Computer Networks 196 (2021) 108220

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Pi-Radio v1: Calibration techniques to enable fully-digital beamforming at 60 GHz

Aditya Dhananjay [a,d,*], Kai Zheng [b], Marco Mezzavilla [a,d], Lorenzo Iotti [c], Dennis Shasha [a,d], Sundeep Rangan [a,d]

[a] Pi-Radio Inc., 155 Water Street Unit 4/10, Brooklyn, 11201 NY, USA
[b] University of California San Diego, 9500 Gilman Dr. La Jolla, CA 92093, USA
[c] Nokia, 171 Madison Ave Suite 1100, 10016 NY, USA
[d] New York University, 370 Jay Street, Brooklyn, 11201 NY, USA

ARTICLE INFO

Keywords:
Next-generation wireless
Millimeter-wave
Software-defined-radio
Prototyping
Beamforming
Calibration

ABSTRACT

The Pi-Radio v1 software-defined radio (SDR) platform incorporates a 4-channel fully-digital transceiver board that operates in the 57–64 GHz band and connects to the powerful Xilinx RFSoC-based ZCU111 evaluation board. This paper illustrates various calibration procedures that have been implemented to avoid relying on expensive laboratory equipment and infrastructure like spectrum analyzers, signal generators, or even anechoic chambers. We conclude this paper with a demonstration of beamforming enabled through geometrically determined beamforming weights, thereby demonstrating that the SDR nodes have been calibrated correctly.

1. Introduction and motivation

The millimeter wave (mmWave) bands – roughly corresponding to frequencies above 30 GHz – represent the new frontier of cellular wireless communications [1–6]. This spectrum offers orders of magnitude more available bandwidth than the congested bands in conventional microwave frequencies below 6 GHz. Advances in CMOS RF technology allow to leverage the small wavelengths at mmWave frequencies to enable large numbers of electrically steerable antenna elements to be placed in tiny form factors. Further gains can be realized via adaptive beamforming and spatial multiplexing. Real-world deployments show how the massive bandwidth along with the large numbers of spatial degrees of freedom enable orders of magnitude increases in capacity over current cellular systems [7,8].

However, much remains to be experimentally validated in order to ultimately enable cellular systems to achieve the potential of the mmWave bands [9–11]. Such experimentation is not conducted as widely as the community desires; the problem is mainly one of access. Of course, theory and simulations represent a critical aspect of any research efforts. Nonetheless, the community at large recognizes that more experimentation is urgently needed. The main reason is that existing software defined radio (SDR) systems are either prohibitively expensive or feature technologies that are at least 10 years behind the curve. This is particularly problematic in the millimeter wave (mmWave) bands, which will, quite simply, dominate the wireless world for the next 10–15 years. Systems level experimental work has been monopolized by large industrial labs, while academicians are left out without adequate access to prototyping platforms and the software implementations they need to succeed.

The vision at Pi-Radio[1] is to make SDR systems featuring advanced transceiver technologies available to the research community at reasonable rates [12–14]. We have designed, built, and tested our v1 SDR that features a 4-channel fully-digital transceiver and operates in the 57–64 GHz band. Along these lines, we would like to point out a very impressive mmWave SDR effort which is currently under development at UCSD [15]; this team shares our passion for democratizing wireless research through affordable and advanced SDR systems.

The goal of this paper is to provide a detailed description of the most challenging steps towards getting millimeter-wave hardware to work, i.e., *calibration*. Importantly, our calibration techniques do not need expensive laboratory bench equipment like mmWave spectrum analyzers, signal generators (synthesizers), or even anechoic chambers, as shown in Fig. 1.

The various stages involve: **(a)** calibration of crystal frequency offsets; **(b)** identification of linear operating ranges; **(c)** timing offset

* Corresponding author.
E-mail addresses: aditya.dhananjay@pi-rad.io (A. Dhananjay), kai.zheng@ucsd.edu (K. Zheng), mezzavilla@nyu.edu (M. Mezzavilla), lorenzo.iotti@nokia.com (L. Iotti), shasha@cims.nyu.edu (D. Shasha), srangan@nyu.edu (S. Rangan).

[1] Pi-Radio is a spin-off from the New York State Center for Advanced Technologies in Telecommunications (CATT) located at the Tandon School of Engineering at New York University.

https://doi.org/10.1016/j.comnet.2021.108220
Received 13 November 2020; Received in revised form 22 April 2021; Accepted 31 May 2021
Available online 6 June 2021

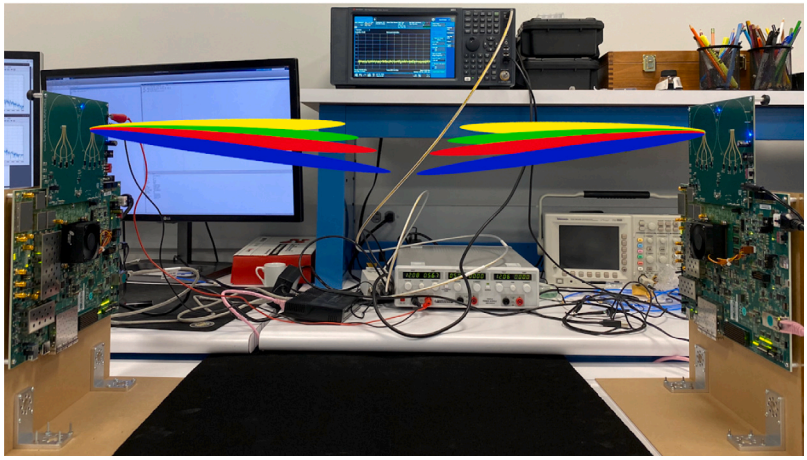

Fig. 1. A pair of Pi-Radio v1 SDR nodes under test at NYU. Critically, this fully-digital beamformer enables concurrent transmission and reception of signals in different directions.

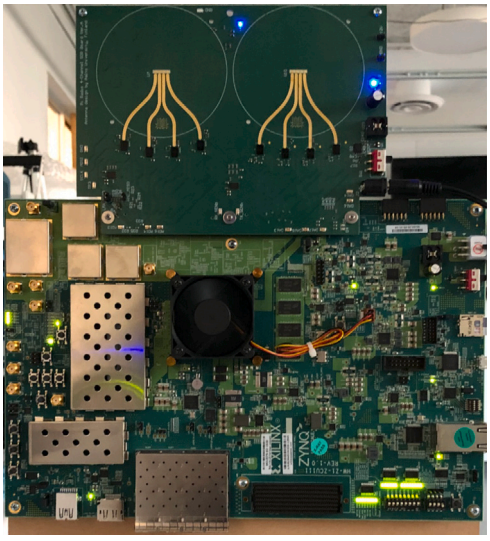

Fig. 2. The Pi-Radio v1 SDR being tested at NYU. The bottom board is the Xilinx RFSoC-based ZCU111 evaluation board; the board at the top is the Pi-Radio transceiver board. The RF shield has been removed to show the transceiver board clearly.

corrections; **(d)** LO phase offset corrections; **(e)** magnitude corrections; **(f)** IQ gain imbalance corrections; **(g)** IQ quadrature LO phase imbalance corrections; and **(h)** power-on self calibration of polarities. We conclude this paper with a demonstration of TX/RX beamforming through geometrically determined beamforming weights, thereby demonstrating that the SDR nodes have been calibrated correctly.

2. Hardware description

The SDR consists of three main parts: (a) the Xilinx RFSoC-based ZCU111 baseband board; (b) the Pi-Radio 4-channel fully-digital transceiver board; and (c) a host computer running various software tool chains (see Fig. 1 and Fig. 2).

Xilinx ZCU111 FPGA Board: The beating heart of this board is the powerful Zynq UltraScale+ XCZU28DR RFSoC FPGA chip that features 930k logic cells, 60.5 Mb of block memory, over 4k DSP slices, several high-speed gigabit transceiver (GT) ports, and over 300 general purpose I/O pins. Critically, this RFSoC also features eight 14-bit high-speed DACs and eight 12-bit high-speed ADCs. We clock the DACs and ADCs at 3932.16 MSps, leading to a maximum theoretical baseband bandwidth of nearly 4 GHz (since the I and Q components are treated independently). The RFSoC also features four ARM Cortex-A53 cores and two ARM Cortex-R5 cores; these powerful ARM cores can be used to run Linux, various applications, and also perform real-time operations. The ZCU111 board also features 4GB of DDR4 memory and supports 100 Gbps optical/copper connections to a host computer or additional FPGA co-processors. Importantly, this FPGA contains eight soft-decision forward error correction (SD-FEC) cores in silicon; these can be used to implement Turbo, Viterbi, LDPC, or Polar decoders (highly computationally intensive operations) without using any resources on the actual FPGA.

Pi-Radio v1 Transceiver Kit: This kit consists of a single board that implements a fully-digital 4-channel mmWave transceiver. On the TX side, it receives four I/Q analog baseband pairs from the eight DACs on the RFSoC, and feeds them to a bank of four Analog Devices (ADI) HMC6300 mmWave up-converters. The resulting mmWave signals are fed to a 1×4 patch antenna array using co-planar waveguide transmission lines, both designed by Aalto University (Finland). The RX side is symmetrical, with the ADI HMC6301 mmWave down-converters used to convert the four incoming RF signals to baseband, which are then fed into the eight ADCs of the ZCU111 board. While the baseband bandwidth (supported by the ZCU111) is 4 GHz, the mmWave HMC chips support 2 GHz of bandwidth. The operating frequency range is 57–64 GHz. All eight channels (four TX and four RX) are phase synchronized by a network comprising of: **1:** LO generation using the Texas Instruments LMX2595; **2:** dual-stage LO amplification using the ADI HMC962 and HMC441 parts; and **3:** LO distribution using Wilkinson dividers by Knowles Dielectric Labs. The boards are fabricated and assembled (including fully automated pick-and-place) by Sierra Circuits in Sunnyvale, CA. The boards also have two large circular *keep-out* areas around the antenna arrays, which users can use to mount their own passive dielectric lenses, if needed.

