

FPGA-Accelerated Coffee-Can Radar Final Report

Will Vu
MIT EECS
Cambridge, MA
willvu@mit.edu

Jason Li
MIT EECS
Cambridge, MA
jason_li@mit.edu

Jia Yuan "Amy" Hu
MIT EECS
Cambridge, MA
jyhu@mit.edu

Abstract—The MIT OpenCourseWare Coffee-Can radar is a 2.4 GHz Frequency-Modulated Continuous-Wave (FMCW) radar used in educational settings to demonstrate fundamental radar concepts. However, due to its low-cost and simple design, it is unable to perform modern radar processing techniques using its original microcontroller-based digital backend. We present a modification of the Coffee-Can radar that integrates FPGA-based sampling and real-time processing to produce a doppler and ranging radar system for tracking fast-moving objects. The modifications include the integration of a DAC and ADC to send and sample FMCW chirps, respectively, a 2-D FFT processing chain to create a range-doppler image, and an HDMI display controller to view the detected radar targets. In this final report, we detail specific efforts towards implementing these modifications and demonstrate a basic radar processing chain that can display target range or doppler.

Index Terms—FMCW, FPGA, Radar, Range-Doppler

I. INTRODUCTION

This report presents a real-time doppler and ranging radar system for tracking fast-moving objects. The radar hardware consists of the MIT OpenCourseWare Coffee-Can Radar system, which is a 2.4 GHz Frequency-Modulated Continuous-Wave (FMCW) radar, created in 2011, that combines low-cost radio frequency components with a simple microcontroller that enable basic radar functionality. The latest revision of the original system uses an Arduino microcontroller to sample and record radar returns. After measurements are performed, the samples are offloaded from the Arduino to a computer where they are processed into radar images using MATLAB (1)(2). This process limits the speed by which radar measurements can be taken, as real-time operation is not possible, and makes correlation of true target range and velocity to radar measurements difficult in an educational setting. Additionally, using the Arduino microcontroller limits sampling bandwidth to 15 kSPS, which results in a significant tradeoff between the radar's range extent and its unambiguous doppler measurement (i.e., maximum doppler velocity before aliasing) that limits the production of usable range-doppler maps (a key radar output that shows both an object's range and velocity simultaneously) (1).

In mitigating the limitations of the original system and enhancing its capabilities, we augment the Coffee-Can Radar by replacing its Arduino-based backend with the Real Digital Urbana FPGA board. By controlling the radar and processing returns with an FPGA, the radar is able to: 1. provide real-time measurements 2. measure range & doppler simultaneously, and

3. achieve greater accuracy in range & velocity measurements as compared to processing on a microcontroller. These benefits are possible through the use of an external 1 MSPS analog-to-digital converter (ADC), which is fast enough to produce non-aliased range-doppler data for relevant targets, such as a moving car traveling under 100 MPH from a few hundred meters away. Additionally, using a 1 MSPS digital-to-analog converter (DAC) allows for the synthesis of custom radar chirps. A version of this FPGA-extended radar system has been demonstrated prior and is documented in (3). However, this report attempts to add additional capabilities by performing a 2-D Fast Fourier Transform on received radar pulses in real-time to produce range-doppler plots that are simultaneously displayed on a screen.

This report documents modifications made to the Coffee-Can Radar and the FPGA pipeline created to enable real-time radar measurements and attempts to perform real-time range-doppler imaging. The report will describe the full radar system shown in Fig. 1, its hardware components, FPGA modules, measurement performance, and conclude with an evaluation the project as a whole in achieving its goal of improving the Coffee-Can Radar system.

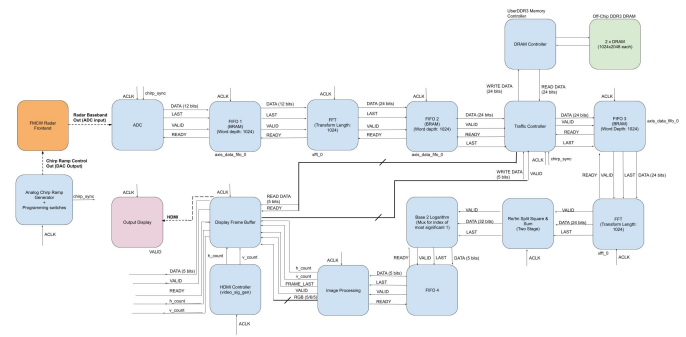


Fig. 1. Block Diagram of the fully-modified Coffee-Can Radar with FPGA processing.

