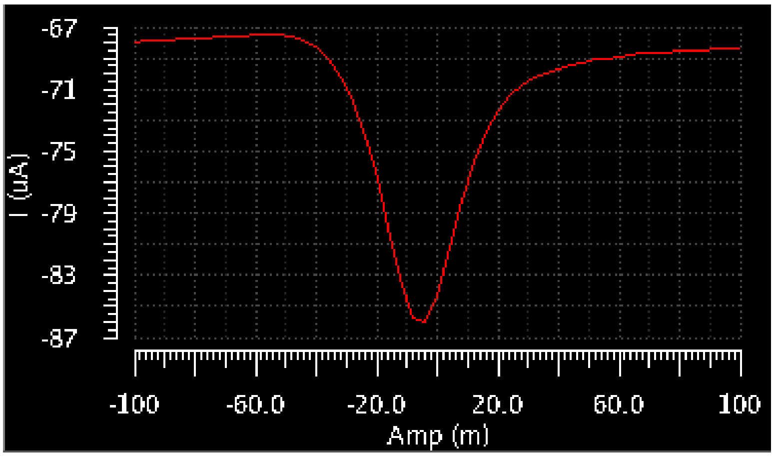
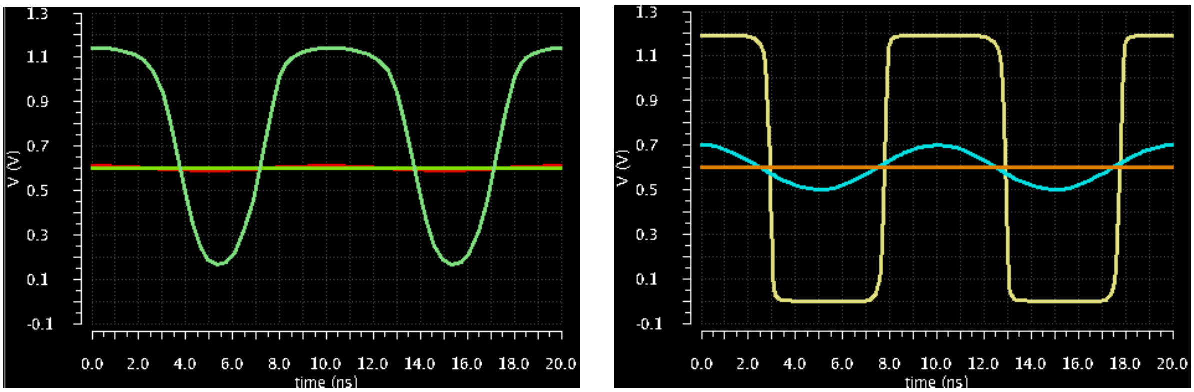


Figure E.10: Comparator simulation showing power supply current vs. input differential voltage $V_{PLUS} - V_{MINUS}$



(a) (b)

Figure E.11: Comparator transient simulation showing output for (a) 10 mV amplitude sinusoidal input (b) 100 mV amplitude sinusoidal input

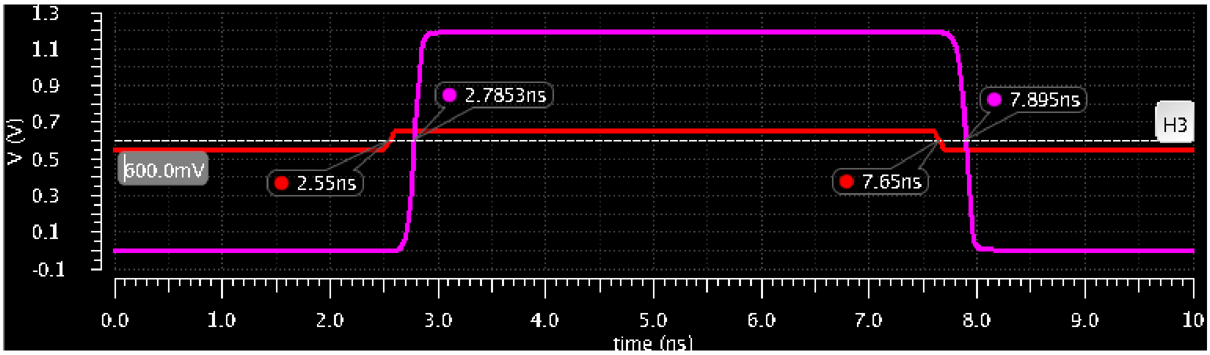


Figure E.12: Comparator simulation showing responds time of comparator to 100 mV step change in input differential voltage

Figures E.7 and E.8 show the comparator's implementation and characterization simulation setup. Fig. E.9 shows that an input voltage differential of approximately 4 mV corresponds to the the logic threshold crossing of 50% VDD, or 0.6 V. Fig. E.10 shows the current supplied by the 1.2 V power supply for different input voltage differentials, with a peak power consumption of 103 μW and an average power consumption of 82 μW. Fig. E.11 shows transient simulations of 500 MHz sinusoidal inputs with 10 mV and 100 mV amplitudes, demonstrating appropriate behavior. Fig. E.12 shows a step input applied to the comparator to determine the total propagation delay for both rising and falling inputs: 235 ps and 245 ps respectively, or a 240 ps average propagation delay.