Host Computer: The ZCU111 board connects to a host computer using a gigabit ethernet interface and a simple TCP/IP control/data interface. MATLAB (or any other TCP/IP capable software like GNU Radio) can be run on the host computer to control the SDR operation. We have already implemented MATLAB-based drivers for this system. On the TX side, this allows the per-channel waveforms to be created in MATLAB, and shipped over to the RFSoC to be transmitted in a loop, until configured otherwise. On the RX side, the MATLAB code *triggers* the RFSoC to capture a set of samples from all ADCs synchronously, and then ship them over to MATLAB for further processing. While this method does not involve real-time processing on the FPGA, it was ideal to design, implement, and iterate through the calibration procedures quickly.

3. Calibration procedures

It is well known that mmWave systems rely on *beamforming* to close the link budgets. The very process of beamforming requires the TX and RX array elements to transmit and receive with known amplitudes and phases relative to each other. Achieving this deterministic behavior requires several types of calibration, which we examine in this section. Because a primary goal was to perform this calibration without needing lab bench equipment like mmWave spectrum analyzers and signal generators that can cost several hundreds of thousands of dollars, the techniques below perform *two-node calibration*. This involves placing two SDR nodes in boresight (i.e., facing each other). One is the node under calibration (NUC) and the other is the reference node (REF).

3.1. Frequency offset calibration

Many of the calibration techniques in this paper involve performing frequency domain correlation. For this to work properly, the local oscillators (LOs) on the NUC and REF need to be very close together, without large frequency offsets. The LO on the Pi-Radio SDR board is generated by the TI LMX2595 PLL chip that takes in a reference crystal oscillator input. We use a nominal 156.25 MHz crystal on our boards, but practical crystals always vary from their nominal values. Even small variations in the crystal frequency can lead to large variations in the final RF frequency. We therefore measured the frequency of the crystal on each board independently by *probing* the crystal output pads, and connecting the probe to a low-cost spectrum analyzer (Tektronix RSA306 that can measure up to 6 GHz). While calibrating two particular nodes, we observed that their crystal frequencies were 156.249725 MHz and 156.246375 MHz. This difference can lead to the mmWave RF center frequencies on the two nodes to differ by as much as 650kHz. Given that many calibration procedures utilize FFT bins as narrow as 500kHz, these frequency offsets need to be fixed.

Fortunately, fixing these offsets is simple. Once the crystal output frequencies are accurately measured, we simply recalculate the LMX2595 PLL registers (fractional and integer divider values) with the measured frequencies, instead of the nominal frequencies.

To test correct performance, we modulated a single tone at the TX side of the NUC, and measured the spectrum at the RX side of the REF, and observed that the correct bin on the RX side was populated. We then recalculated the LMX2595 registers for crystal frequencies that very slightly deviated from the measured frequencies, and observed significant leakage into neighboring frequency bins. This showed that the LO frequencies were calibrated correctly, and we could proceed to the next step.

3.2. Linearity measurements

The HMC6300 mmWave up-converter has an output P1 dB of about 12 dBm. Considering nominal patch antenna gains of 3 dB each on the TX and RX side, as well as a nominal 1m spacing between the nodes, the received power is calculated to be about −50 dBm. The input P1 dB point on the HMC6301 mmWave down-converter is rated at −19 dBm. We therefore have no concerns about the linearity on the RX side, but we need to measure the linearity on the TX side to ensure correct operation and prevent saturation.

To do this, we ran a simple experiment. The TX NUC transmitted a single tone baseband signal from TX channel $txIndex$. We varied the transmitted power (by digitally scaling the TX waveform), and measured the receive power on one channel of the RX REF. The results are shown in Fig. 3. The X axis is simply the RMS value of the digital TX waveform, expressed in dB. The Y axis is the received power on the RX REF, at the tone of interest. This shows that as long as we keep that RMS TX magnitude (digital) below approximately 74 dB, we will operate in the linear range. These results are consistent across different TX channels. If the TX waveform is below 40 dB, noise begins to dominate.

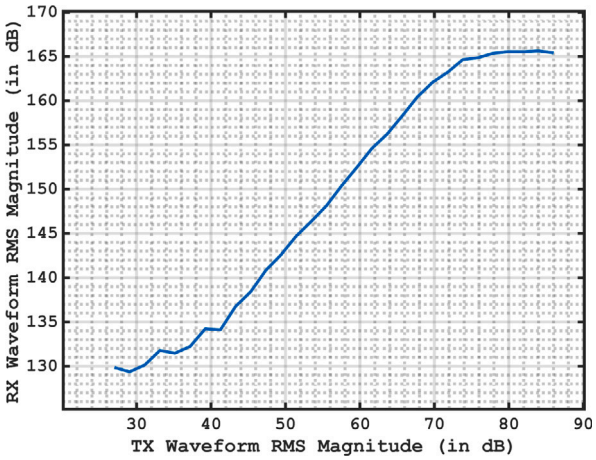

Fig. 3. [14] Measuring the Linear Operating Range of the TX: Keeping the TX waveform RMS magnitude below 165 dB ensures linear operation. Below 135 dB, the system becomes noise limited.

3.3. Timing and phase offset calibration

What causes the problem? The baseband IQ transmission lines (TLines) from the RFSoC connector to the HMC6300/6301 chips might not always be length matched. For a given channel, the IQ differential lines were designed to be length matched; however, the lengths of the TLines for different channels are different in the board layout. This is often the case because length-matching TLines can lead to additional losses and routing complexity. Further, the length of the TLines between the RFICs (the HMC6300 and HMC6301) and the antenna elements are also not length matched because the additional losses in the mmWave bands caused by increasing TLine length can be very significant. These length mismatches cause *per-channel timing offsets* that need to be calibrated out.

On the Pi-Radio transceiver board, the eight channels (4 TX and 4 RX) are phase synchronized using a Wilkinson tree that distributes the local oscillator (LO) signal. The TLines in this tree are also of different lengths, in order to make routing feasible and to minimize board losses. This leads to *per-channel phase offsets* that also need to be calibrated out, along with the *per-channel timing offsets*.

How do we calibrate? We now describe the method to calibrate the TX array on the NUC. The RX array on REF uses just one active channel; the other channels can either be turned off or used for redundancy. When two nodes are placed in boresight, the TX array on the NUC will be directly in front of the RX array on REF, with about two feet of spacing between them to ensure far-field operation.

The first step is to estimate the fractional timing offset. As described in Algorithm 1 and illustrated in Fig. 4, the TX NUC transmits four orthogonal wideband sounding sequences, one per channel. These sequences are randomly generated QPSK signals, known to have excellent correlation properties. The RX REF passes the received waveform through a fractional delay filter (for several fractional timing offset hypotheses), and correlates this against each of the four sounding sequences; this process determines which fractional timing offset leads to the largest peak in the power delay profile for each of the four TX channels. This process is repeated over several iterations, and the results are *smartly* averaged (as explained later) to estimate the fractional timing offsets for each channel. The intuition behind this process is that for incorrect fractional timing offset corrections, the power in the PDP peak gets *spread out* over adjacent peaks; but for the correct fractional timing offset correction, the power is *concentrated* in just one peak.

The correctness of this method is verified by repeating the experiment, with the difference that the TX NUC pre-compensates the

sounding sequence by fractionally delaying it by the previously determined fractional timing offset. This time, we observe that the optimal fractional timing offset determined by the RX REF is very close to 0 for all channels, thereby demonstrating that the correction is indeed proper.

The next step is to estimate the *per-channel phase offsets*. Repeat the experiment above, where the transmitted waveforms are pre-compensated by the optimal fractional timing offsets for each TX channel on the NUC. On the RX REF, correlate the received waveform against each of the four sounding sequences, and observe the phase of the maximum peak in the resulting PDPs. Suppose the phases are β_n, for $n \in \{1,2,3,4\}$. The phase correction factors are therefore $\gamma_n = (\beta_n - \beta_1)$ for $n \in \{1,2,3,4\}$. To verify correct detection of the phase offsets, further pre-compensate the transmitted sounding sequences on the TX NUC by de-rotating each sounding sequence n by $exp(j\gamma_n)$; this time, we observe that the phases of all the received peaks are equal, thereby demonstrating proper per-channel phase offset correction.

Algorithm 1: Measure Fractional Timing Offsets (One Iteration)

Result: *best_to* contains the best fractional timing offset estimates for each TX channel on NUC
Initialize orthogonal time-domain sounding sequences *tx_td* for all TX channels;
Initialize ($n = 100$) fractional timing offset hypotheses *tos* uniformly spaced in $[-0.50, 0.49]$;
Initialize *best_to[4] = {0,0,0,0}*;
Continuously and simultaneously **transmit** *tx_td* for all TX channels on NUC;
Capture samples *rx_td* from one RX channel on REF;
for *txIndex = 1:4* **do**
  Initialize *max_peak = 0*;
  for *to* ∈ *tos* **do**
    Pass *rx_td* through a fractional delay filter of delay *to*;
    Correlate against *tx_td[txIndex]*;
    if *peak* ≥ *max_peak* **then**
      max_peak = peak;
      best_to[txIndex] = to;
    end
  end
  best_to[txIndex] = best_to[txIndex] - best_to[1];
  Note: This is in the range of [-1, +1];
end

A Note on Smart Averaging: While searching through all possible fractional timing offsets, the experiment is run over several iterations, and the results are averaged; this helps to get clean and stable calibration factors. However, we cannot simply average the fractional timing offsets across iterations, because we might get -0.5 as the *best_to* in one iteration, but $+0.5$ in the next, leading to an incorrect average of 0, even though the two offsets are equivalent. To overcome this issue, we use a simple and well-known trick: **(a)** Multiply *best_to* by 2π; **(b)** Create a complex number with the resulting phase; **(c)** Add these complex numbers across all iterations; and **(d)** take the phase of the summed up complex number. This leads us to another realization: the algorithm so far has been unable to distinguish between positive and negative fractional timing offsets (for example, -0.3 and $+0.7$). Therefore, we need a final step of integer timing offset correction to be performed.

The experiment is repeated, with the TX NUC pre-compensating the sounding sequence by the estimated per-channel fractional timing offset and per-channel phase offsets. The RX REF captures the received waveform, and checks the location (in time samples) of all four received PDP peaks (one per TX channel). It is immediately visible that some peaks are earlier than others, thereby yielding the integer timing offsets that need to be augmented to the existing fractional timing offsets.