II. HARDWARE INTERFACE

A. Hardware Components

The fully-assembled FPGA-Accelerated Coffee-Can Radar is shown in Figure 2 and consists of:

- MIT OpenCourseWare Coffee-Can Radar frontend
- Custom-built active anti-aliasing filter with variable gain

- Real Digital Urbana Board, containing the Xilinx Spartan-7 XC7S50-CSGA324 FPGA.
- Digilent Pmod AD1 external ADC development board
- Digilent Pmod DA3 external DAC development board

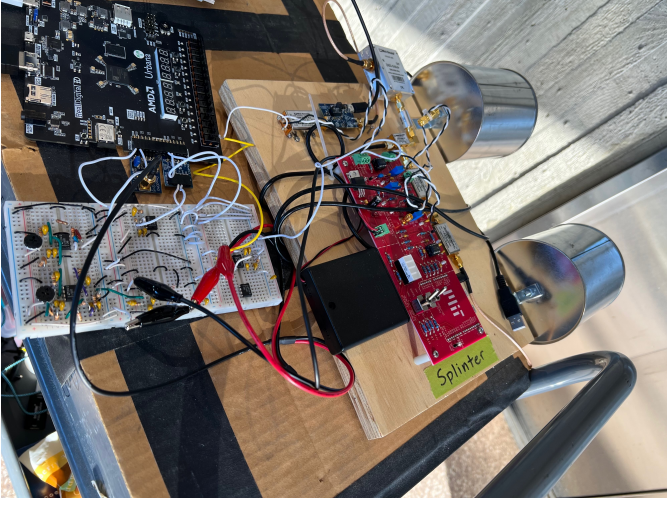


Fig. 2. Complete modified Coffee-Can Radar setup.

The radar frontend, shown in Fig. 3, performs the actual transmission and reception of radar signals towards/from targets. In its ranging-mode, it transmits a 75 MHz linear-frequency ramp at approximately 500 mW. The radar provides an analog baseband signal containing target returns (e.g., scatterings from moving cars). The frequency of the baseband signal within a single frequency ramp period linearly corresponds to a target's range, which can be obtained using a Fast-Fourier Transform (FFT). An additional FFT across multiple frequency chirps provides the doppler shift caused by a moving target, and hence its velocity. A variable gain amplifier (VGA) is used to manually compensate for the varying baseband signal amplitudes found in different environments in order to maximize the dynamic range of the ADC.

B. Radar Chirp Generator

A 1 kHz sawtooth wave, captured in Fig. 4 is generated using the AD5541A at 1 MSPS on the external DAC board attached to the FPGA board's PMOD header. This sawtooth waveform is fed into an op-amp driver circuit, shown in Fig. 5, to control the radar frontend's voltage-controlled oscillator (VCO). The DAC communicates with the FPGA using the Serial Peripheral Interface (SPI) at a 18.5 MHz clock rate.

C. Analog-to-Digital Conversion

An op-amp-based active filter, shown in Fig. 6, precedes the AD7476A used on the external ADC board to isolate signals within its 1st Nyquist region. It consists of a 4th-order Chebyshev filter with 30 dB stopband rejection at 500 kHz and a tunable gain from 20 dB to 54 dB, built using the LT1632.

After low-pass filtering, the ADC digitizes the baseband radar return signal into the FPGA using SPI. A capture of the digital output of the ADC is shown in Figure 7.