E.3 Charge Pump Simulations

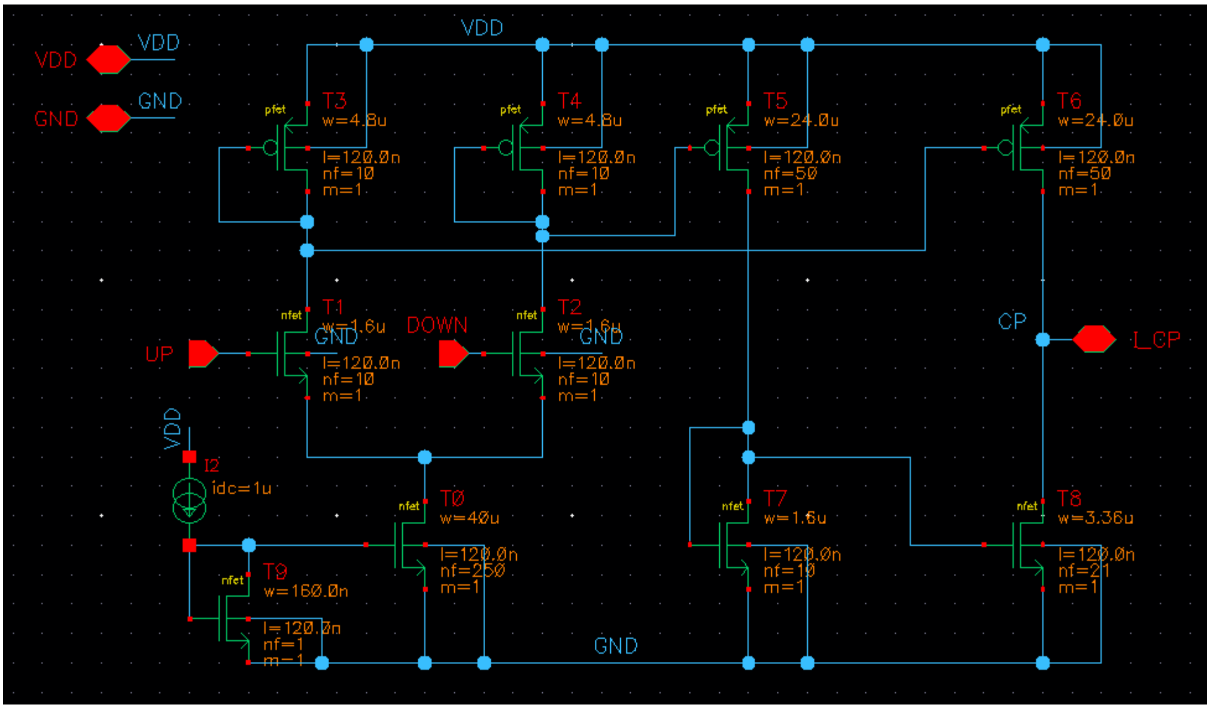


Figure E.13: Cadence Virtuoso transistor-level charge pump schematic

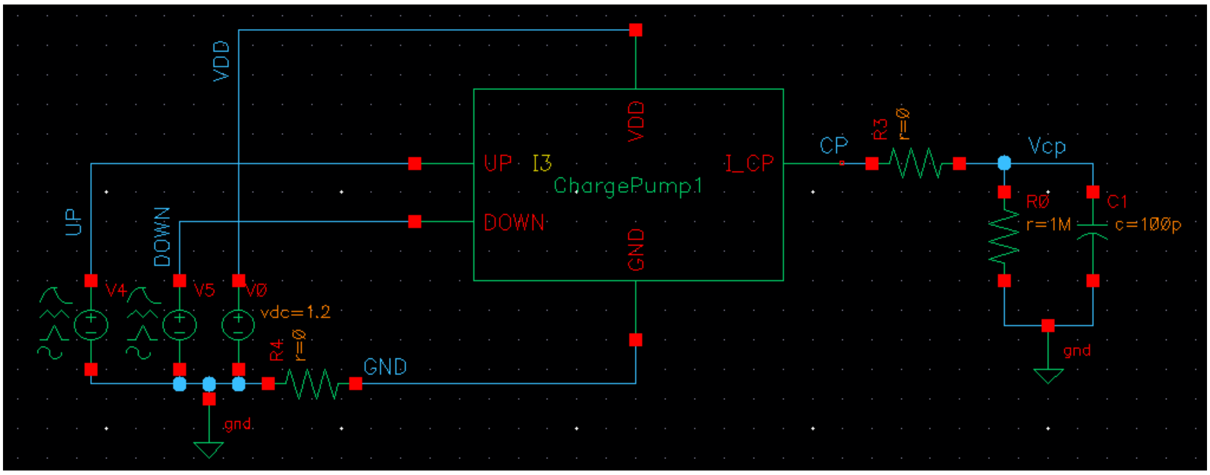


Figure E.14: Cadence Virtuoso schematic for charge pump characterization tests

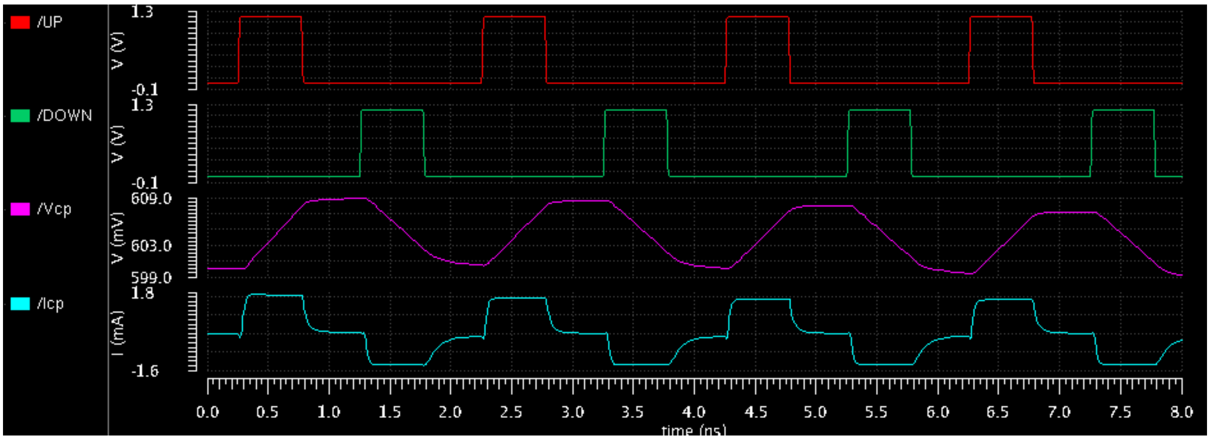


Figure E.15: Charge pump transient simulation showing equal duration UP and $DOWN$ pulses are well balanced and leave V_{CP} at the starting value

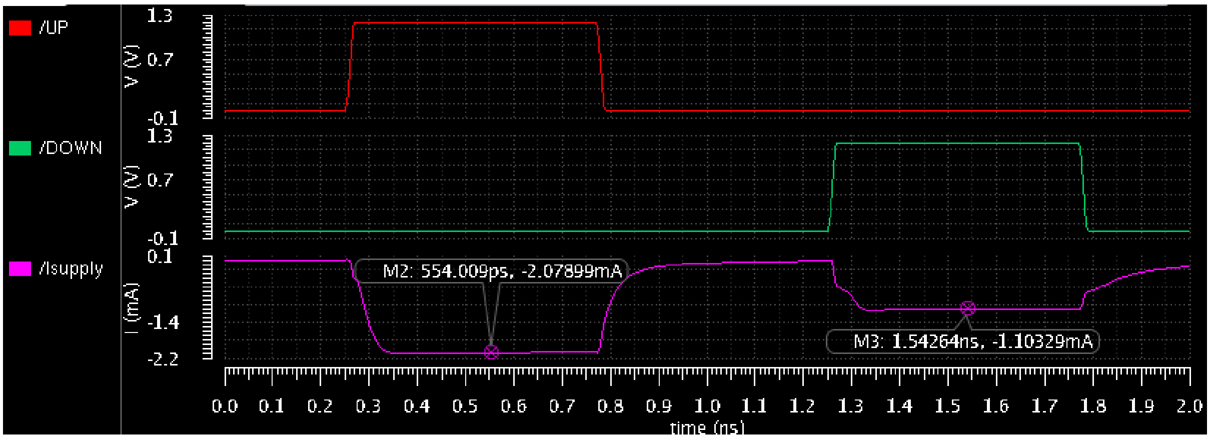


Figure E.16: Charge pump transient simulation showing power supply current during UP and $DOWN$ pulses

