A Python Based Testbed for Real-Time Testing and Visualization using TI's 77 GHz Automotive Radars

Onur Toker
Dept. of Electrical and Computer Engineering
Florida Polytechnic University
Lakeland, FL 33805
otoker@floridapoly.edu

Brent Kuhn
Dept. of Electrical and Computer Engineering
Florida Polytechnic University
Lakeland, FL 33805
bkuhn4669@floridapoly.edu

***Abstract*—In the paper, we introduce a Python based software library and a testbed based for TI's 77 GHz automotive radars using the FPGA based hi-speed ethernet link. The main design objective is to be able to test high level algorithms in real-time, show live results with minimal effort, avoid using low speed serial ports, and build the whole system on FPGA based hi-speed communication. All of these are possible using TI's C/C++ toolchain, but may require much more effort, especially for testing high DSP level algorithms. This is true even if all advanced MATLAB and Python DSP libraries are made available in TI's framework. To be able to test different high level algorithms with less effort, TI is providing a closed source Windows application to record raw ADC data for offline analysis in MATLAB. This setup requires a companion Lattice FPGA board, DCA1000, which sends raw ADC data from all receive antennas as UDP packets. However, this setup is useful only for offline analysis. That's why we have implemented a Python based software library and a testbed for real-time testing and visualization using this raw ADC data. Implemented library does all UDP processing, parsing, and returns `numpy` arrays representing raw ADC data for each receive antenna. A couple of code samples, including a real-time direction of arrival estimation program, are presented to demonstrate the simplicity and usefulness of the developed system.**

***Index Terms*—Automotive radars, Real-time testing and Visualization, Python.**

I. Introduction

Automotive industry's focus on advanced drivers assistance systems (ADAS), and autonomous vehicles (AV) is increasing every year, and this in turn is triggering substantial research on automotive radars. The 77 GHz band is gaining significant importance for automotive radar systems, and multiple transmitter/receiver antenna based systems are becoming increasingly popular. Currently, there are very few commerically available MIMO automotive radars at 77 GHz band. In this paper, we focus on Texas Instruments' (TI) AWR family of automotive radars [1], and demonstrate a testbed for real-time testing and visualization using FPGA based hi-speed links.

For automotive radars with multiple transmit/receive antennas, the DSP problem is more complex because there are usually multiple moving targets, with different distance, velocity, radar cross section, and angular direction relative to the sensor's main axis. There are also multiple antennas, hence multiple ADC streams to be processed. Therefore, there is room for more advanced DSP techniques. However, implementation, testing, and performance analysis of such techniques may require more effort if TI's C/C++ toolchain is used as the development platform.

TI's automotive radar development kits [2] require a high bandwidth connection to be able to send raw ADC data from each receive antenna. With the help of the Lattice FPGA board, DCA1000, raw ADC data can be sent to a processing equipment in the form of UDP packets [3]. The processing element can be either an ethernet enabled Cortex-M class ARM board, a more advanced SoC class Zynq board, or a desktop/laptop computer with an ethernet port. TI also has a closed source Windows application to record and parse this UDP data stream. Recorded data can later be analyzed in MATLAB, but this will be an **offline** analysis. Furthermore, TI's application records data for a short period of time, which imposes a limitation on the experiment duration.

The motivation for this paper comes from the need for real-time (not offline) experiments with no limitation imposed on the experiment duration. Another motivating factor was the need for a high level software development/testing platform for rapid prototyping of advanced MIMO DSP algorithms. The authors have implemented a Python based software library which hides all the complexity of TI's UDP packet capture, reordering, and parsing. With the help of this library, raw ADC data from each receive antenna can be read by simple function calls. For each chirp duration, the DSP developer will be able to get `numpy` arrays containing the raw ADC data of each receive antenna. Currently, all read operations are implemented as blocking reads. Since data is arriving at a very high speed, for simpler tasks where packet loss is not a major concern, current single threaded version seems sufficient. But for more advanced algorithms for which UDP packet loss, equivalently loss of raw ADC data for certain chirps, is a major concern, a multithreaded design may be more suitable. The authors would like to cite a related work, [7], which is about vital signs monitoring using TI's 77 GHz radars. This work is mostly MATLAB/Simulink based, but they also consider various high level agorithms, and do performance evaluation on real data. In [8], [9], FPGA based hardware acceleration of related problems are studied, but both are for FMCW radars operating at lower frequencies.