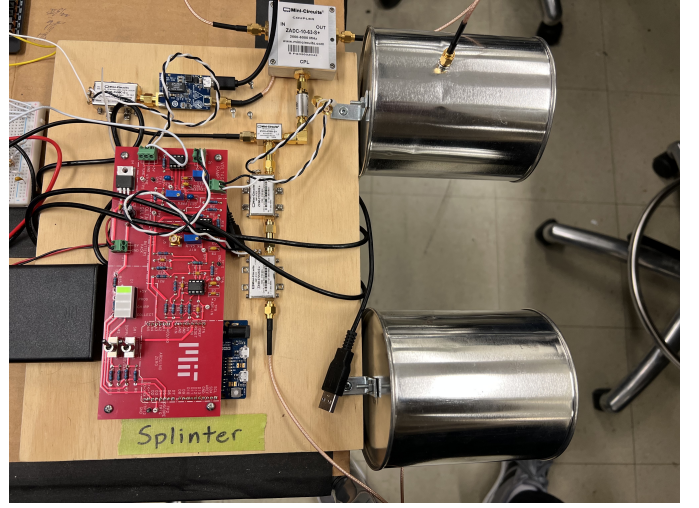


Fig. 3. Radar frontend, modified from the original Coffee-Can radar and includes an additional power amplifier, output coupler and receiver low-noise amplifier.



Fig. 4. Oscilloscope capture of the sawtooth wave used to produce frequency chirps (shown in yellow), the baseband signal (shown in green), and its 1-D FFT (shown in purple)

Both the filter for the baseband signal and driver circuit for controlling the radar's VCO are shown implemented on a breadboard, shown in Fig. 8

III. DIGITAL SIGNAL PROCESSING

Collections of 1024 12-bit samples of the baseband signal from the ADC are stored into a FIFO and then undergo an FFT. The DAC and ADC are synchronized such that the FFT of the baseband signal occurs at the same time instance within a frequency chirp. This ensures that displayed target ranges are consistent across time (i.e., a static target will occupy the same range bin on the display as it updates) and enables range-doppler processing to be performed using a second FFT across the multiple frequency chirps. The modules used to perform this signal processing are connected via AXI-Stream.

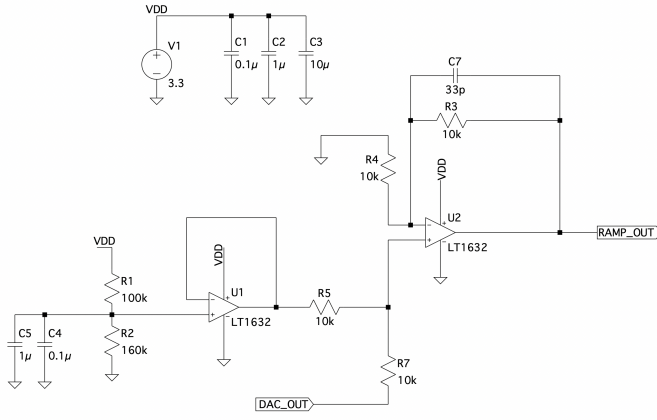


Fig. 5. Circuit schematic of the sawtooth wave generator circuit implemented with LT1632 opamps.

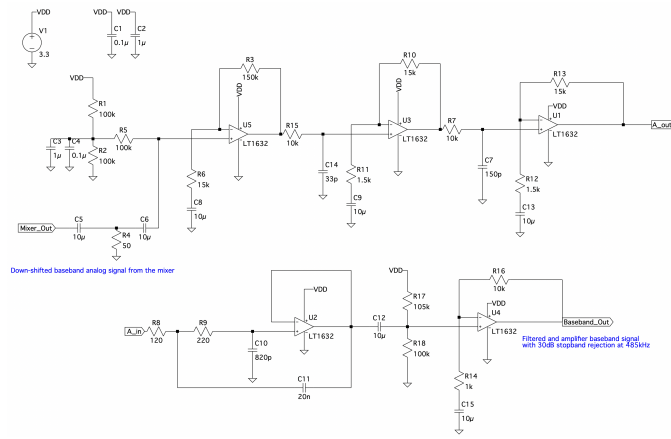


Fig. 6. Circuit schematic of variable-gain active Chebyshev anti-aliasing filter, implemented with LT1632 opamps.

A. Fast Fourier Transform

Discrete Fourier transforms of received radar signals are performed using the AMD LogiCORE™ IP Fast Fourier Transform (FFT) core. The core is configured for a 16-bit input and a 1024-point transform length. The Radix-4 burst architecture is used in order to efficiently utilize FPGA resources (as compared to a fully-pipelined FFT), which provides a processing delay of about 1370 cycles, and is a sufficient conversion speed given the ADC's 1 MSPS sampling rate.