As a final test, the experiment is repeated with the TX NUC pre-compensating the per-channel sounding sequence by the aggregate

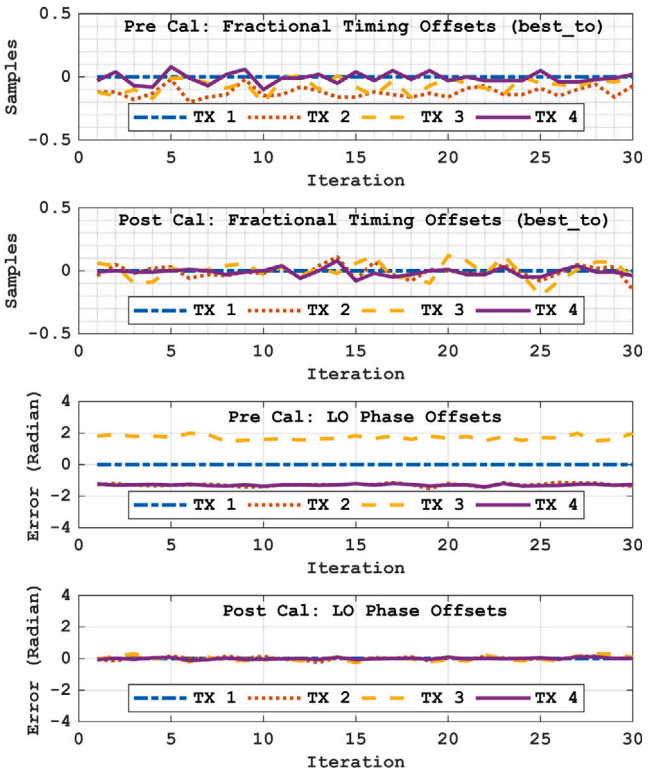

Fig. 4. [14] Calibrating the fractional timing offsets and LO phase offsets on the TX array. The integer timing offsets are 0, and have hence not been plotted. Calibrating the RX array is analogous.

timing offsets (fractional and integer) and the per-channel phase offsets. At the RX REF, we can observe that all peaks appear in the same time sample and have the same phase. This is verified over several iterations.

We have just described the procedure to calibrate the *per-channel timing offset* and *per-channel phase offset* for the TX array. Calibration of the RX array proceeds in the same way and is provided in our GitHub repository [16].

What happens if the nodes are not placed perfectly in boresight? Using simple trigonometric calculations, we have shown that even if the nodes are slightly misaligned, this has a negligible impact on the fidelity of the calibration procedures. For example, consider the RX-side calibration, wherein a single TX antenna on the REF is transmitting, and this signal is received by all antennas on the NUC. Suppose the distances from the TX antenna to each of the RX antennas is $\{d_1, d_2, d_3, d_4\}$. We have calculated that even with node misalignment of several degrees (easily detectable by the naked eye), the difference between any d_i and d_j, for any two RX channels i and j, is extremely small. In fact, this difference $(d = d_i - d_j)$ is so much smaller than λ, that it leads to phase errors of only a few degrees in the worst case. This has a negligible impact on the beams, and is therefore not a problem for our techniques.

3.4. Magnitude correction

What causes the problem? Multiple copies of the same module (like the HMC6300) will have manufacturing and packaging variations that lead to the conversion gain showing variances. There are further variances caused by temperature gradients on the transceiver board. Warmer modules will show lower gain than the cooler modules during steady state operation. Correct beamforming operation relies on deterministic gains or signal powers across these modules; these therefore have to be measured and calibrated out.

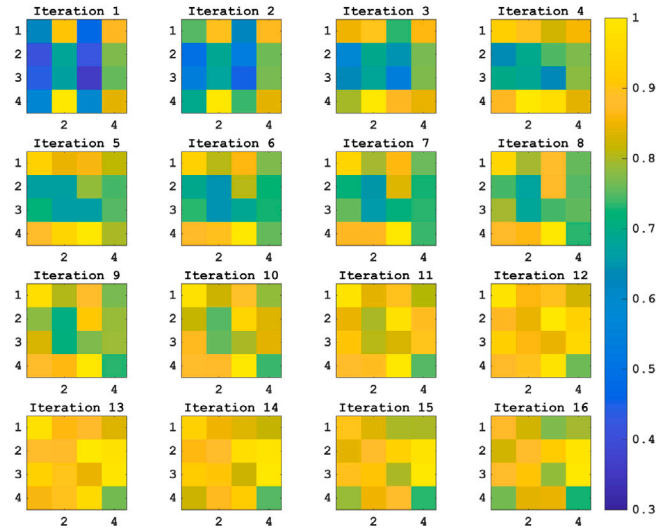

Fig. 5. [14] Calibrating the TX and RX magnitudes (digital gains). The rows and columns correspond to TX and RX indices. Magnitudes are normalized by the (TX, RX) pair with maximum power within that iteration.

How do we calibrate? We describe the magnitude calibration of the TX array on NUC; the RX calibration is symmetrical. This simple procedure is shown in Algorithm 2. Transmit a wideband signal from each TX channel $txIndex$, one channel at a time (i.e., not simultaneously). Measure the receive power on REF, and divide it by the received power when $txIndex = 1$ was transmitting. Take the average of these correction factors across all RX channels on REF. The result is the magnitude correction factor for each TX channel on NUC. Run this experiment over several iterations (in our experience, 3–4 iterations are sufficient), and take the average of the results for the sake of stability and convergence. The results from magnitude calibration are shown in Fig. 5. Observe that within a few iterations, the TX and RX magnitude correction factors have stabilized to steady state values.

Algorithm 2: Measure Magnitude Calibration Factors (One Iteration)

Result: *tx_mag_correction* contains the best magnitude correction estimates for each TX channel on NUC
Initialize orthogonal a wideband time-domain signal *tx_td[1:N]* (1-D vector of length N);
Initialize *mags*, a 4x4 matrix of zeros. Rows correspond to TX channels on NUC; columns correspond to RX elements on REF;
for *txIndex = 1:4* **do**
  Transmit continually *tx_td[1:N]* on TX channel *txIndex* of NUC. Other TX channels are silent;
  Receive *rx_td[1:4][1:N]*, which is 4 vectors each of length N (one vector per RX channel on REF);
  for *rxIndex = 1:4* **do**
    Measure power within the frequency band of interest in *rx_td[rxIndex][1:N]*, and save in *mags[txIndex][rxIndex]*;
  end
  $m\_v = mags[txIndex][1:4]./mags[1][1:4]$;
  Note: Pairwise division in MATLAB syntax;
  $tx\_mag\_correction[txIndex] = mean(m\_v)$;
end

3.5. IQ calibration on the receiver

What causes the problem? Consider a receiver that down-converts from RF to baseband. Without loss of generality, assume that the RF signal is $cos(2\pi(f_c+f_m)t$, where f_c is the carrier frequency and f_m is the transmitted baseband signal frequency. An ideal receiver produces the complex baseband signal by mixing the received signal with a complex carrier at frequency f_c, and passing through a low pass filter (LPF). This can be written as:

$$i(t) = LPF(cos(2\pi f_c t)cos(2\pi(fc + f_m)t)) \quad (1)$$

$$q(t) = LPF(sin(2\pi f_c t)cos(2\pi(fc + f_m)t)) \quad (2)$$

This yields $i(t) = sin(2\pi f_m t)$ and $q(t) = cos(2\pi f_m t)$, as desired. However, a practical receiver will have the following imperfections: **(a)** the conversion gain on the I channel and Q channel can differ by a factor of $\alpha \neq 1$; and **(b)** the quadrature carrier might not be exactly $\pi/2$ radians out of phase with the in-phase carrier, the phase error being $\upsilon > 0$ radians. Using standard trigonometric identities, the practically demodulated signals reduce to:

$$i'(t) = \alpha cos(2\pi f_m t) \quad (3)$$

$$q'(t) = sin(2\pi f_m t + \upsilon) \quad (4)$$

Note that the magnitude factor α and phase factor υ were arbitrarily assigned to the i and q channels respectively, without any loss of generality. These imbalances lead to poor suppression of undesired sidebands, thereby driving up the error vector magnitudes (EVMs) in the receiver. Expanding equation (4) and writing in matrix notation, we get:

$$\begin{bmatrix} i'(t) \\ q'(t) \end{bmatrix} = \begin{bmatrix} \alpha & 0 \\ sin(\upsilon) & cos(\upsilon) \end{bmatrix} \begin{bmatrix} i(t) \\ q(t) \end{bmatrix} \quad (5)$$

Call this middle matrix M; the values of α and υ determine how imbalanced the receiver is, and therefore the magnitude of the undesired sidebands. Overcoming these imbalances involves estimating and inverting this matrix M to recover the desired baseband signal.

Generating clean sinusoids: Some IQ calibration procedures rely on being able to generate clean sinusoids (at mmWave frequencies) without sidebands or spurs; but how do we generate such signals? Lab bench synthesizers that can generate mmWave signals cost several hundreds of thousands of dollars. Another possibility is to configure REF (the SDR other than the node under calibration, or NUC) at center frequency f_c and to transmit a modulated tone at f_m to generate a sinusoid at $f_c + f_m$. This, however, will lead to a situation in which the transmitter-side IQ imbalances perturb the signal, with the result that the calibration technique will be unable to distinguish between TX-side and RX-side imbalances. Our work-around is to use *Offset LO*.

As an example, say we are calibrating the RX IQ imbalances on the NUC at center frequency 58 GHz, and suppose we want f_m = -1 GHz. To generate this 57 GHz signal, we will configure REF at f_c = 56 GHz, and modulate a baseband signal at frequency +1 GHz. The transmitted signal will therefore have a desired sideband at 57 GHz, and a smaller undesired sideband at 55 GHz. At the receiver NUC, f_c = 58 GHz. The undesired sideband is 3 GHz down from center, and is therefore filtered away in the IF stage of down-conversion. Using this simple Offset LO technique, we can generate the required clean tones as seen by the receiver NUC.