This paper is organized as follows: In Section 2, brief infor-

978-1-7281-4571-6/19/$31.00 ©2019 IEEE

mation about TI's automotive radars is presented, in Section 3, TI's configuration software for both the AWR development kit and the Lattice FPGA is presented. Architecture of the developed software library, the programming model, and a demo examples are presented in Section 4. In Section 5, we demonstrate the usefulness of the developed system by implementing a direction or arrival estimation algorithm. Finally, in Section 6, we provide some concluding remarks.

II. TI'S AUTOMOTIVE RADARS: A HARDWARE REVIEW

TI has a family of automotive radar integrated circuits are based on the frequency modulated continuous wave (FMCW) architecture. According to the system block diagram given in [1], there is a single voltage controlled oscillator (VCO) at 20 GHz followed by a 4x frequency multiplier. This signal goes to two transmit antennas each having its own power amplifier, and phase shifter. On the receive side, there are four receive antennas each having their own low noise amplifier, mixer, low pass filter, and analog to digital converter (ADC). Rest of the AWR1642 chip is mostly digital consisting of an ARM Cortex-R4F processor, memory, standard peripherals, LVDS interface for high speed ADC output, and other supporting digital subsystems and/or accelerators.

AWR1642 is an FMCW radar operating at 77 GHz with a maximum bandwidth of 4 GHz. The default value for the chirp duration is $t_d = 160$ μs, ADC sampling frequency is 10 MHz, and during each chirp total $N_s = 256$ samples are collected for each antenna. Each chirp is repeated $N_r = 128$ times over a $T_m = 40$ ms time frame. Normally, a single chirp can be used to estimate distance, but the default setup has T_m as the measurement duration, and after N_r chirps of duration t_d, there will be a blank period of duration $T_m - N_r t_d$ where no chirp signal is generated and no data is captured.

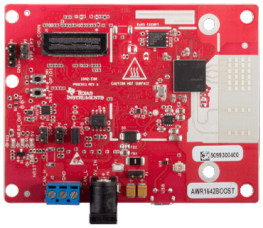

Fig. 1. 77 GHz Texas Instruments AWR1642 automotive radar [2].

TI's AWR1642 development kit has 4 receive antennas, and in its default configuration, it can reach output rates up to several hundred Mbps. Currently, TI has a companion Lattice FPGA board, DCA1000, which communicates with the AWR1642 chip over LVDS, and sends raw data as UDP packets over an ethernet link. With this companion board, raw ADC data can be processed easily on a multitude of devices.

III. TI'S AUTOMOTIVE RADARS: CONFIGURATION

TI's configuration application communicates with the AWR1642 development kit, and the Lattice FPGA DCA1000 over two separate USB cables in the form of serial and SPI formats. Raw ADC data is sent over a separate ethernet cable connected between the DCA1000, and the computer.

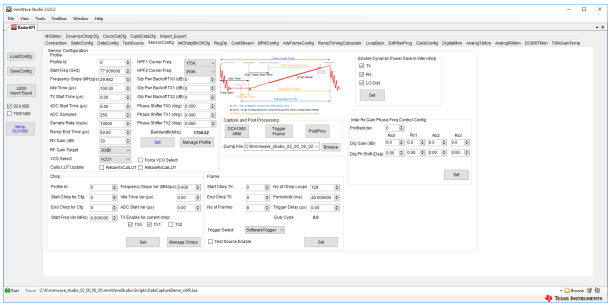

Fig. 2. Texas Instruments AWR1642 automotive radar configuration application [2].

There are tens of parameters that can be set, and then by a trigger message sent from the computer, raw ADC data capture will start and will be sent over the ethernet cable as UDP packets. The format of the UDP packets are described in the next section.