Figures E.13 and E.14 show the charge pump's implementation and characterization simulation setup. Fig. E.15 shows a transient simulation where equal duration UP and $DOWN$ pulses are applied; as V_{CP} ends this simulation with almost the same value it started with, the sink and source currents are well-matched. Fig. E.16 shows a transient simulation where the 1.2 V power supply currents are measured during UP and $DOWN$ pulses; this allows us to determine that the charge pump circuitry itself consumes 1.103 mA or 1.324 mW of power when either UP or $DOWN$ is active, as the charge pump is sinking current directly from V_{CP} and not from the power supply during a $DOWN$ pulse. Additionally, DC simulations found that the charge pump circuitry consumes 1.3 μW of power when neither UP or $DOWN$ are active.

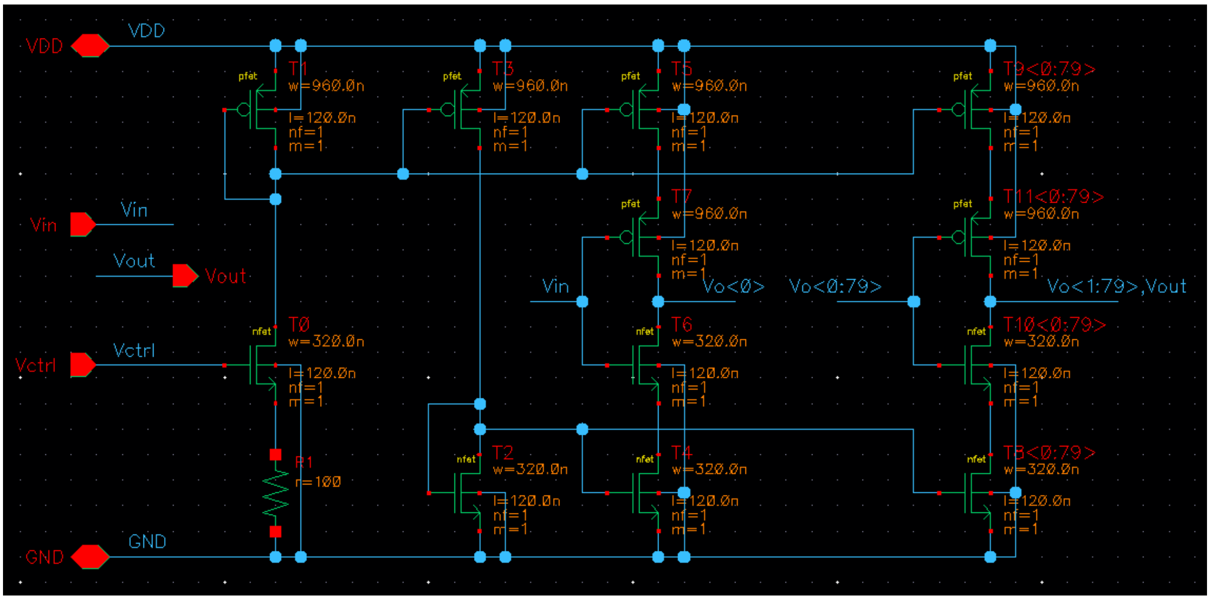


Figure E.17: Cadence Virtuoso transistor-level voltage controlled delay line schematic

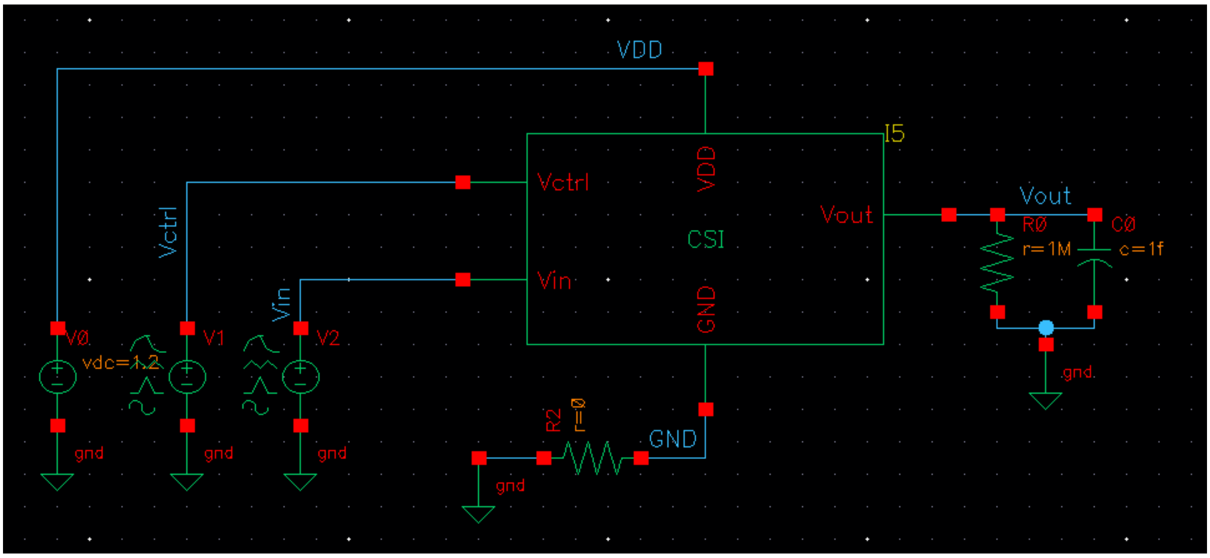


Figure E.18: Cadence Virtuoso schematic for voltage controlled delay line characterization tests