How do we calibrate? Our technique is inspired by Ellingson's IQ calibration procedure [17]. Let us start with the α calibration on the RX channels of the NUC, shown in Algorithm 3. The NUC is configured at center frequency 58 GHz, and is expecting to receive a clean sinusoid at 57 GHz, which is generated by the REF TX as explained in the previous paragraph. Capture the I and Q baseband waveforms on the NUC, and measure the power on them. The result is that α is the integrated energy on the I channel divided by the integrated energy on the Q channel. This measurement is repeated over several iterations and averaged. There is a further improvement, in which the time-domain signals are passed through an FFT, and only the energy corresponding to the frequency bin of interest (-1 GHz) is used in averaging. The two methods provide results that are virtually indistinguishable. Once the α term is known, calibration simply involves dividing the real component

Algorithm 3: Measure RX-side IQ α imbalance (One Iteration)

Result: *rx_iq_alpha* contains the best RX IQ α estimates for each RX channel on NUC. This is a vector of size 4.
Configure f_c on the NUC to 58 GHz;
Configure f_c on the REF to 56 GHz;
Initialize *tx_td*, a row vector containing time domain samples for a single tone sinusoid at 1 GHz. There are N samples in this waveform;
Transmit continually *tx_td* from one TX channel on the REF. Other TX channels are silent;
Receive *rx_td[1:4][1:N]*, which is 4 complex vectors each of length N (one vector per RX channel on the NUC);
for *rxIndex = 1:4* **do**
  Measure E_i as the RMS energy in the I channel of $rx\_td[rxIndex][1:N]$;
  Measure E_q as the RMS energy in the Q channel of $rx\_td[rxIndex][1:N]$;
  Calculate $rx\_iq\_alpha[rxIndex] = \frac{E_i}{E_q}$;
end

of the received time-domain waveform by α, and leaving the imaginary component untouched.

Once the α imbalance is corrected, Eq. (5) reduces to:

$$\begin{bmatrix} i'(t) \\ q'(t) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ sin(\upsilon) & cos(\upsilon) \end{bmatrix} \begin{bmatrix} i(t) \\ q(t) \end{bmatrix} \tag{6}$$

The next step is to calibrate out the quadrature phase imbalances υ. The phase imbalance matrix in the equation above needs to be inverted to recover I and Q from I' and Q' as shown:

$$\begin{bmatrix} i(t) \\ q(t) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -tan(\upsilon) & sec(\upsilon) \end{bmatrix} \begin{bmatrix} i'(t) \\ q'(t) \end{bmatrix} \tag{7}$$

This task boils down to estimating the value of υ, applying equation (7), and we are done. But how do we estimate υ? The idea is to take the received waveform, and apply corrections for a large number of υ hypotheses in the range of [-1 1] radians. For each hypothesis, measure the sideband suppression (i.e., the received baseband power at -1 GHz divided by the received baseband power at $+1$ GHz). The υ hypothesis that leads to the greatest suppression is the correct estimate, as shown in Algorithm 4. Average over a number of iterations for good performance and stability. The results from one such iteration is shown in Fig. 6. This graph plots the resulting sideband suppression as a function of different phase correction factors that have been applied. One can see that the I and Q LO paths in the receiver are not $\pi/2$ out of phase, but instead have an additional phase error of 0.2 radians. Once this phase correction is applied (Eq. (7)), the sideband suppression improves from 19.91 dB to 31.17 dB.

We have just described how to estimate and correct the RX-side IQ imbalances: the magnitude correction factor α and the phase correction factor υ. To finally show the device's overall performance, we run an experiment that does the following: **(a)** capture samples at the RX, plot the spectrum, and show the uncalibrated sideband suppression; **(b)** apply the α correction to the same signals; and **(c)** apply the υ correction to the same signals. Fig. 7 shows the sideband suppression getting better as the α and υ corrections are applied. The columns correspond to RX channel 1, 2, and 3 (4 is not shown for space reasons). Despite the HMC6301 (mmWave receiver) being a heterodyne down-converter (as opposed to a direct converter), we observed that the typical υ correction factors were rather high, in the range of -0.3 to $+0.3$ radians. Fig. 7 shows that correctly performing RX-side IQ calibration results in additional sideband suppression in the 10–13 dB range.

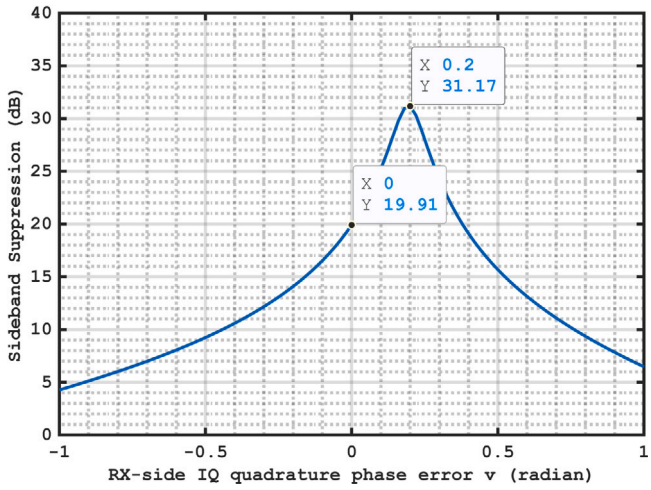

Fig. 6. [14] Estimating the quadrature phase offset on one RX channel on the NUC (one iteration). Observe that after υ correction, the sideband suppression is improved by over 11 dB.

Algorithm 4: Measure RX-side IQ υ imbalance (One Iteration)

Result: *rx_iq_v* contains the best RX IQ υ estimates for each RX channel on NUC. This is a vector of size 4.
Configure f_c on the NUC to 58 GHz;
Configure f_c on the REF to 56 GHz;
Initialize *tx_td*, a row vector containing time domain samples for a single tone sinusoid at $f_m = +1$ GHz. There are N samples in this waveform;
Initialize *vs*, a vector containing $n = 100$ hypotheses of υ, uniformly spaced in [-1 1] radians;
Transmit continually *tx_td* from one TX channel on the REF. Other TX channels are silent;
Receive *rx_td[1:4][1:N]*, which is 4 complex vectors each of length N (one vector per RX channel on the NUC);
for *rxIndex = 1:4* **do**
  Apply the α correction to $rx\_td[rxIndex][1:N]$;
  Initialize $s_{max} = 0$;
  for $\upsilon \in vs$ **do**
    Apply υ to $rx\_td[rxIndex][1:N]$ using equation (7);
    Calculate $rx\_fd[rxIndex][1:N]$ by passing the time-domain samples through an FFT;
    Measure E_{LSB} as the energy in the desired (lower) sideband of $rx\_fd[rxIndex][1:N]$ (corresponding to $f_m = -1$ GHz);
    Measure E_{USB} as the energy in the undesired (upper) sideband of $rx\_fd[rxIndex][1:N]$ (corresponding to $f_m = +1$ GHz);
    Calculate $s = \frac{E_{LSB}}{E_{USB}}$. Express in dB;
    if $s \geq s_{max}$ **then**
      $s_{max} = s$;
      $rx\_iq\_v[rxIndex] = \upsilon$;
    end
  end
end

3.6. IQ calibration at the transmitter

What causes the problem? An up-converting transmitter takes the complex baseband signal (say it is a single tone $e^{j2\pi f_m t}$) and mixes it with a complex carrier $e^{j2\pi f_c t}$. The real component of this result is then amplified and transmitted. However, the quadrature component of the carrier might not be phased at perfectly $\pi/2$ away from the in-phase

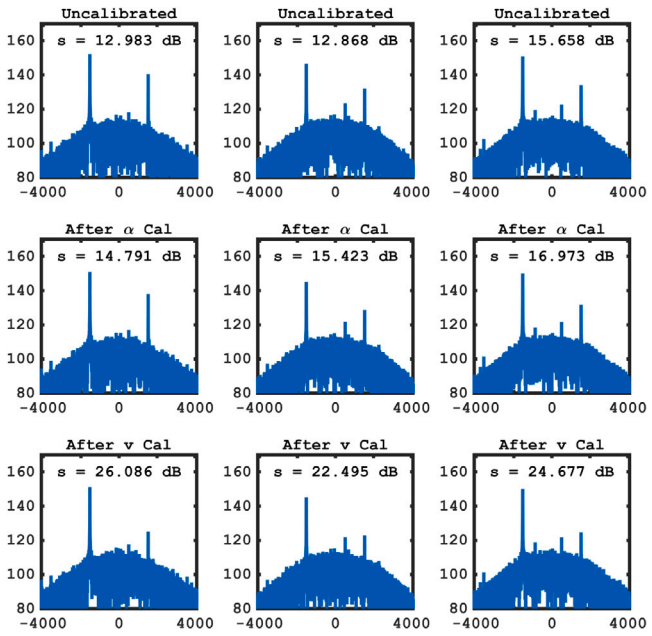

Fig. 7. [14] Applying the RX IQ calibration factors results in *additional* suppression of the unwanted sideband by 10–13 dB (the total suppression is denoted by s). The columns represent RX channel 1, 2, and 3 (the figure would become too small if channel 4 were also plotted). The X-axis is the subcarrier index (frequency) and the Y-axis is the power in dB. This figure plots just one iteration.

component of the carrier; this might have a phase error of υ radians. Further, the conversion gains on the I and Q channels might differ by a factor of α. If the baseband signal is given by $i(t) = cos(2\pi f_m t)$ and $q(t) = sin(2\pi f_m t)$, the baseband equivalent of the transmitted signal in a practical transmitter is given by: $i'(t) = \alpha cos(2\pi f_m t)$ and $q'(t) = sin(2\pi f_m t + \upsilon)$; we have arbitrarily assigned the α and υ factors to the I and Q channels respectively, without any loss of generality. Using standard trigonometric identities, this can be written as:

$$\begin{bmatrix} i'(t) \\ q'(t) \end{bmatrix} = \begin{bmatrix} \alpha & 0 \\ sin(\upsilon) & cos(\upsilon) \end{bmatrix} \begin{bmatrix} i(t) \\ q(t) \end{bmatrix} \tag{8}$$

How do we calibrate? The first step is to measure and calibrate out the α factor, as shown in Algorithm 5. Activate one TX channel at a time on the NUC, and transmit only the real component of a single tone baseband signal; measure the received power on the RX REF. Next, transmit only the imaginary component of the baseband signal, and measure the received power on the RX REF. The ratio of these two received power values, when averaged over all RX channels on REF, is the estimate of α. Average over multiple iterations to get stable calibration factors. Once the α calibration factors are known, applying them simply involves dividing the real component of the transmitted baseband waveform by α and leaving the imaginary component untouched. The next step is measuring and calibrating the υ phase offsets in the TX chains.