The TI's standard solution requires recording raw ADC data to a file, then doing post processing to reorder UDP packets, and then an open source MATLAB code [4] to read the data file for further processing. But this whole setup is for offline analysis. What we propose in the next section is a Python based real-time analysis and visualization solution using hi-speed FPGA based communication.

IV. DEVELOPED PYTHON LIBRARY

In this section, we describe our Python library, summarize the programming model, and provide a couple code samples to illustrate how to read raw ADC data within our framework.

Let's start with the UDP packet format. The raw ADC data is sent as UDP packets, and payload consists of

- 4 bytes for "sequence number,", which is the UDP packet sequence number,
- 6 bytes for "bytes count," which is the number of bytes in the UDP packet, and
- 48-1462 bytes for raw ADC data.

The authors used the document [4], and the parsing source code given there to develop a Python based solution for UDP packet capture, reordering, and parsing. Note that, ADCs are 16 bit, we have I and Q channels for each antenna, total 4 antennas, and a frame consists of 256 chirps, i.e. $4 \times 2 \times 2 \times 256 = 4096$ bytes for a single chirp. Therefore, data for a single chirp can be split in to multiple UDP packets.

The programming model is basically as follows:

1) Import the `ti77radar` module.
2) Create a `Device('awr1642')` object. Currently only the AWR1642 model is supported.
3) Call the `config(Ns = 256)` method for configuration.
4) Call the `capture_frame()` method to get raw ADC data as a `numpy` array of size `(4,`N_s`)`.

978-1-7281-4571-6/19/$31.00 ©2019 IEEE

5) Do all DSP processing using `numpy`, `scipy`, and possibly other more advanced libraries.
6) Display results using `matplotlib` with or without live animation.
7) Goto Step 4.

The `ti77radar` has the following data capture methods:

- `clear_buffer` clears all buffered UDP data. For more responsive real-time live demos, discarding all previously buffered data is highly recommended.
- `get_channel(k)` returns the raw ADC data captured at antenna k. This will be for a single chirp, and the output format will be a `numpy` array of size $(N_s,)$.
- `capture_frame()` captures raw ADC data for a single chirp, and returns as a `numpy` matrix of size $(4, N_s)$.
- `average_frames(r=8)` captures raw ADC data for `r` chirps, and returns averaged data as a `numpy` matrix of size $(4, N_s)$.

All sample codes are available in the GitHub repo, `https://github.com/onurtoker/ti77radar`, and some of which will be described in more detail in the following subsections.

- `ti77radar` is the main Python module for UDP capture, reordering, and parsing.
- `ti77_oneShotDisp` reads raw ADC data from each antenna, and displays individual results as a 2×2 graph matrix. This example reads data for a single chirp, and does not provide live update feature.
- `ti77_liveDisp` is the real-time live version of `ti77_oneShotDisp`. Instead of displaying 4 graphs, their sum is shown as a single graph.
- `ti77_liveFFT`, `ti77_liveDist` show the real-time FFT of the beat signal, and the real-time target distance.
- `ti77_radialDisp` has implementation of a direction of arrival estimation algorithm. It computes and displays

$$\theta \text{ versus } \max_{\ell} \left| \sum_{n=0}^{N_s-1} \left(\sum_{k=0}^{3} e^{jk\theta} s_{b,k}[n] \right) e^{-j2\pi \frac{n\ell}{N_s}} \right|$$

  where $s_{b,k}[\cdot]$ is the beat signal for the k^{th} antenna. This example reads data for a single chirp, and does not provide live update feature.
- `ti77_liveRadial` is the real-time live version of `ti77_radialDisp`.