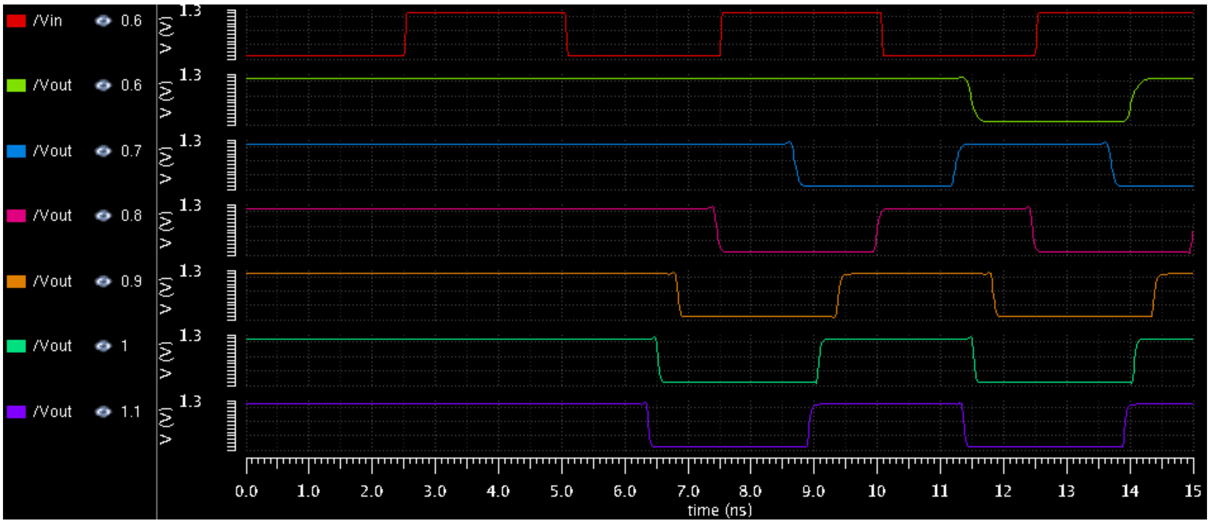


Figure E.19: Voltage controlled delay line transient simulation showing signal delay for various control voltages

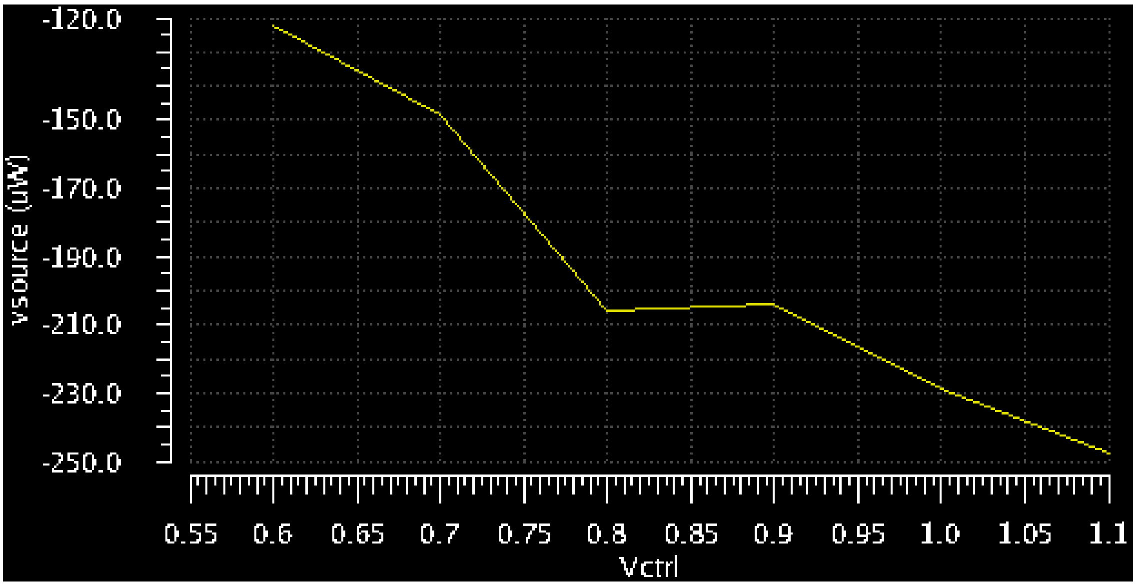


Figure E.20: Voltage controlled delay line average power consumption vs. control voltage V_{ctrl} over operating range

Figures E.17 and E.18 show the voltage controlled delay line's implementation and characterization simulation setup. Fig. E.19 shows that control voltages of 0.6 V and 1.1 V correspond to a full phase rotation of 360°; these voltages allow the R-PFD thresholds to be set to values where the charge pump and comparators can operate well. Fig. E.20 shows the average power consumption of the current-starved inverter circuit for different control voltages with a range of 120 μW to 245 μW; the circuit consumes more power with a higher control voltage, as this leads to a much higher bias current for each of the 80 inverters in the cascaded chain.