After the α correction factors have been applied, Eq. (8) reduces to:

$$\begin{bmatrix} i'(t) \\ q'(t) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ sin(\upsilon) & cos(\upsilon) \end{bmatrix} \begin{bmatrix} i(t) \\ q(t) \end{bmatrix} \tag{9}$$

Overcoming the υ imbalances involves *precoding* the transmitted waveform by the inverse of the central matrix in Eq. (9). This inverse is given by:

$$M^{-1} = \begin{bmatrix} 1 & 0 \\ -tan(\upsilon) & sec(\upsilon) \end{bmatrix} \tag{10}$$

What we have left to do is to estimate υ for each of the four channels on the TX NUC. Let us say the TX NUC is operating at $f_c = 58$ GHz. The

Algorithm 5: Measure TX-side IQ α imbalance (One Iteration)

Result: *tx_iq_alpha* contains the best TX IQ α estimates for each TX channel on NUC. This is a vector of size 4.
Configure f_c on the NUC and REF to be the same (say, 58 GHz);
Initialize *tx_td*, a row vector containing time domain samples for a single tone baseband signal. There are N complex samples in this waveform;
for *txIndex = 1:4* **do**
  // Measure the I channel on txIndex;
  Transmit continually *real(tx_td)* on the I component of TX channel $txIndex$. The Q component of $txIndex$, as well as all other TX channels $\neq txIndex$ are silent;
  Receive *rx_td[1:4][1:N]*, which is 4 complex vectors each of length N (one vector per RX channel on the REF);
  for *rxIndex = 1:4* **do**
    Measure $E_i[rxIndex]$ as the energy at the tone of interest within the received signal $rx\_td[rxIndex][1:N]$;
  end
  // Measure the Q channel on txIndex;
  Transmit continually *imag(tx_td)* on the Q component of TX channel $txIndex$. The I component of $txIndex$, as well as all other TX channels $\neq txIndex$ are silent;
  Receive *rx_td[1:4][1:N]*, which is 4 complex vectors each of length N (one vector per RX channel on the REF);
  for *rxIndex = 1:4* **do**
    Measure $E_q[rxIndex]$ as the energy at the tone of interest within the received signal $rx\_td[rxIndex][1:N]$;
  end
  // Final Calculations;
  Calculate $tx\_iq\_alpha[txIndex]$ as $mean(E_i[1:4] \,./\, E_q[1:4])$;
  Note: Pairwise division in MATLAB syntax;
end

intuition behind the technique is the following: **(a)** transmit a single tone (at $f_m = +1$ GHz) from the TX NUC, leading to the desired (upper) sideband at 59 GHz, and the undesired (lower) sideband at 57 GHz; **(b)** the RX REF is operating at an offset LO of $f_c = 56$ GHz, and is capable of measuring the power in only the undesired lower sideband at 57 GHz; **(c)** sequentially cycle through and apply the precoding matrices for every υ hypothesis on the TX NUC as per Eq. (10); **(d)** use the RX REF to determine which υ precoding led to the lowest undesired sideband; and **(e)** since υ is the only variable, this must correspond to the *best* υ correction on the TX NUC. As a simple rule of thumb, the υ hypotheses should be spaced at most 0.1 radian apart, but ideally closer.

The *Offset LO* method ensures that the RX REF sees only the transmitted lower sideband at 57 GHz, but cannot see the transmitted upper sideband at 59 GHz (this gets filtered out in the IF stage). The procedure above is repeated for every $txIndex \in [1, 2, 3, 4]$. In each run, the power in the undesired lower sideband is measured by REF across all υ hypotheses and all RX channels on REF. The optimal υ value is estimated using an MMSE error metric across all υ hypotheses and all 4 RX channels on REF. These υ estimates should be averaged over a few iterations, so as to arrive at stable calibration factors. One iteration of this procedure is formalized in Algorithm 6.

To examine the TX-side IQ υ calibration process, we ran a simplified experiment where only one RX channel on REF was used (otherwise the data is too much to plot in this paper). As shown in Algorithm 6, each TX channel sequentially cycles through multiple υ hypotheses in the range of $[-1, 1]$ radians; in each case, the RX REF measures the power in the undesired lower sideband at 57 GHz. This has been plotted in Fig. 8; the four graphs correspond to different TX channels $txIndex$ on the NUC. Consider $txIndex = 1$; the optimal υ correction

Algorithm 6: Measure TX-side IQ v imbalance (One Iteration)

Result: $tx\_iq\_v[1:4]$ contains the best TX IQ v estimates for each TX channel on NUC.
Configure f_c on the NUC to 58 GHz;
Configure f_c on the REF to 56 GHz;
Initialize tx_td, a row vector containing time domain samples for a single tone baseband signal at $f_m = +1$ GHz. There are N complex samples in this waveform;
Initialize vs, an $n = 100$ sized vector containing the v hypotheses, uniform in $[-1, 1]$;
for *txIndex = 1:4* **do**
  // Make all the required measurements;
  for $v \in vs$ **do**
    Precode tx_td with the matrix from equation (10), using the current v value;
    Transmit the waveform from TX channel $txIndex$ on the NUC. All other TX channels on NUC are silent;
    Receive *rx_td[1:4][1:N]*, which is 4 complex vectors each of length N (one vector per RX channel on the REF);
    for *rxIndex = 1:4* **do**
      Measure $p[rxIndex][v]$ as the energy (in dB) in $rx\_td[rxIndex][1:N]$ at baseband frequency $f_m = +1$ GHz (corresponding to the undesired sideband being transmitted for the current v hypothesis);
    end
  end
  // Normalize the measurements in p (in dB);
  for *rxIndex = 1:4* **do**
    $m = min(p[rxIndex][:])$;
    for $v \in vs$ **do**
      $p[rxIndex][v] = p[rxIndex][v] - m$;
    end
  end
  // Estimate optimal v;
  for $v \in vs$ **do**
    $err[v] = 0$;
    for *rxIndex = 1:4* **do**
      $err[v] = err[v] + (p[rxIndex][v])^2$;
    end
  end
  // Which v has the lowest error?;
  $tx\_iq\_v[txIndex] = argmin(err[v], v)$;
end

is 0.46 radians, showing that the hardware imperfections can be *really large*. If this correction had not been applied, this undesired sideband would be larger by 8.567 dB. Channel $txIndex = 3$ has a small IQ v error of 0.12 radian, and the additional sideband suppression offered by v calibration is small (0.8663 dB). Channel $txIndex = 4$ has a moderate IQ v correction of 0.26 radian, but the benefit of applying the calibration is quite large (11.59 dB).

Why are the graphs in Fig. 8 noisy? The goal of the *Offset LO* method is to ensure that the RX REF sees only the transmitted lower sideband, with the transmitted upper sideband being filtered away. We therefore want the f_c at the TX NUC and the RX REF to be far apart. The tone corresponding to the transmitted lower sideband (57 GHz) is near the left edge of the transition band of the transmitter IF filter; similarly, this tone is also near the right edge of the transition band of the receiver IF filter. This observation, when combined with the fact that the lower sideband is not the desired sideband at the TX NUC, makes the signal very weak at the RX REF. These low-power signals make the measurements in Fig. 8 appear noisy.

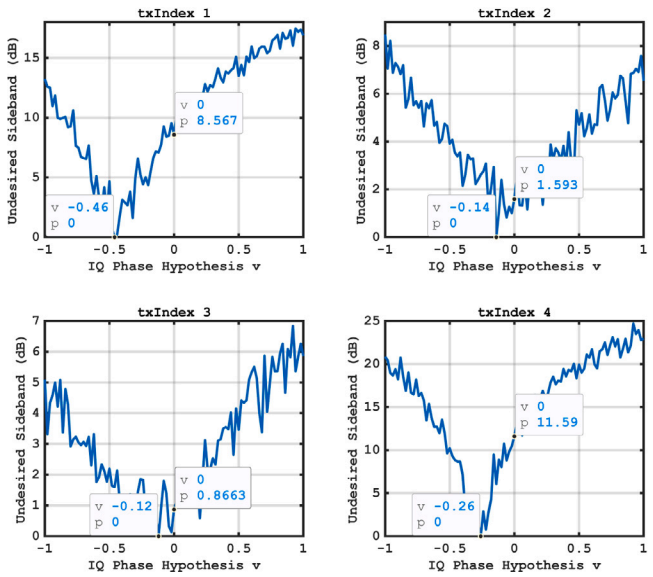

Fig. 8. [14] TX IQ calibration: Each TX cycles through and precodes using multiple v hypotheses; in each case, the power in the undesired (lower) sideband is measured by the RX REF. Observe that the v corrections can be quite large (as high as 0.46 radians), and correcting the v imbalance can yield additional sideband suppression of up to 11dB. The data marker v is the IQ phase hypothesis, and p refers to the power in the undesired sideband (same p as in Algorithm 6).

3.7. Summary of IQ calibration

Despite the fact that both the HMC6300 mmWave up-converter and the HMC6301 mmWave down-converter have heterodyne architectures (as opposed to direct conversion), the IQ imbalances are quite significant. This can lead to large unwanted sidebands, thereby lowering the EVM of the signals. We have observed that applying the IQ corrections can lead to additional suppression of up to 15 dB, thereby demonstrating the importance of performing these calibrations carefully.

The suite of TX and RX array calibrations (fractional + integer timing offsets, LO phase offsets, and magnitude calibration) takes about 30 min to run. This process calibrates both nodes, since they take turns behaving as the NUC and the REF. These calibrations take a long time to run because of the discontinuous nature of the MATLAB drivers; shipping a new waveform to the FPGA, or triggering and receiving the waveform in MATLAB takes about $1s$ to run. Given that the calibration process requires several hundred iterations, the process takes time. Of course, we can speed it up by running the calibration procedures directly on the FPGA. This is planned in future work.

The IQ calibrations take about an hour to run, the time being dominated by the sheer number of waveform writes and reads needed by the TX IQ v calibration. Once all the procedures are run, the calibration factors are written to a file, and these can be loaded upon startup. However, we realized that there are random starting LO phase variations across power cycles; re-running the calibration routines upon every power cycle is infeasible. We therefore designed and implemented a self-calibration routine that can be run every power cycle; this takes about one minute to run, and is described next.