A. Single chirp capture example

The following is the most basic usage of our hi-speed FPGA based raw ADC data capture library. Only a single chirp is captured and the corresponding four beat signals are displayed in static format. Python code for this example is `ti77_oneShotDisp` given in the GitHub repo.

```
import ti77radar
import numpy as np
import matplotlib.pyplot as plt

tir = ti77radar.Device('awr1642')
tir.config(NS = 256)
tir.clear_buffer()          # clear kernel UDP buffer
tir.capture_frame()         # capture all RX channels

# generate a subplot for each receive antenna
fig, axv = plt.subplots(2,2)

for k in range(tir.rx):
    s = tir.get_channel(k)  # get channel k data
    sI=np.real(s)
    sQ=np.imag(s)

    axv[k//2, k%2].plot(np.array(range(tir.NS)), sI, 'b-')
    axv[k//2, k%2].plot(np.array(range(tir.NS)), sQ, 'r-')
    axv[k//2, k%2].grid()
    axv[k//2, k%2].legend(['real','imag'])
    axv[k//2, k%2].set_title('RX' + str(k))
    axv[k//2, k%2].set_xlabel('sample index')
    axv[k//2, k%2].set_ylabel('raw ADC output')

plt.show()
```

In Fig. 3, output window of the single chirp capture example is presented.

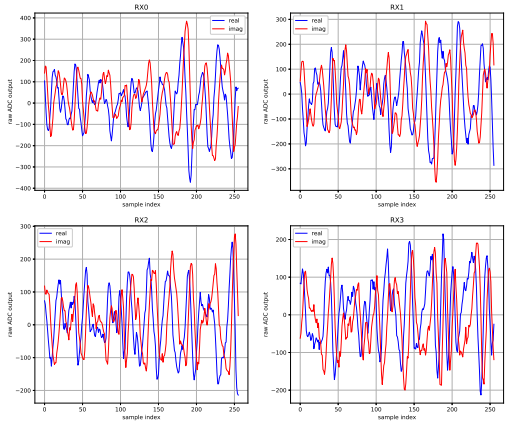

Fig. 3. Output window of the single chirp capture example. Raw ADC data for all receive antennas are shown with real and imaginary parts (I/Q channels).

B. Live beat signal capture example

In this section, we present a live demo example using the Python code `ti77_liveDisp` given in the GitHub repo. There are in total 4 receive antennas, and 4 separate beat signals. This example displays the sum of all individual beat signals in live format. In Fig. 4, output window of the live beat signal capture example is presented.

Compared to the previous sample, there are certain improvements as shown below:

```
f = tir.average_frames(8)
s = np.matmul(np.ones((1,tir.rx)) / tir.rx, f)
```

In this improved code, total of 8 chirps are averaged to obtain a single chirp data. Then data for all of the 4 antennas averaged, and a single averaged result is displayed in live format. To be able to do live display, all DSP operations are written inside the `animate` function of `FuncAnimation`. See the sample code in the GitHub repo for full details.

978-1-7281-4571-6/19/$31.00 ©2019 IEEE

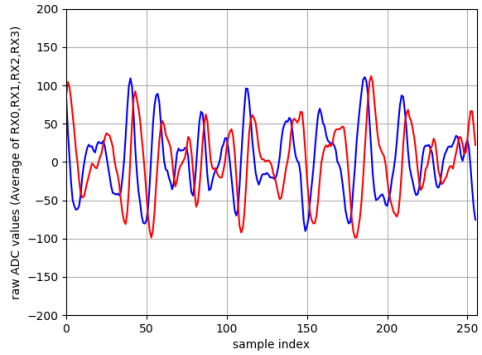

Fig. 4. Output window of the live beat signal capture example (Blue is in the real, and red is complex part).

C. Live FFT based target distance measurment example

This example is similar to the previous one, but instead of the averaged beat signal, target distance is displayed similar to a scope window. The Python code `ti77_liveDist` is used for this example. A total of 8 chirps are captured, and averaged. Then data from all receive antennas are averaged, and an FFT analysis is done to find the beat signal frequency and target distance. All of these are done inside the `animate` function using `numpy` and `scipy` libraries.