E.5 Open-Loop R-PFD Simulations

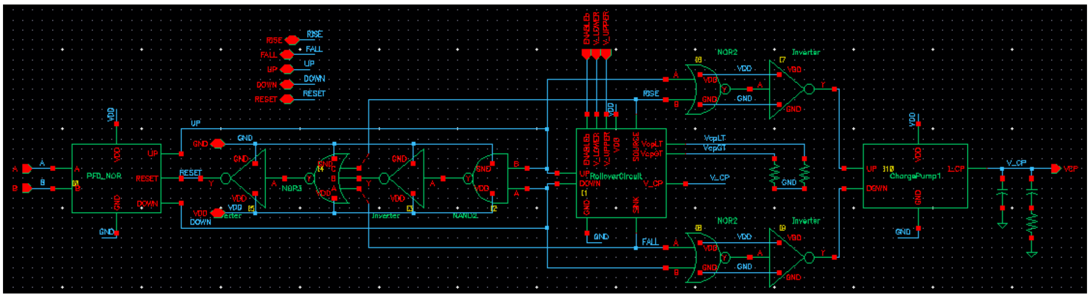


Figure E.21: Cadence Virtuoso open-loop R-PFD system schematic

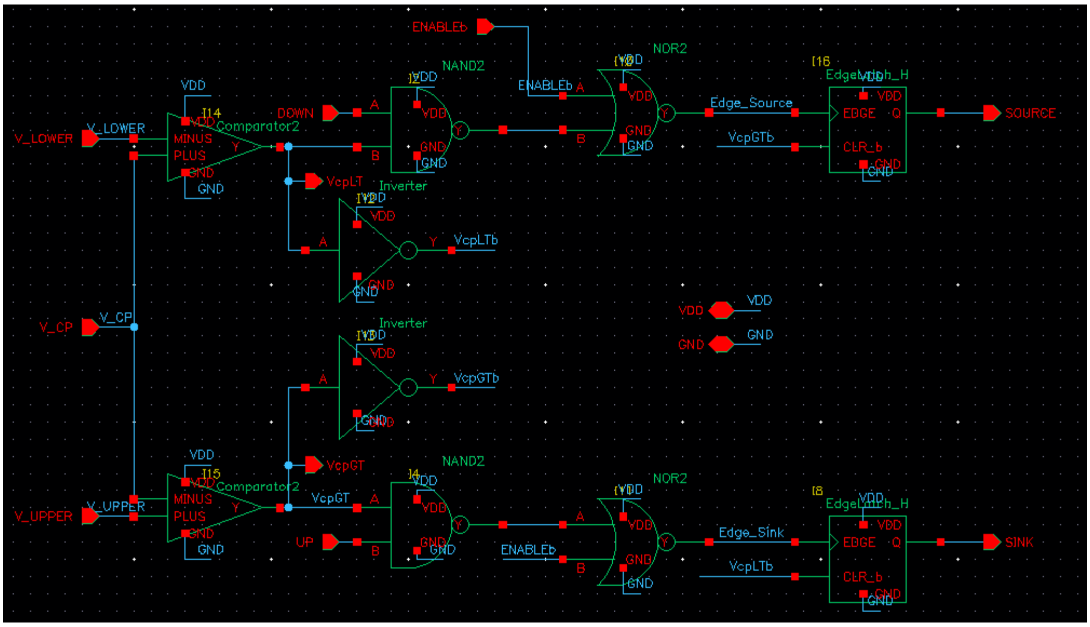


Figure E.22: Cadence Virtuoso rollover feedback circuit system schematic

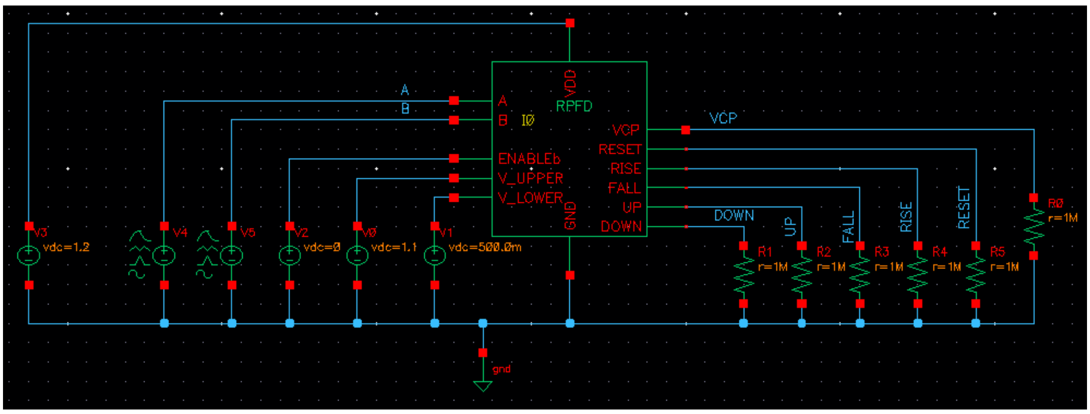


Figure E.23: Cadence Virtuoso schematic for open-loop R-PFD system characterization tests

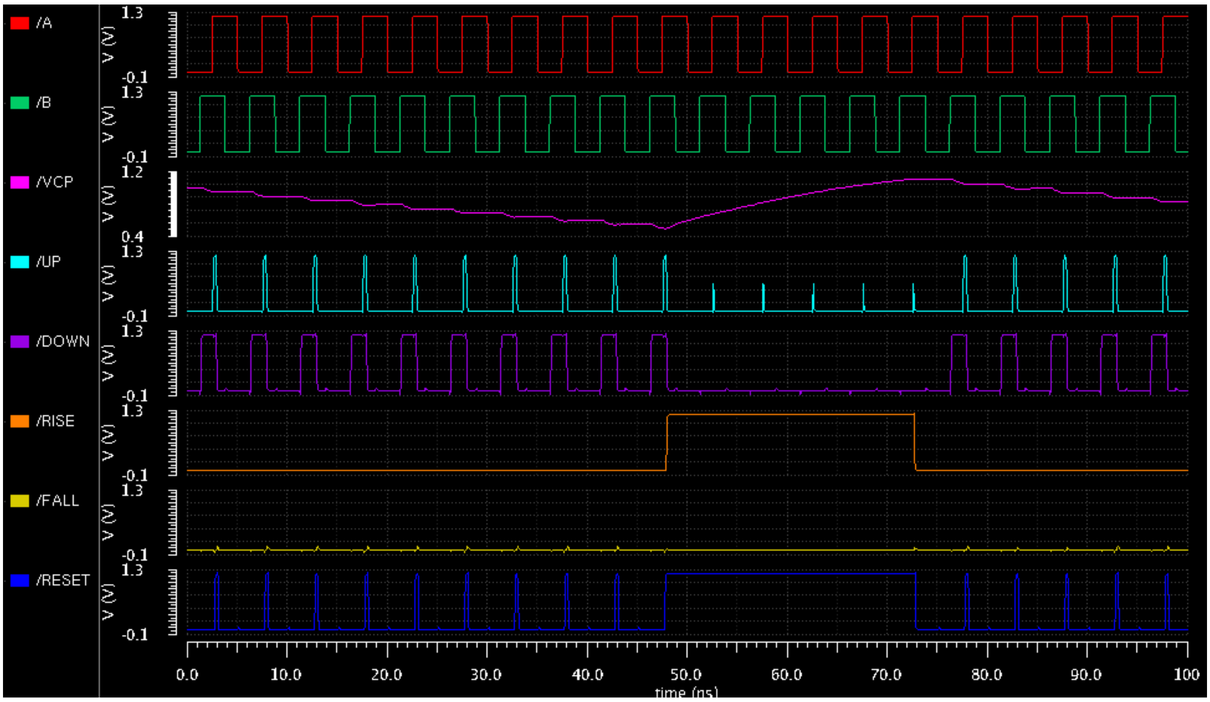


Figure E.24: Open-loop R-PFD simulation for two 200 MHz signals with A lagging B by 1.25 ns