The 1st FFT in the range-doppler processing chain takes in the 12-bit signed output of the ADC and outputs a 16-bit real and 16-bit imaginary number. This complex output is later taken in by the 2nd FFT to perform a 2-D FFT, which also has a 32-bit output.

B. FIFO

Given the desire to have the radar system output a real-time stream of target information onto a display, FIFOs are required to compensate for the burst operation of the FFT, as well as the need to conduct independent Fourier transforms of each received radar chirp. The FIFOs store both the 1024

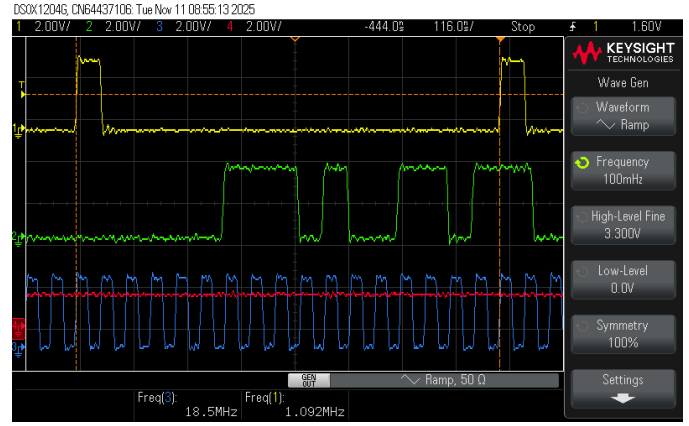


Fig. 7. Oscilloscope capture of the ADC's digital output, including its chip select signal (shown in yellow) indicating the beginning of a 16 cycle conversion operation, the 18.5 MHz serial clock (shown in blue), and the serial output from the ADC (shown in green).

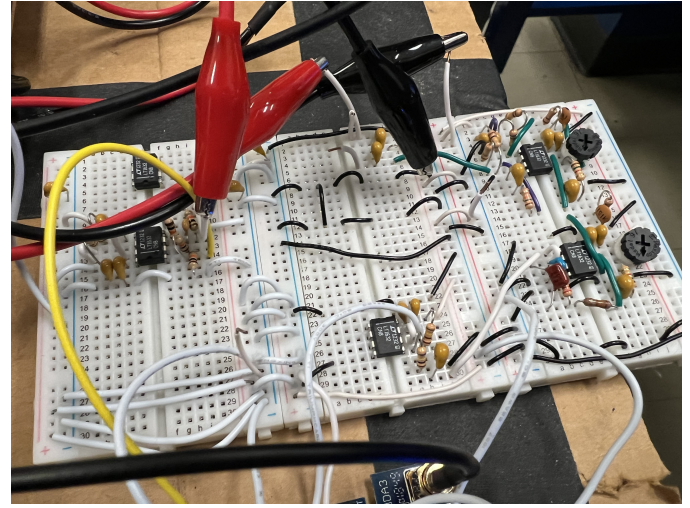


Fig. 8. Breadboard implementation of anti-aliasing filter and ramp driver circuit.

samples taken of a single received radar chirp and the 1024-point frame produced by the FFT of the frequency chirp. A custom FIFO was created using the FPGA's BRAM for this task, and it is combined with a small register-based FIFO to create a resource-efficient complete FIFO with asynchronous read capability.

C. Log2-Magnitude

A two-stage pipelined log2-magnitude calculation is performed on the output of the 2nd FFT to provide a more visible scaling when displayed. Both the 16-bit real and imaginary number outputs are squared and then summed together. The base-2 logarithm of this 32-bit magnitude is taken to reduce it to 5-bits that are then sent to the display. These calculations are spread across 2 pipeline stages to meet timing.

IV. MEMORY INTERFACE

The FPGA's DRAM contains two regions used for two purposes: the FFT DRAM region provides a cache for the 2-D FFT and the Frame Buffer DRAM region is used to display pixels over HDMI. Access to these caches are controlled by a Traffic Generator.