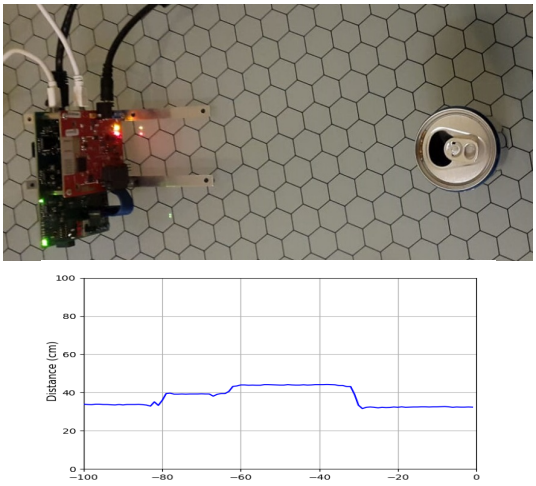

Fig. 5. Live target distance measurement with FPGA based hi-speed ethernet link.

See [5] for a demo video of the experiment. The GitHub repo has the full source code.

V. Real-time direction of arrival estimation

In this section, we consider a direction of arrival estimation algorithm. The authors do not claim anything about the estimation quality of this algorithm compared to other know techniques. What we are trying to demonstrate is how easy is to test a high level algorithm, and see live results quickly. In other words, we are trying to demonstrate the suitability of the developed system for rapid prototyping, and nothing beyond that.

The implemented algorithm is based on solving the problem

$$\arg\max_{\theta}\max_{\ell}\left|\sum_{n=0}^{N_s-1}\left(\sum_{k=0}^{3}e^{jk\theta}s_{b,k}[n]\right)e^{-j2\pi\frac{n\ell}{N_s}}\right|.$$

The following code inside the `animate` function generates a live plot of max of absolute value of FFT of $\sum_{k=0}^{3}e^{jk\theta}s_{b,k}$ versus θ.

```
try:
    tir.clear_buffer()
    f = tir.average_frames(8)
    yv = 0 * tetv
    for (k, tet) in enumerate(tetv):
        w = np.exp(1j * tet)
        s = np.matmul(np.array([w ** p for p in range(tir.rx)]), f)
        s = s.reshape((-1,))
        S = fft(s, M)
        S = np.abs(S[:kmax])
        S = S / M
        yv[k] = np.max(S)
except:
    yv = np.zeros(kmax)
line.set_ydata(yv)
return [line]
```

VI. Conclusion

In this paper, we have presented a Python based real-time raw ADC data capture and visualization library for TI's 77 GHz radars using an FPGA based hi-speed ethernet link. The library is easy to use, and is **not** limited to offline analysis. The GitHub repo [6] has full source code and examples discussed here. Furthermore, a demo video is available in [5]. Since this library is Python based, it can run on Raspberry Pi, and Nvidia Jetson boards as well, however the number of chirps that can be processed per second, will depend on the computational capabilities of the processing device. Developed system can easily be used for autonomous vehicles experiments, and rapid prototyping of MIMO radar algorithms.

Acknowledgment

This work is supported by the Florida Polytechnic University.

References

[1] Texas Instruments AWR1642 automotive radar IC, `http://www.ti.com/product/AWR1642`
[2] Texas Instruments AWR1642 development kit, `http://www.ti.com/tool/AWR1642BOOST`
[3] FPGA based real-time data-capture adapter for TI's AWR development kits, `https://www.ti.com/tool/DCA1000EVM`
[4] DCA1000EVM User Guide, `http://www.ti.com/lit/ug/spruij4a/spruij4a.pdf`
[5] Live distance measurement demo using TI's AWR1642 and the ti77radar Python library, `https://youtu.be/T0YE5dZwlBY`
[6] Github repo for `ti77radar`, `https://github.com/onurtoker/ti77radar`
[7] M. Alizadeh, "Remote vital signs monitoring using a mm-wave FMCW radar," M.S. Thesis, University of Waterloo, Waterloo, Ontario, Canada, 2019.
[8] M. Brinkmann, O. Toker, and S. Alsweiss, "Design of an FPGA/SoC Hardware Accelerator for MIT Coffee Can Radar Systems," IEEE SouthEastCon 2019, Huntsville, AL, April 2019.
[9] M. Brinkmann, "Design and Implementation of Improved Nonlinearity Correction Algorithms for FMCW Radar Sensors," M.S. Thesis, Florida Polytechnic University, Lakeland, FL 33805, August 2019.