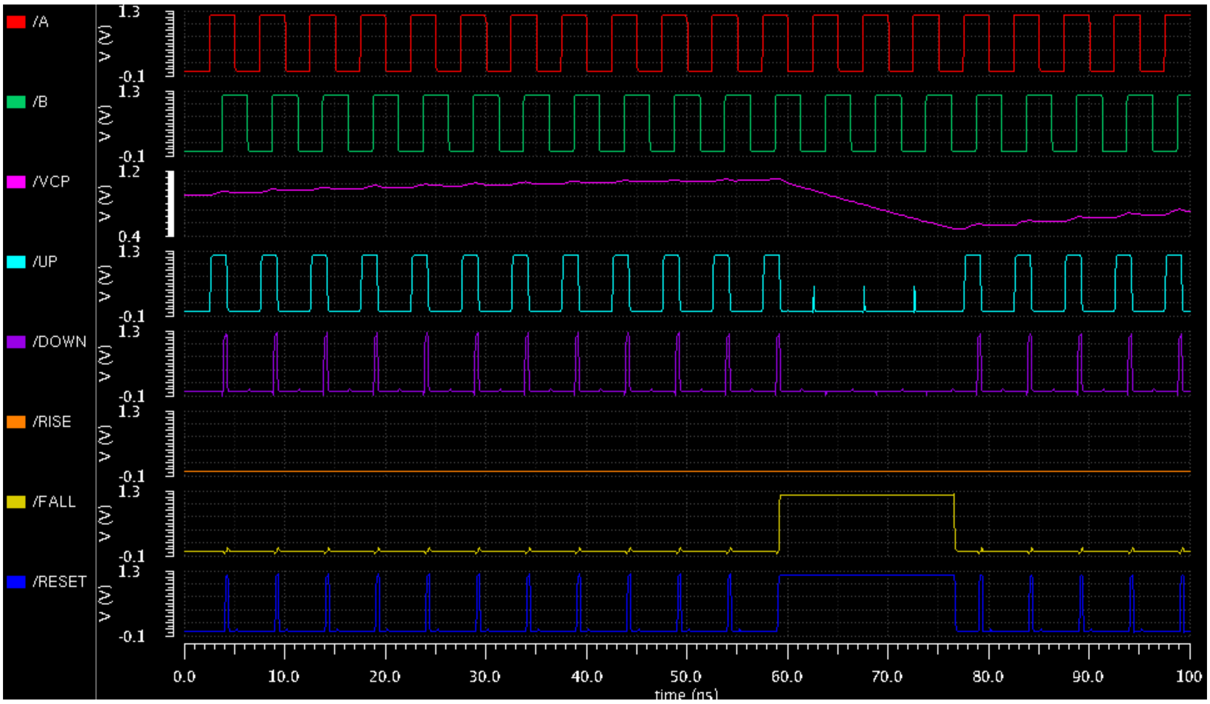


Figure E.25: Open-loop R-PFD simulation for two 200 MHz signals with A leading B by 1.25 ns

