CS 437 EKS Extended Document - Raw ADC Data Capture

This document provides the instructions of collecting and interpreting raw ADC data from the TI IWRL6432BOOST radar. To start:

- Make sure you have all **Prelab 4 files and configurations** done, including:
  - Anaconda and Industrial Visualizer Setup
  - Radar flashed with proper .appimage file

  These should be already configured if you are following the lab schedule.
- Download the following files from canvas:
  - `ADCDataCapture.cfg` - This special configuration file disables all onboard processing and only aims to output raw ADC data.
  - `parse_bin_data.py` - This converts the collected .bin files from the Industrial Visualizer to .npy files we can work with.

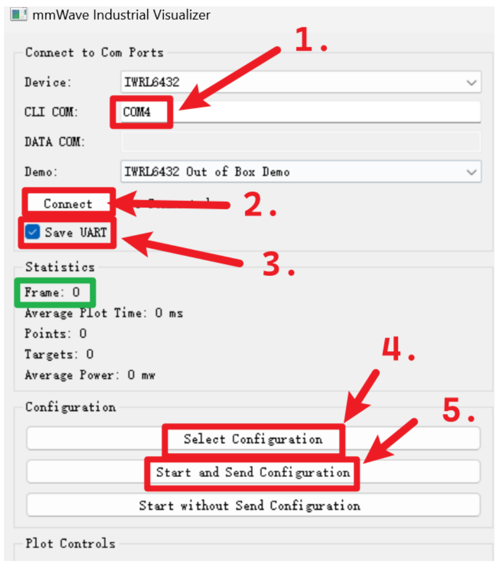
For any questions: Feel free to come to office hours on Tuesday 3:30-4:30 or directly message *Jizheng (Daniel) He* on Slack.

Collecting Raw ADC Data

First, disconnect and reconnect the radar to your laptop. This should be performed every time you perform data collection, or else the visualizer freezes.

Next, follow the steps according to the screenshot below:

1. Set your CLI COM. Details are in Prelab 4's document.
2. Connect to the radar.
3. Check "Save UART". This will store the raw .bin data files under **`/path/to/Industrial_Visualizer/binData/<timestamp_of_experiment>/`**.
4. Select the provided `ADCDataCapture.cfg`.
5. Start the data collection by clicking "start and send configuration". You can start moving when you see the frame count (green block in screenshot) start incrementing.



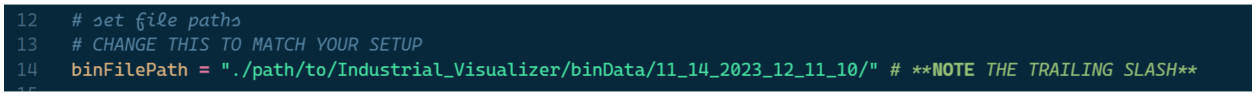
Device: IWRL6432
CLI COM: COM4
DATA COM:
Demo: IWRL6432 Out of Box Demo
[✓ Save UART]
Frame: 0
Average Plot Time: 0 ms
Points: 0
Targets: 0
Average Power: 0 mw

Ignore all "`Warning: invalid TLV type: 316`" messages given by the visualizer in the command line.

Note that there are no range profile or point cloud visualizations in the visualizer - they are disabled for maximum raw data throughput. Close the industrial visualizer when you collect enough data. The number of frames collected will always be a multiple of 100.

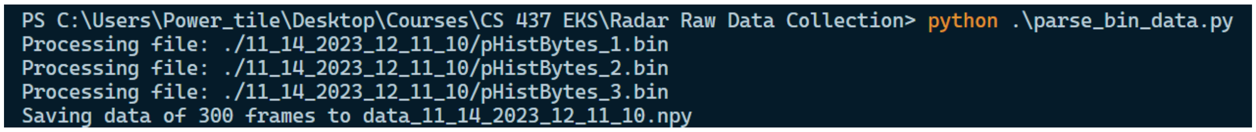
Converting .bin files to .npy file

You should see a folder named as your experiment timestamp (something like `11_14_2023_12_11_10`) in the industrial visualizer directory. Open `parse_bin_data.py` and set the variable binFilePath to the folder. **Note: You must include a trailing slash (/) in your path.**



```
# set file paths
# CHANGE THIS TO MATCH YOUR SETUP
binFilePath = "./path/to/Industrial_Visualizer/binData/11_14_2023_12_11_10/" # **NOTE THE TRAILING SLASH**
```

Run `parse_bin_data.py` and the parsed file named `data_<timestamp>.npy` will be generated in the same directory as the python file.



```
PS C:\Users\Power_tile\Desktop\Courses\CS 437 EKS\Radar Raw Data Collection> python .\parse_bin_data.py
Processing file: ./11_14_2023_12_11_10/pHistBytes_1.bin
Processing file: ./11_14_2023_12_11_10/pHistBytes_2.bin
Processing file: ./11_14_2023_12_11_10/pHistBytes_3.bin
Saving data of 300 frames to data_11_14_2023_12_11_10.npy
```

You can load the .npy file with the following starter code:

```
rawData = np.load("./path/to/data_<timestamp>.npy")
data = np.array([frame["adcSamples"][:, 128:] for frame in rawData])
data.shape # chirp* rx * samples per chirp
```

Interpreting Raw Data

The `data` variable after the starter code parsing is a 3-dimensional array containing integer raw ADC samples. The dimensions are in [chirps, receiving antennas, samples per chirp]. An example dimension will be `(300, 3, 128)` indicating we have 300 chirps. Each chirp transmitted is received by each of the three receiving antennas simultaneously. Each chirp received by any receiving antenna is sampled on 128 consecutive time points.