4. Self calibration upon power up

The HMC6300 and HMC6301 chips on the Pi-Radio transceiver board are kept phase-locked, thanks to the network that generates, amplifies, and distributes this LO signal (which is 1/3.5 the desired RF center frequency f_c) to all TX and RX channels. Recall that the HMC chips have two LO multipliers on chip, which are used for: (a)

a 0.5x multiplier for the conversion between baseband and IF; and (b) a 3x multiplier for the conversion between IF and RF. This leads to the RF center frequency being 3.5× that of the input LO frequency. Upon power cycling, the *starting phase* of f_c generated on an arbitrary subset of channels could be shifted by π radians with respect to the other channels on the board. We observed this process to be seemingly random, and we could not figure out any digital controls to the HMC chips to achieve deterministic phase across all chips. We therefore needed a method to: **(a)** measure the starting phases of each HMC chip in the system at the time of calibration; **(b)** run a simple test at every subsequent power cycle to see which TX and RX channels have flipped their starting phase as compared to the previous power cycle; and **(c)** perform the detection and correction of this phase/anti-phase issue (also known as polarities) within one minute. This is important because users do not want to wait too long to use an SDR, and they *definitely do not want to have to physically move two SDR nodes in boresight and re-run the long calibration every time the nodes are power cycled*.

After the SDR nodes have been calibrated using the two-node process (involving the NUC and the REF), we run an additional self-measurement phase (involving only one node). All the phases of the TX and RX channels are assumed to be correct, because that is assured through the two-node calibration. We then *measure the channel* between each TX channel and each RX channel on the same node; each of these 16 measurements contains the *system frequency response* for that (TX, RX) channel pair. Generating these *system frequency responses* is shown in Algorithm 7. This system frequency response implicitly contains embedded information that is unique to that SDR node, including the reflections from the RF shields and minor variations in part placement. This set of system frequency responses is written to a file; this contains all the information necessary to correct the polarities of each channel (i.e., the phase/anti-phase issue) upon power up.

Algorithm 7: Measure System Frequency Responses

Result: *self_interference_fd[1:4][1:4]* contains the system frequency response between each TX each RX channel.
Initialize a known wideband channel sounding sequence tx_td of length N in time domain;
for *txIndex=1:4* **do**
  Transmit tx_td from $txIndex$ continuously. All other TX channels are silent.;
  Trigger and Receive $rx\_td[1:4][1:N]$ from all RX channels simultaneously.;
  for *rxIndex=1:4* **do**
    Calculate *rx_fd = FFT(rx_td[rxIndex][:])*;
    Assign *self_interference_fd[txIndex][rxIndex] = rx_fd*;
  end
end
Write to file *self_interference_fd[1:4][1:4]*;

When the SDR is power cycled, each TX and RX chip might arbitrarily shift its starting phase by π radians. To correct this, we run the polarity detection and correction procedure from Algorithm 8. The intuition is as follows. Measure the *system frequency response* between each TX channel and each RX channel. For now, consider one TX channel $txIndex$ and one RX channel $rxIndex$. Correlate this newly measured system frequency response against the previously saved system frequency response from Algorithm 7. Observe the output of the correlator in the time domain; if the phase of the maximum amplitude peak is close to 0, it means that either: **(a)** neither $txIndex$ nor $rxIndex$ have flipped their polarities; or **(b)** both $txIndex$ and $rxIndex$ have flipped their polarities. Conversely, if the phase of the maximum amplitude peak is close to π, it means that exactly one of $txIndex$ and $rxIndex$ have flipped their polarities. Stage 1 in Algorithm 8 populates a 4×4 matrix *pol_error* with the rows and columns corresponding to TX and RX channels respectively. Each matrix element $pol\_error[txIndex][rxIndex]$ contains a 0 if none or both of $txIndex$ and $rxIndex$ have flipped, and 1 otherwise. This has been illustrated through a few examples shown:

Algorithm 8: Self Cal: Polarity Correction

Result: $tx\_pols[1:4]$ and $rx\_pols[1:4]$ contain the polarity correction factors.
Initialize the known wideband channel sounding sequence tx_td of length N in time domain;
Initialize $tx\_pols[1:4]$ and $rx\_pols[1:4]$ to all 1s;
Read from file the saved calibration factors *self_interference_fd[1:4][1:4]* from Algorithm 7;

while *Self-Calibration Not Complete* **do**
  // Stage 1: Populate $pol\_error[1:4][1:4]$;
  Initialize $ang[1:4][1:4]$ to all 0s;
  Initialize $pol\_error[1:4][1:4]$ to all 0s;
  for *txIndex=1:4* **do**
    Transmit tx_td (with polarity $tx\_pols[txIndex]$) from $txIndex$ continuously. All other TX channels are silent.;
    Trigger and Receive $rx\_td[1:4][1:N]$ from all RX channels simultaneously. Apply $rx\_pols[rxIndex]$ appropriately;
    for *rxIndex=1:4* **do**
      Calculate *rx_fd = FFT(rx_td[rxIndex][:])*;
      Calculate *corr_fd = rx_fd .* conj(self_interference_fd[1:4][1:4])*;
      Calculate *corr_td = IFFT(corr_fd*;
      Calculate *ang[txIndex][rxIndex] = angle(max_peak(corr_td))*;
      if $abs(ang[txIndex][rxIndex] > 2$ **then**
        $pol\_error[txIndex][rxIndex] = 1$;
      end
    end
  end
  // Stage 2: Correct the Polarities;
  if $pol\_error[1:4][1:4]$ *is all* 0*s* **then**
    Terminate. We are done;
  end
  // Stage 2.1: Do we need to flip any tx_pols?;
  for *txIndex = 1:4* **do**
    Count n = the number of 1s in $pol\_error[txIndex][:]$;
    if *n>=3* **then**
      Assign $tx\_pols[txIndex] = tx\_pols[txIndex] * (-1)$;
      Assign $flag = 1$;
    end
  end
  if *(flag == 1)* **then**
    Assign $flag = 0$;
    continue;
  end
  // Stage 2.2: Do we need to flip any rx_pols?;
  for *rxIndex = 1:4* **do**
    Count n = the number of 1s in $pol\_error[:][rxIndex]$;
    if *n>=3* **then**
      Assign $rx\_pols[rxIndex] = rx\_pols[rxIndex] * (-1)$;
      Assign $flag = 1$;
    end
  end
  if *(flag == 1)* **then**
    Assign $flag = 0$;
    continue;
  end
  // Stage 2.3: If we are here, we have two RX and two TX flips. To break the deadlock, randomly flip rx_pols[1];
  Assign $rx\_pols[1] = rx\_pols[1] * (-1)$;
end

(a) TX channel 1 flipped; **(b)** RX channels 1,3 flipped; **(c)** TX channel 1, RX channels 1,3 flipped; **(d)** TX channels 1,3 flipped, and RX channels 1,2 flipped.

$$\begin{bmatrix}1&1&1&1\\0&0&0&0\\0&0&0&0\\0&0&0&0\end{bmatrix},\begin{bmatrix}1&0&1&0\\1&0&1&0\\1&0&1&0\\1&0&1&0\end{bmatrix},\begin{bmatrix}0&1&0&1\\1&0&1&0\\1&0&1&0\\1&0&1&0\end{bmatrix},\begin{bmatrix}0&0&1&1\\1&1&0&0\\0&0&1&1\\1&1&0&0\end{bmatrix}$$

Stage 2 of the algorithm processes the matrix *pol_error* and iteratively flips the polarities of $txIndex$ and $rxIndex$ appropriately. Let us consider the simple example (d) above, where the TX channels 1,3 were flipped, and RX channels 1,2 were flipped when power cycling.

$$\begin{bmatrix}0&0&1&1\\1&1&0&0\\0&0&1&1\\1&1&0&0\end{bmatrix} \xrightarrow{\text{Stage 2.3: Flip } rxIndex = 1}$$

$$\begin{bmatrix}1&0&1&1\\0&1&0&0\\1&0&1&1\\0&1&0&0\end{bmatrix} \xrightarrow{\text{Stage 2.1: Flip } txIndex = 1,3}$$

$$\begin{bmatrix}0&1&0&0\\0&1&0&0\\0&1&0&0\\0&1&0&0\end{bmatrix} \xrightarrow{\text{Stage 2.2: Flip } rxIndex = 2}$$

$$\begin{bmatrix}0&0&0&0\\0&0&0&0\\0&0&0&0\\0&0&0&0\end{bmatrix} \xrightarrow{\text{Terminate. We are done.}}$$

Observe that the algorithm un-flipped all the flipped channels correctly. The reader can work through a few more examples to see that this algorithm converges quickly to an equivalent, but correct set of polarities. As a simple example, consider the case where TX channels 1,3 have flipped, and RX channels 2,3 have flipped. The algorithm will iterate as follows:

$$\begin{bmatrix}1&0&0&1\\0&1&1&0\\1&0&0&1\\0&1&1&0\end{bmatrix} \xrightarrow{\text{Stage 2.3: Flip } rxIndex = 1}$$

$$\begin{bmatrix}0&0&0&1\\1&1&1&0\\0&0&0&1\\1&1&1&0\end{bmatrix} \xrightarrow{\text{Stage 2.1: Flip } txIndex = 2,4}$$

$$\begin{bmatrix}0&0&0&1\\0&0&0&1\\0&0&0&1\\0&0&0&1\end{bmatrix} \xrightarrow{\text{Stage 2.2: Flip } rxIndex = 4}$$

$$\begin{bmatrix}0&0&0&0\\0&0&0&0\\0&0&0&0\\0&0&0&0\end{bmatrix} \xrightarrow{\text{Terminate. We are done.}}$$

We can observe that instead of flipping TX channels 1,3 and RX channels 2,3, the algorithm instead flipped the other TX channels 2,4 and the other RX channels 1,4. These are equivalent outcomes.

In conclusion, this self-calibration of polarity upon power-up can be run in one minute, after which the nodes are ready to be used. Still, for best performance, we recommend performing a full two-node calibration every few days, and running the routines after the nodes have reached steady state temperature.