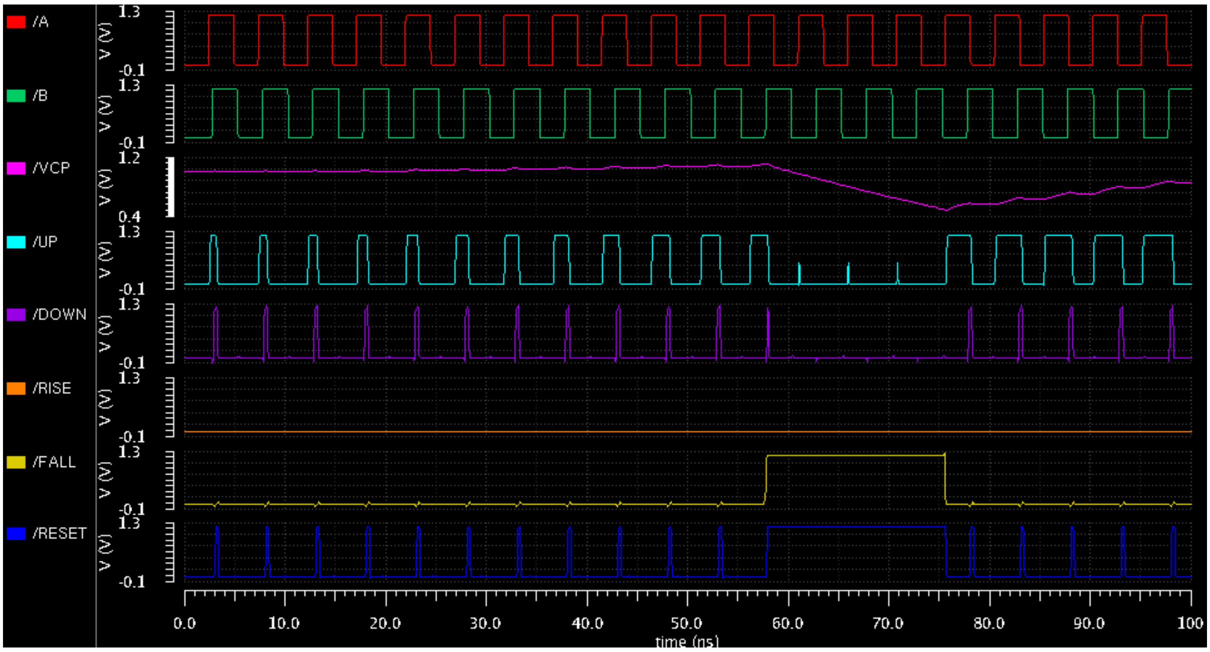


Figure E.26: Open-loop R-PFD simulation for $A = 205$ MHz and $B = 200$ MHz

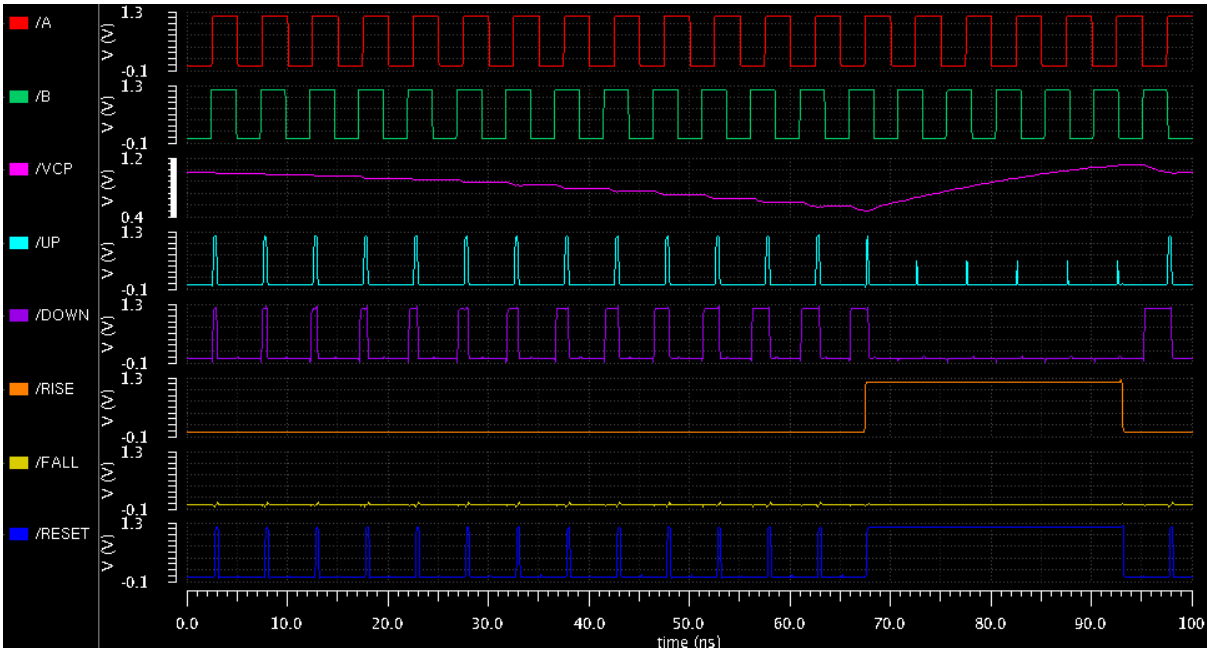


Figure E.27: Open-loop R-PFD simulation for $A = 200$ MHz and $B = 205$ MHz

Figures E.21, E.22, and E.23 show the open-loop R-PFD system implementation and characterization simulation setup. Figures E.24 and E.25 show transient simulations of two 200 MHz signals with a 25% duty cycle difference between the two; these show that the R-PFD rolls over at the chosen voltage thresholds of 0.5 V and 1.1 V appropriately. Figures E.26 and E.27 show transient simulations where A and B are connected to either a 200 MHz or 205 MHz source to show open-loop R-PFD behavior for a frequency difference; again, the R-PFD contains the charge pump loop filter voltage within the chosen thresholds of 0.5 V and 1.1 V. Power calculations from various simulations show that the R-PFD and charge pump circuitry consume 187.2 μW on average when the charge pump is inactive and 1.464 mW on average when the charge pump is active.

E.6 Closed-Loop R-PFD Based DLL Simulations

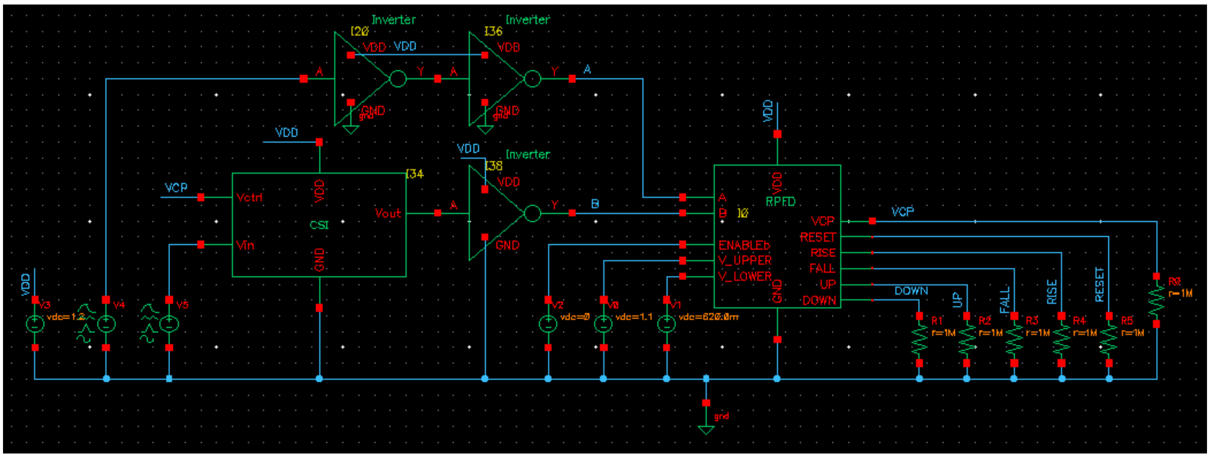


Figure E.28: Cadence Virtuoso schematic for closed-loop R-PFD DLL system characterization

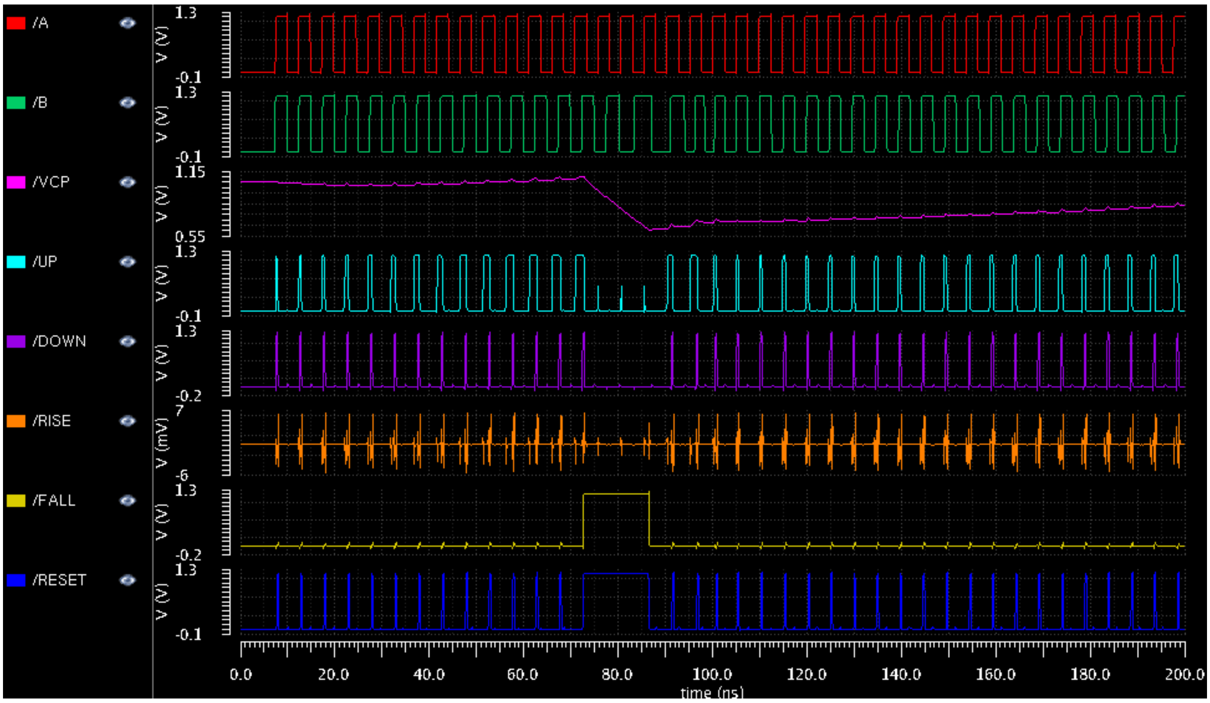


Figure E.29: Closed-loop R-PFD DLL simulation showing a 200 MHz signal tracking a 205 MHz signal through rollover