The 2nd FFT in the 2-D FFT must be conducted over points of the same time index within a frequency chirp's transform. Given that the transform length of both FFTs are 1024 points, this requires the storage of 1024 chirp transforms from the 1st FFT, totaling in the storage of 1024x1024 32-bit points. Additionally, these stored points must be outputted "orthogonally" of the direction they were entered into DRAM. Thus a Traffic Generator was created which stores these points in a 1024x1024 matrix. The output of the 1st FFT is stored row-by-row into the matrix, and once full is then outputted "orthogonally" column-by-column to the 2nd FFT. Two of these 1024x1024 matrices are created in memory, with one being written to and the other being read from at a given point in time. This ensures that reads/writes can occur continuously and that no radar samples are lost.

A. Traffic Generator

In order to provide priority for read/write accesses to the Frame Buffer DRAM region, the Traffic Generator operates in a round-robin format, alternating between two usages: 1) WRITE/READ to FFT DRAM and 2) WRITE/READ to the Frame Buffer DRAM. The Frame Buffer DRAM (1280x720 pixels / 8 pixels per DRAM address chunk) is stacked on top of the FFT DRAM (1024x1024 DRAM address chunks x 2 sections). The Traffic Generator continuously commands reads and writes from DRAM, with reads from the FFT DRAM done vertically (orthogonal to the direction that data was written into memory) in order to enable the future 2-D FFT functionality. The WRITE/READ operations to the Frame Buffer DRAM are performed 40 times more often than operations to the FFT DRAM in order to provide the HDMI display with a sufficiently fast stream of pixel data.

B. Pipeline Skid Buffer

A skid buffer module is adopted from (4) in order to address timing violations created over AXI-Stream connections. It adds a pipeline stage between modules that combinationally isolates either side of a ready/valid handshake.

V. VISUALIZATION

A. Image Processing

The Image Processing module converts the 5-bit log2-magnitude output of the 2nd FFT module into a set of 32 distinct RGB colors. As seen in Figure 10, a rainbow color scale (red, orange, yellow, green, teal, blue) is used, with blue representing the lowest radar return magnitude of 0 and red representing the largest magnitude of 32. However, a 4-bit input of switches on the FPGA board can be used to linearly adjust the color scale. The logic for Image Processing is implemented through a Mealy finite state machine which

alternates between two states: BLANK and LINE. Given that the HD display has a 1280-pixel horizontal width, but the FFT produces 1024-sample long transform, the LINE state is used to plot the FFT data onto the screen while the BLANK state is used to show a static image (chosen to be black in Figure 10). By alternating between these two states, the Image Processing module is able to align each 1024-point line of FFT data vertically with one another (as seen in the non-black region in Figure 10).

B. Frame Buffer

The 6.205 Lab 6 frame buffer logic is used to display the output from Image Processing onto an HDMI monitor. The frame buffer logic has been modified to accept data from Image Processing instead of a camera, and uses the aforementioned Traffic Generator to share DRAM accesses with the FFTs.

VI. EVALUATION

A. Functionality

Overall, the system is able to perform basic radar functions in showing target range vs. time and target velocity over time. While all of the modules and processing steps required for outputting a range-doppler map were created and empirically verified to function properly individually, complete integration of these modules together to create a 2-D FFT was unsuccessful. It was determined that this integration failed due to an unknown issue in the communications interface between the 1st FFT and the Traffic Generator that accesses DRAM. Nonetheless, the processing pipeline was configurable with switchable bypasses that allow the 1st FFT or the FFT DRAM region to be skipped, which allows for range- or doppler-only operation. These bypasses helped to verify that the Traffic Generator, arguably the most complicated module, performs orthogonal FFT data reads/writes correctly while sharing Frame Buffer accesses.

Range-only operation is achieved by bypassing the 1st FFT such that ADC samples end up inputted directly into the 2nd FFT module, as shown in Fig. 9. Additionally, a sawtooth wave is outputted from the DAC to transmit frequency chirps. The output of this operation is shown in Fig. 10. The center of the image corresponds to zero range (from the radar) and movement symmetrically along the x-axis, away from the center, corresponds to farther range. Range measurements taken at 3 different target distances, shown in Fig. 12, show a proportional change in the range bins that are illuminated on the display - indicating properly functioning range measurements.