A Note on Stability: We have observed that the HMC6300 up-converter and HMC6301 down-converter chips demonstrate time-varying phase drifts (albeit over very long time scales). These phase drifts are temperature dependent. We have not yet had the opportunity to calibrate our SDR nodes in an oven that carefully controls the temperature. It is for this reason that our system cannot yet support a mechanism wherein *golden calibration* is performed once in our labs, and then *self calibration* can be performed at run time. We hope to get there soon, but until then, we recommend that the users perform two-node calibration prior to running experiments.

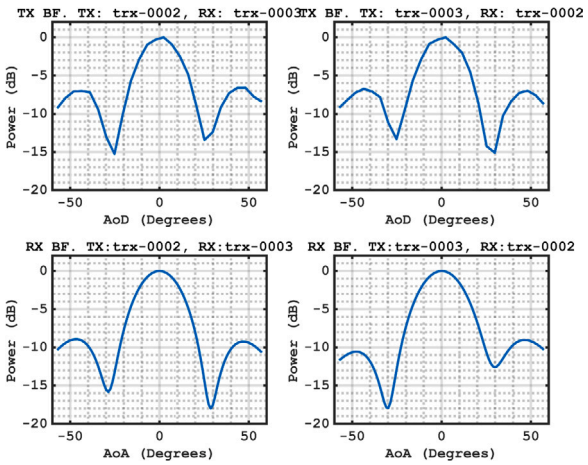

Fig. 9. [14] When the two nodes are placed in boresight, the AoA and AoD values are correctly estimated to be 0 on both ends.

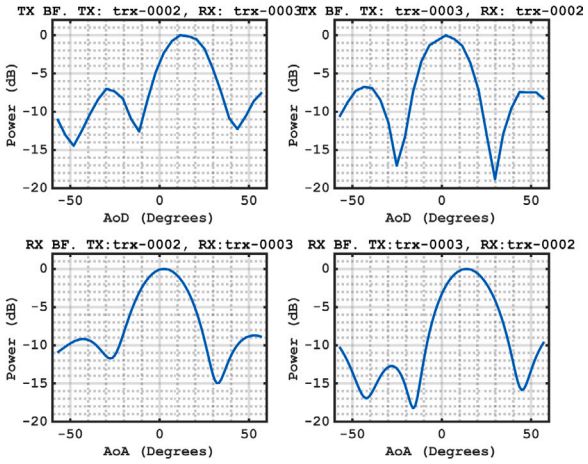

Fig. 10. [14] Beamforming demonstration when node `trx-0002` is rotated counter-clockwise by about 15 degrees.

5. Beamforming demonstration

The ultimate test of whether an array has been calibrated properly is if we can correctly form beams in the required directions. To do this, we first apply all the required calibration factors to the signal to/from each TX/RX channel. To transmit a beam in direction θ (where $\theta = 0$ is boresight), the baseband signal for TX channel $txIndex$ should be multiplied by $e^{j\cdot\pi\cdot txIndex\cdot sin(\theta)}$. This is classical geometric beamforming in the case of antenna elements being spaced $\lambda/2$ apart. Conversely, on the receiver side, to receive a signal in direction θ (where $\theta = 0$ is boresight), the baseband signal from each RX channel $rxIndex$ needs to be multiplied by $e^{j\cdot\pi\cdot rxIndex\cdot sin(\theta)}$ prior to being combined. To demonstrate beamforming on two nodes (A,B), the experiment involves:

- Transmit from a single channel on A. Receive on all RX channels of B. Apply multiple beamforming vectors to *look* in all directions;

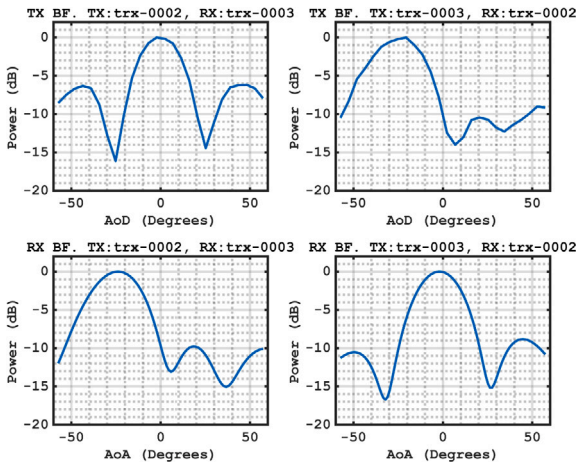

Fig. 11. [14] Beamforming demonstration when node `trx-0003` is rotated clockwise by about 25 degrees.

- Transmit from all channels on A. Sequentially, apply beamforming vectors to scan the transmit beams in different directions; in each case, use a single RX channel on B, and measure the power;
- Transmit from a single channel on B. Receive on all RX channels of A. Apply multiple beamforming vectors to *look* in all directions;
- Transmit from all channels on B. Sequentially, apply beamforing vectors to scan the transmit beams in different directions; in each case, use a single RX channel on A, and measure the power;

Essentially, this demonstrates RX and TX beamforming on both nodes in the link, along with the estimation of the angle of arrival (AoA) and angle of departure (AoD). We first place both nodes (named `trx-0002` and `trx-0003`) directly facing each other and ran the experiment. Fig. 9 shows the TX and RX beam patterns, showing that the AoA and AoD are estimated correctly. Next, we rotated node `trx-0002` counterclockwise by about 15 degrees, and repeated the experiment; the results are shown in Fig. 10, showing that the correct AoA and AoD have been detected. Finally, we returned `trx-0002` to the original position, but rotated `trx-0003` clockwise by about 25 degrees; the results are shown in Fig. 11; again, the correct AoA and AoD were detected. These experiments demonstrate that the node calibration has been performed correctly, as evidenced by successful beamforming and estimation of AoA and AoD values. We do not plot the pre-calibration beams as a point of comparison, because by definition, uncorrected phase and timing offsets lead to meaningless beams.

5.1. OFDM-based physical layer

In this experiment, the TX transmits four (the max allowed with 4 antennas) independent streams of data: one stream in each direction. The RX *looks* in all four directions simultaneously. For each RX angle, it attempts to synchronize and decode *each* of the four transmitted streams. We demonstrate that all four TX streams can be decoded in O(1) time, without any scanning or synchronization overhead. This has been illustrated in Fig. 12. The code, alongside with a detailed tutorial, is provided in [16,18].

6. Conclusion

Getting the Pi-Radio v1 SDR to work correctly required careful calibration. There are many ways in which the behavior of practical devices deviates from ideal, and these need to be calibrated in order to get precise beams: **(a)** crystal frequency offset correction; **(b)** identification of linear operating ranges; **(c)** timing offset corrections; **(d)** LO phase offset corrections; **(e)** magnitude corrections; **(f)** IQ gain imbalance corrections; **(g)** IQ quadrature LO phase imbalance corrections; and **(h)** power-on self-calibration of polarities. We do not claim scientific novelty for any of these methods. Rather, this paper serves as a tutorial for calibrating an SDR node inexpensively and semi-automatically.

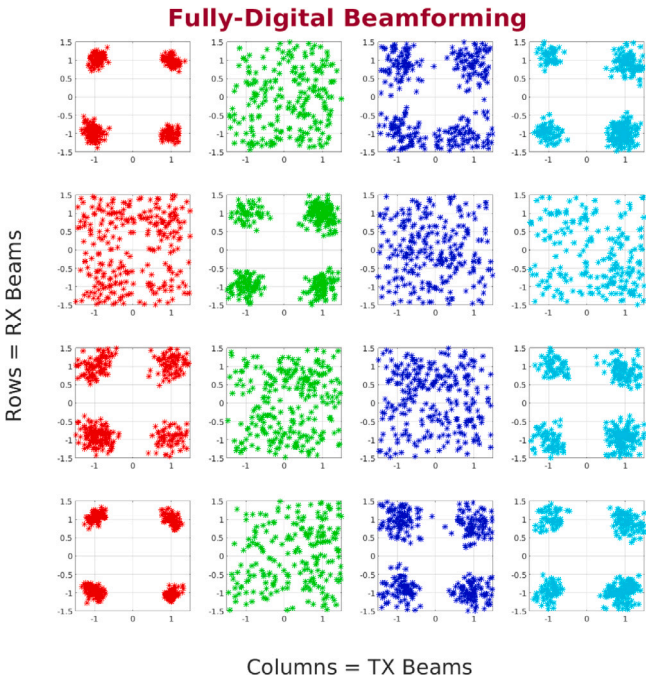

Fig. 12. Constellation plots show the received quadrature phase shift keyed (QPSK) symbols at nominal 60 GHz carrier frequency, when decoded in each of four RX directions, with each RX direction decoding data corresponding to four TX directions. The processing was done in MATLAB (non-real-time), and does not include a coding layer.

Performing calibration has hitherto relied on expensive lab bench equipment like spectrum analyzers and signal synthesizers. In the mmWave frequency range, these are particularly expensive. This paper aims to demonstrate one set of techniques that can be used to calibrate SDRs *in an affordable manner*. All calibration code (as well as Pi-Radio's v1 SDR hardware design schematics) have been released on GitHub [16] using the free and highly permissive MIT license. It is our vision to democratize access to experimental wireless research not only through affordable SDRs that feature advanced transceiver technologies (like fully-digital beamformers), but also through open sourcing calibration code that can be used by the community either on the Pi-Radio SDR platform or any other SDR platform.

CRediT authorship contribution statement

Aditya Dhananjay: Conceptualization, Methodology, Software, Writing - review & editing, Funding acquisition. **Kai Zheng:** Hardware design, Ssoftware. **Marco Mezzavilla:** Visualization, Supervision, Writing - review & editing, Funding acquisition. **Lorenzo Iotti:** Software, Advisory. **Dennis Shasha:** Writing - review & editing. **Sundeep Rangan:** Supervision, Writing - review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The Pi-Radio v1 SDR hardware was designed, built, and tested (along with early calibration efforts) during the performance period of NSF STTR Phase-I Award #1821150, *A Fully-Digital Transceiver Design for mmWave Communications* (Phase-II recently awarded). More advanced calibration techniques were implemented during the performance period of the ARMY STTR Phase-I Award, *Millimeter Waveforms For Tactical Networking* (contract #W911NF20P0038). An early version of the SDR (based on a different architecture) was developed at New York University, funded in part by a NIST, USA grant, *An End-to-End Research Platform for Public Safety Communications above 6 GHz* (Award # 70NANB17H166).