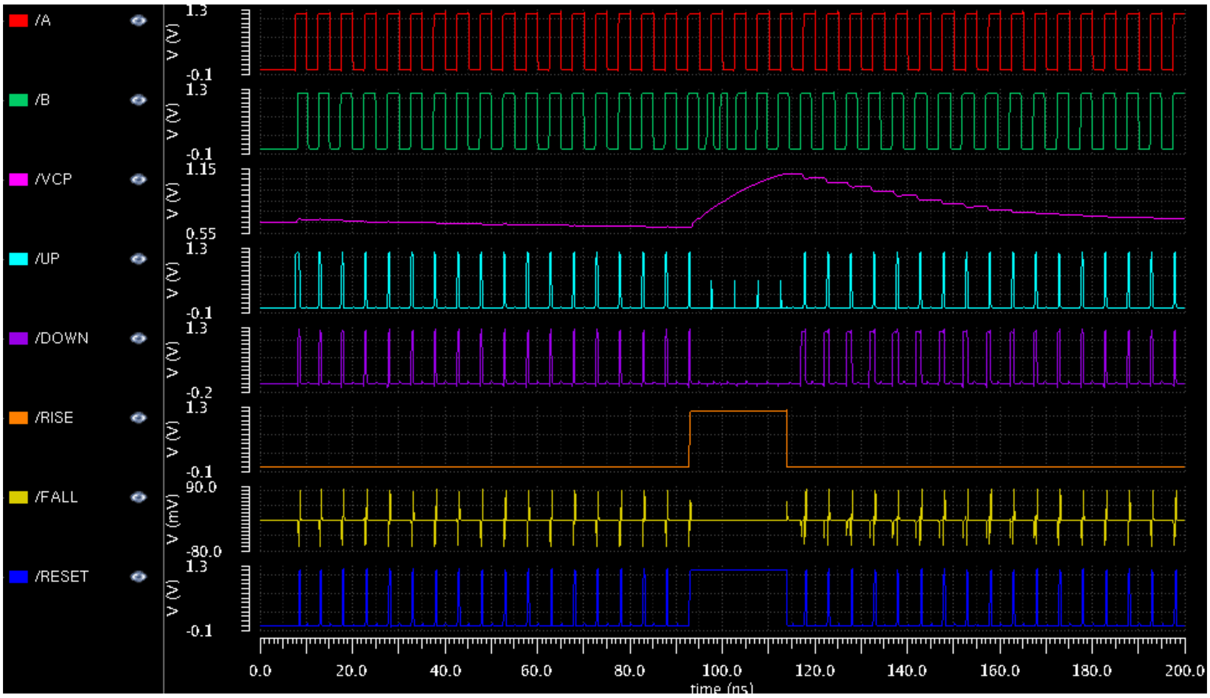


Figure E.30: Closed-loop R-PFD DLL simulation showing a 205 MHz signal tracking a 200 MHz signal through rollover

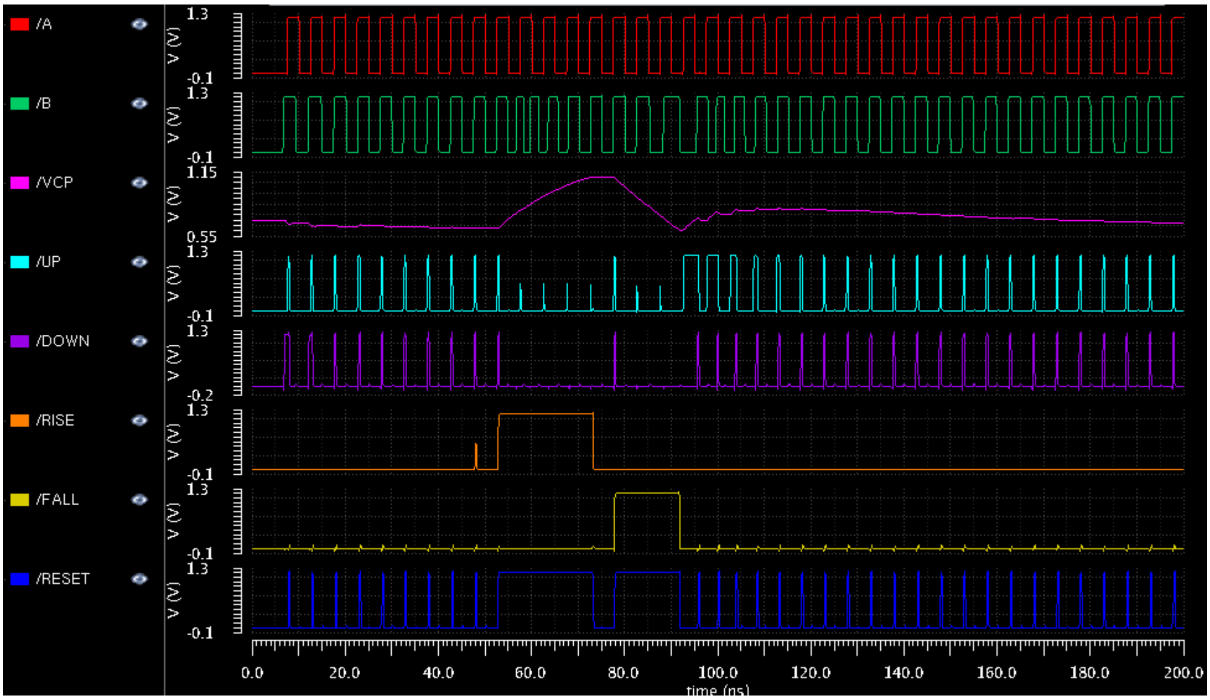


Figure E.31: Closed-loop R-PFD DLL simulation showing accurate tracking after output ringing

Fig. E.28 shows the closed-loop R-PFD DLL characterization setup, which implements feedback of the charge pump's output voltage to control the VCDL module. Figures E.29 and E.30

show transient simulations where two signals of different frequencies are able to accurately track using a pure DLL: Fig. E.29 shows a 200 MHz signal tracking a 205 MHz signal, while Fig. E.30 shows a 205 MHz signal tracking a 200 MHz signal. This 5 MHz frequency difference far exceeds any realistic expectations for received signal relative phase changes but allows for a single transient simulation to show closed-loop operation. Fig. E.31 is indicative of a worst-case scenario in which the UP and $DOWN$ pulses cause the R-PFD to ring, or bounce between thresholds. Extension of the threshold voltages to include a phase shift range of more than 360° can minimize the likelihood of the occurring by providing a valid lock point near each threshold. Further, the total ringing time of 50 ns in this simulation is fast enough that any beamformer error accrued during this time would cause very little error and could be accommodated for in the communication protocol through coding gain or re-transmission.