Velocity- or doppler-only operation is also achieved by bypassing the 1st FFT, with a singular frequency tone transmitted from the radar instead. The output of this operation is shown in Fig. 11. The center of the image corresponds to zero velocity and movement symmetrically along the x-axis, away from the center, corresponds to greater velocity. Velocity measurements taken for two moving targets with changing velocity, shown in Fig. 13, show a proportional change in the doppler bins that

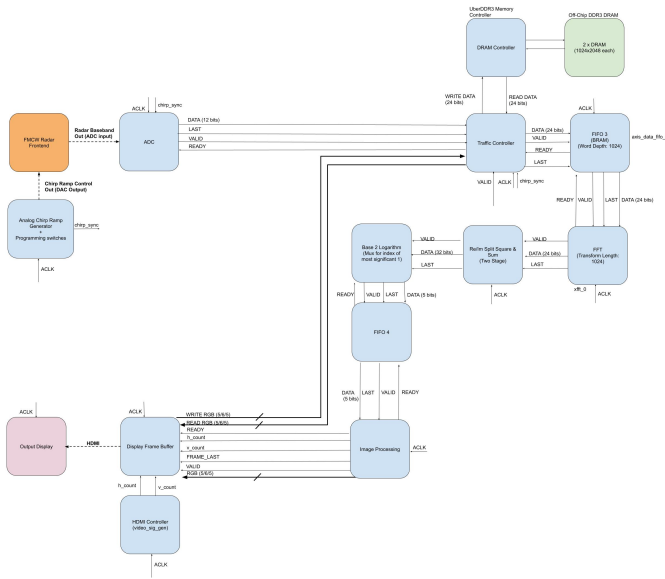


Fig. 9. Block diagram of currently functional radar modifications. The 1st FFT module is bypassed to perform range- or doppler-only measurement.

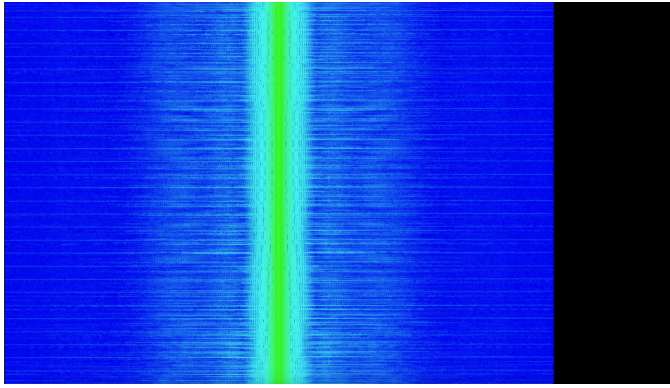


Fig. 10. Displayed radar output showing target range versus time. x-axis corresponds to range; y-axis correspond to time.

are illuminated on the display over time as the target changes speed - indicating properly functioning velocity measurements. Notably, the measured doppler of the targets in Fig. 13 change across the y-axis, showing that the targets have changing velocity with time.

B. Throughput

During normal operation, the system refreshes at a roughly 1 Hz update rate, which matches the calculations during the planning of this project. This rate is limited by the ADC sampling rate and entails that all received radar samples are successfully processed and displayed. This fact is particularly seen in Fig. 10, which shows the display for a target at a static distance from the radar. The constant width of this line indicates that the radar is properly performing and displaying FFTs at the over the same time instances within a particular frequency chirp - if this were not the case, the line would have

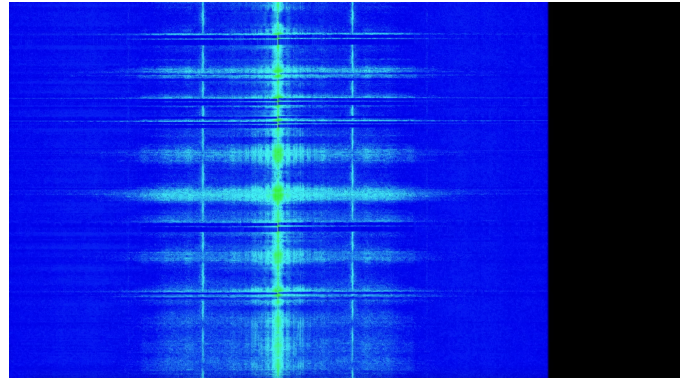


Fig. 11. Displayed radar output showing target doppler versus time. x-axis corresponds to doppler; y-axis correspond to time. The sinc wave in the center corresponds to a moving target