We would like to thank Prof. Arjuna Madanayake and Viduneth Ariyarathna from Florida International University; their work on calibrating their sub-6 GHz and 28 GHz arrays were a starting point in this work. We would also like to thank Prof. Jim Buckwalter, Prof. Mark Rodwell, and Prof. Upamanyu Madhow from the University of California - Santa Barbara for enjoyable discussions about SDR calibration. We would also like to thank Dr. Jaakko Haarla (Aalto University, Finland) and Dr. Vasilii Semkin (UC Louvain, Belgium) for their help with antenna design. Finally, we would like to thank the NSF RCN community for their energy and vocal demands for increased access to affordable real-world mmWave experimentation.

References

[1] F. Khan, Z. Pi, An introduction to millimeter-wave mobile broadband systems, IEEE Commun. Mag. 49 (6) (2011) 101–107.
[2] T.S. Rappaport, S. Sun, R. Mayzus, H. Zhao, Y. Azar, K. Wang, G.N. Wong, J.K. Schulz, M. Samimi, F. Gutierrez, Millimeter wave mobile communications for 5g cellular: It will work! IEEE Access 1 (2013) 335–349.
[3] S. Rangan, T.S. Rappaport, E. Erkip, Millimeter-wave cellular wireless networks: Potentials and challenges, Proc. IEEE 102 (3) (2014) 366–385.
[4] J. Andrews, S. Buzzi, W. Choi, S. Hanly, A. Lozano, A. Soong, J. Zhang, What will 5g be? IEEE J. Sel. Areas Commun. 32 (6) (2014) 1065–1082.
[5] A. Ghosh, T.A. Thomas, M.C. Cudak, R. Ratasuk, P. Moorut, F.W. Vook, T.S. Rappaport, G. MacCartney, S. Sun, S. Nie, Millimeter wave enhanced local area systems: A high data rate approach for future wireless networks, IEEE J. Sel. Areas Commun. 32 (6) (2014) 1152–1163.
[6] C. Dehos, J.L. Gonzalez, A.D. Domenico, D. Ktenas, L. Dussopt, Millimeter-wave access and backhauling: the solution to the exponential data traffic increase in 5G mobile communications systems? IEEE Commun. Mag. 52 (9) (2014) 88–95.
[7] M. Akdeniz, Y. Liu, M. Samimi, S. Sun, S. Rangan, T. Rappaport, E. Erkip, Millimeter wave channel modeling and cellular capacity evaluation, IEEE J. Sel. Areas Commun. 32 (6) (2014) 1164–1179.
[8] T. Bai, R. Heath, Coverage and rate analysis for millimeter-wave cellular networks, IEEE Trans. Wireless Commun. 14 (2) (2015) 1100–1114.
[9] H. Shokri-Ghadikolaei, C. Fischione, G. Fodor, P. Popovski, M. Zorzi, Millimeter wave cellular networks: A MAC layer perspective, IEEE Trans. Commun. 63 (10) (2015) 3437–3458.
[10] M. Mezzavilla, M. Zhang, M. Polese, R. Ford, S. Dutta, S. Rangan, M. Zorzi, End-to-end simulation of 5G mmwave networks, IEEE Commun. Surv. Tutor. 20 (3) (2018) 2237–2263, http://dx.doi.org/10.1109/COMST.2018.2828880.
[11] M. Zhang, M. Polese, M. Mezzavilla, J. Zhu, S. Rangan, S. Panwar, M. Zorzi, Will TCP work in mmwave 5G cellular networks? IEEE Commun. Mag. 57 (1) (2019) 65–71, http://dx.doi.org/10.1109/MCOM.2018.1701370.
[12] K. Zheng, A. Dhananjay, M. Mezzavilla, A. Madanayake, S. Bharadwaj, V. Ariyarathna, A. Gosain, T. Melodia, F. Restuccia, J. Jornet, M. Polese, M. Zorzi, J. Buckwalter, M. Rodwell, S. Mandal, X. Wang, J. Haarla, V. Semkin, Software-defined radios to accelerate mmwave wireless innovation, in: 2019 IEEE International Symposium on Dynamic Spectrum Access Networks, DySPAN, 2019, pp. 1–4.
[13] M. Polese, F. Restuccia, A. Gosain, J. Jornet, S. Bhardwaj, V. Ariyarathna, S. Mandal, K. Zheng, A. Dhananjay, M. Mezzavilla, J. Buckwalter, M. Rodwell, X. Wang, M. Zorzi, A. Madanayake, T. Melodia, MillimeTera: Toward a large-scale open-source MmWave and Terahertz experimental testbed, in: Proceedings of the 3rd ACM Workshop on Millimeter-Wave Networks and Sensing Systems, mmNets'19, Association for Computing Machinery, New York, NY, USA, 2019, pp. 27–32, http://dx.doi.org/10.1145/3349624.3356764.
[14] A. Dhananjay, K. Zheng, J. Haarla, L. Iotti, M. Mezzavilla, D. Shasha, S. Rangan, Calibrating a 4-channel fully-digital 60 GHz SDR, in: Proceedings of the 14th International Workshop on Wireless Network Testbeds, Experimental Evaluation &Amp; Characterization, WiNTECH'20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 40–47, http://dx.doi.org/10.1145/3411276.3412195.
[15] R. Zhao, T. Woodford, T. Wei, K. Qian, X. Zhang, M-cube: A millimeter-wave massive MIMO software radio, in: Proceedings of the 26th Annual International Conference on Mobile Computing and Networking, MobiCom '20, Association for Computing Machinery, New York, NY, USA, 2020, http://dx.doi.org/10.1145/3372224.3380892.
[16] Fully-digital SDR Repo. [Online]. Available:https://github.com/adityadhananjay/fully-digital.
[17] S.W. Ellingson, Correcting I-Q Imlabance in Direct Conversion Receivers. [Online]. Available: https://www.faculty.ece.vt.edu/swe/argus/iqbal.pdf.
[18] Pi-Radio Website. [Online]. Available: https://www.pi-rad.io/.

Dr. Aditya Dhananjay got his Ph.D. in Computer Science from the Courant Institute of Mathematical Sciences at New York University (NYU) in 2015. He then joined the Tandon School of Engineering at NYU as a post-doc, where he gravitated closer to the Electrical Engineering world. His skills and experience are related to embedded SDR design, PCB design, mesh network implementations, digital signal processing, FPGA programming, and various CPU-based programming languages. He was the co-founder of MilliLabs Inc, a spinoff startup working on mmWave channel emulators.

Kai Zheng received his B.S in electronic engineering in Fudan University, Shanghai, China in 2015. He then received his M.S. in computer engineering from New York University, in 2019. He is now a Ph.D. student in University of California, San Diego. His research focus was mmWave wireless communication and software defined radio. During 2015–2017, Kai worked for Huawei Technology (Shanghai) as a hardware engineer where he designed and delivered smartphone projects. In 2019–2020, Kai worked for Pi-Radio, a startup company in Brooklyn NY, and designed mmWave software defined radio systems for research purposes.

Dr. Marco Mezzavilla is a Research Scientist at NYU Tandon School of Engineering, where he leads various mmWave-related research projects, mainly focusing on 5G PHY/MAC design. He received the B.Sc. (2007) and the M.Sc. (2010) in Telecommunications Engineering from the University of Padova (Italy), and the Ph.D. (2013) in Information Engineering from the same university. He held visiting research positions at the NEC Network Laboratories in Heidelberg (Germany, 2009), at the Telematics Department at Polytechnic University of Catalonia (UPC) in Barcelona (Spain, 2010) and at Qualcomm Research in San Diego (USA, 2012). He has authored and co-authored multiple papers in conferences, journals and some patent applications. He is serving as reviewer for many IEEE and ACM conferences, journals and magazines. His research interests include design and validation of communication protocols and applications to Fourth-generation (4G) broadband wireless technologies, millimeter wave communications for 5G networks, multimedia traffic optimization, radio resource management, spectrum sharing, convex optimization, cognitive networks and experimental analysis.

Dr. Lorenzo Iotti received the B.S. and M.S. degrees in electrical engineering and the Ph.D. degree in microelectronics from the University of Pavia, Italy, in 2011, 2013, and 2017, respectively. His doctoral research was focused on CMOS integrated circuit design for mm-wave LO generation. In 2013, he was a research intern at CEA Leti, Grenoble, France, working on silicon photonics electro-optical modulators. From 2017 to 2020 he was a postdoctoral researcher at the University of California at Berkeley, USA, where he was mainly involved in integrated transceiver design for next-generation wireless applications. He is currently an IC Design Engineer at Nokia, New York, USA, working on integrated optical transceivers.

Prof. Dennis Shasha is a Julius Silver Professor of computer science at the Courant Institute of New York University and an Associate Director of NYU Wireless. In addition to his fascination with wireless computing, he works on meta-algorithms for machine learning to achieve guaranteed correctness rates; with biologists on pattern discovery for network inference; with physicists and financial people on algorithms for time series; on database tuning; and tree and graph matching. Because he likes to type, he has written six books of puzzles about a mathematical detective named Dr. Ecco, a biography about great computer scientists, and a book about the future of computing. He has also written technical books about database tuning, biological pattern recognition, time series, DNA computing, resampling statistics, and causal inference in molecular networks. He has written the puzzle column for various publications including *Scientific American*, *Dr. Dobb's Journal*, and currently the *Communications of the ACM*. He is a fellow of the ACM and an INRIA International Chair.

Prof. Sundeep Rangan received the B.A.Sc. at the University of Waterloo, Canada and the M.Sc. and Ph.D. at the University of California, Berkeley, all in Electrical Engineering. He has held postdoctoral appointments at the University of Michigan, Ann Arbor and Bell Labs. In 2000, he co-founded (with four others) Flarion Technologies, a spin-off of Bell Labs, that developed Flash OFDM, the first cellular OFDM data system and pre-cursor to 4G cellular systems including LTE and WiMAX. In 2006, Flarion was acquired by Qualcomm Technologies. Dr. Rangan was a Director of Engineering at Qualcomm involved in OFDM infrastructure products. He joined NYU Tandon (formerly NYU Polytechnic) in 2010 where he is currently a Professor of Electrical and Computer Engineering. He is a Fellow of the IEEE and the Associate Director of NYU WIRELESS, an industry-academic research center on next-generation wireless systems.