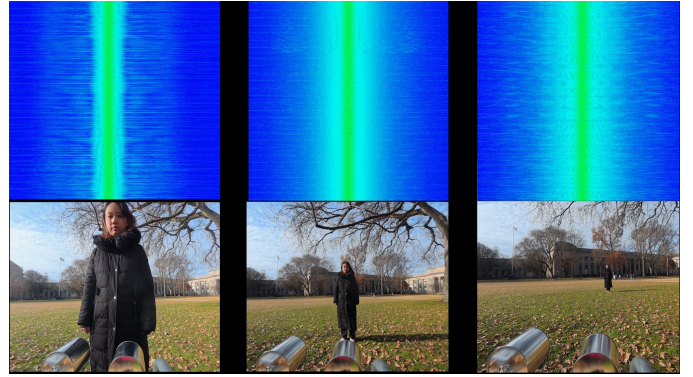


Fig. 12. Displayed radar output showing target range versus time with a static target at 2 m, 4 m, and 10 m from left to right.

significantly varying width over time as the FFT occurs over chirp transition periods.

C. Resource Usage

As shown in Table I, we utilize 12.85% of LUTs, 48.67% of BRAM, 28.3% of DRAM, and 5.83% of DSP Slices. The DRAM usage is calculated as follows: The FFT DRAM stores $2 * 1024 * 1024 * 128$ bits and the magnitude DRAM stores $1280 * 720 / 8 * 128$ bits, which sums up to 35.4 MB.

DRAM memory usage could be optimized by storing the 5-bit pixel magnitude data instead of the 16-bit RGB pixel data, then do inputting it into Image Processing afterwards to then be displayed. This would enable us to pack 25 pixels instead of 8 pixels per memory chunk to reduce the size of the Frame Buffer DRAM section from 14.7MB to 4.72MB.

	Used	Util %
LUT	4190	12.85
BRAM	36.5	48.67
DRAM	35.4 MB	28.3
DSP Slices	7	5.83

TABLE I
MEMORY USAGE

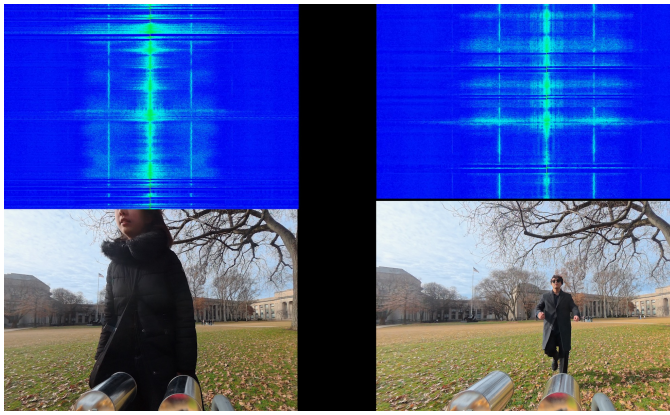


Fig. 13. Displayed radar output showing target doppler versus time with two targets moving at varying velocities. The left pair of images show a target de-accelerating towards the radar while the right pair of images show a target accelerating away from the radar.

D. Timing Requirements

The integration of processing modules together within the FPGA are able to satisfy timing requirements, using a combination of FIFOs, skid buffers, and pipelining stages within modules. The WNS of the final build was 0.077 ns.

E. Deliverable

The most recent project progress accomplishes the minimal checkoff commitments stated in this project's proposal, which were to:

- Receive radar data – implying the successful control of the radar hardware, transmitting and receiving radar chirps, and digitizing them into the FPGA using an external ADC and DAC.
- Perform a 1-D FFT and display target range (or doppler) onto a monitor – implying successful interfacing between the ADC, DAC, FFT IP, FIFOs, and display frame buffer for interfacing with an HDMI monitor.

Since the preliminary report, improvements to the entire 1D FFT pipeline brought alignment to the display of the 1024-point FFT transforms onto a monitor. Moreover, all modules for performing a 2-D FFT were created and experimentally verified to operate properly. Integration of this 2-D FFT pipeline was attempted and fails only at a single communication point. Additional improvements were made including:

- Computing and displaying a 16-bit FFT (as opposed to the 12-bit FFT used previously).
- Dynamic adjustment of output colormap on the HDMI display.
- Variable alignment of center-point (i.e., zero velocity/range line) on the HDMI display.
- Verification of orthogonal FFT reads/writes into the FFT DRAM region while simultaneous accessing and prioritizing the Frame Buffer DRAM via the Traffic Generator.
- Upgrades to transmitter power, receiver sensitivity, and transmit/receive isolation for the radar frontend.

VII. CONTRIBUTION

Jason contributed to the whole interface between the radar hardware and the FPGA including creating filters, driver circuits, and ADC/DAC communication and synchronization. Will developed the system architecture and implemented the FFTs and FIFOs. Amy created the Traffic Generator and Image Processing logic. All authors contributed to integrating and debugging the modules, writing the report, creating figures, and creating the video.

VIII. RETROSPECTION

Implementation of this radar system required careful consideration of the AXI-stream connections used to communicate between modules. Because of the linear pipeline that the radar signal processing chain follows, incorrect communication (even if infrequent) could cascade over time and result in the radar failing. These lessons were learned during the integration of the processing pipeline and resulted in modules being modified to be more resilient to communication failures using FIFOs and skid buffers, and the careful implementation of the AXI-Stream protocol.

Future work on the radar system would focus on enabling the 2-D FFT functionality in order to produce range-doppler plots. Additionally, further signal processing to improve the displayed output would be conducted, including the variable FFT windowing, clutter rejection to subtract the significant zero-velocity radar seen on the display, and digital low-pass filtering to smooth out the noise generated by the FFT IP. Implementation of variable FFT windowing could be done with minimal additional effort as the logic was already attempted and an interface system with the FFTs has been devised that could easily utilize existing skid buffer modules to ensure compatibility with the FFTs.

IX. SOURCE CODE

The following repository contains the Verilog, Coctb/Vicoco simulation testbench, analog front-end circuit schematic, oscilloscope captures, and MATLAB code: [github.mit.edu/6205F25/fa25-6205-team37/tree/radar-32bit-fft](https://github.com/6205F25/fa25-6205-team37/tree/radar-32bit-fft).

X. ACKNOWLEDGMENT

The authors of this report express their deepest gratitude to the 6.205 course instructor, Joe Steinmeyer, and to their project mentor, Monica Chan, for their guidance throughout this project. We also acknowledge all the remaining 6.2050 course staff for all the insightful discussions. Additionally, the authors thank Ken Kolodziej at MIT Lincoln Laboratory for lending the radar frontend hardware for use in this project.

REFERENCES

- [1] G. L. Charvat, J. H. Williams, A. J. Fenn, S. Kogon, and J. S. Herd. (2011) Build a small radar system capable of sensing range, doppler, and synthetic aperture radar imaging. *MIT OpenCourseWare*. [Online]. Available: <https://ocw.mit.edu/courses/res-ll-003-build-a-small-radar-system-capable-of-sensing-/>

range-doppler-and-synthetic-aperture-radar-imaging-/
january-iap-2011/

- [2] P. Gogoi, M. Mandal, and C. Li, "Development of a portable cots-based cw radar system at 2.4 ghz," *IEEE Microwave Magazine*, vol. 26, pp. 37–52, 11 2025.
- [3] M. Brinkmann, O. Toker, and S. Alsweiss, "Design of an fpga/soc hardware accelerator for mit coffee can radar systems," in *2019 SoutheastCon*. IEEE, 2019, pp. 1–5.
- [4] C. E. LaForest. (2019) Pipeline skid buffer. *FPGA Design Elements*. [Online]. Available: https://fpgacpu.ca/fpga/Pipeline_Skid_Buffer.html