Some color-coded examples of indexing into data[] is given below. Note the off-by-1 numbers are because Python is zero-indexed.

- `data[10, 2, :]` corresponds to all 128 samples of the 11th chirp transmitted, received and sampled at the third Rx antenna. This is a minimal received and sampled chirp data.
- `data[10, :, 0]` corresponds to the first sample of the 11th chirp transmitted, received and sampled at all 3 Rx antennas simultaneously.
- `data[:, 0, 1]` corresponds to the second sample of all chirps transmitted, received and sampled at the first Rx antenna.

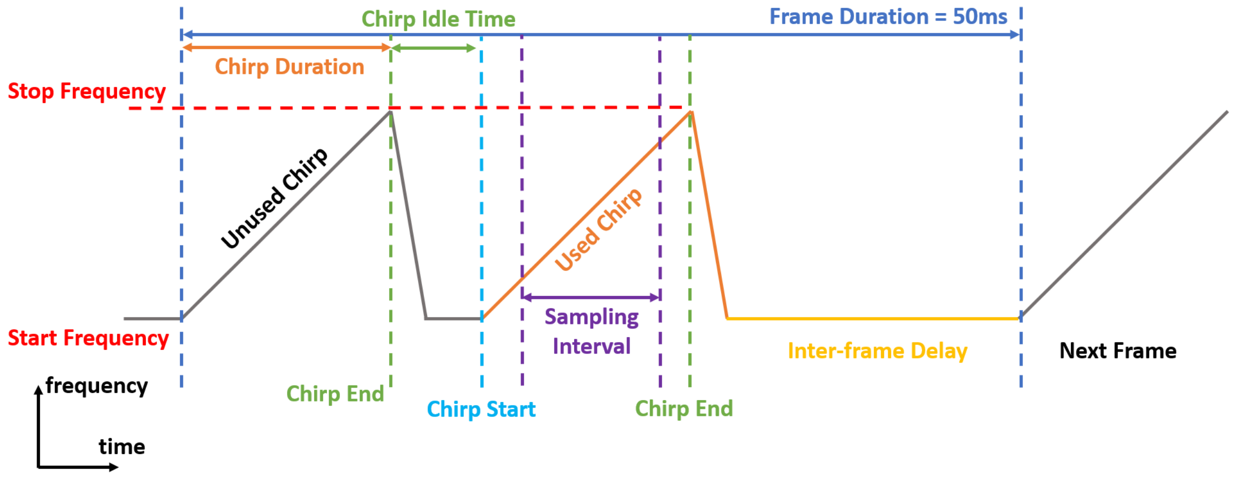
Hint: Taking an FFT across the colon-indexed axis in **one of the above examples** will give you the **range profile**. On top of the result, taking another FFT across **either of the two other axes** will provide the **range-doppler profile** or **range-angle profile**. Figure out which axis is which!

Note: Recall that Doppler FFT is taken across chirps. The time between consecutive chirps is equal to the frame duration, 50 ms in the provided configuration file.

Waveform in a single frame

Under the hood, 2 chirps are transmitted per measurement frame due to hardware limitations, and each chirp is being sampled at 128 timestamps. Empirically, the second chirp is more stable and contains a stronger signal. **We recommend using only the second chirp per frame for processing for simplification. This is already done for you in the starter code:** note the trim `[:, 128:]` on the raw data.

The following figure depicts the actual signal transmitted along the time-frequency axes. The "unused chirp" is the chirp we trimmed away from the raw data. The figure is also not drawn to scale - the inter-frame delay is much larger than the chirp duration.



Parameter List

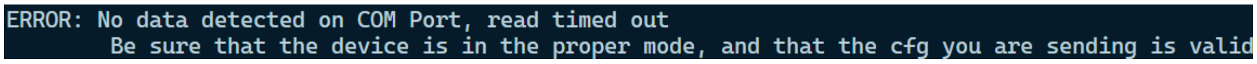
The following table contains the configuration parameters for the provided .cfg file.

Parameter Name	**Value**	**Relative Configuration Parameter**
Start Frequency	58 GHz	chirpTimingCfg.ChirpRfFreqStart
Chirp Duration	20 us	chirpTimingCfg.ChirpRampEndTime
Chirp Slope	220 MHz/us	chirpTimingCfg.ChirpRfFreqSlope
End Frequency	62.4 GHz	Calculated, see next section
Number of Samples Per Chirp**	128 samples	chirpComnCfg.NumOfAdcSamples

Sampling Frequency	8.3333 MHz	chirpComnCfg.DigOutSampRate = 12
Frame Duration	50 ms	frameCfg.framePeriodicity
Chirp Idle Time	6 us	chirpTimingCfg.ChirpIdleTime
Number of Rx antennas**	3 antennas	channelCfg.RxChCtrlBitMask
Number of Idle Samples*	28 samples	chirpTimingCfg.ChirpAdcSkipSamples

* The ADC will wait for (Number of Idle Samples) / (Sampling Frequency) seconds before starting to collect data after a chirp starts, and will stop sampling after all (Number of Samples per Chirp) samples are collected. This corresponds to the sampling interval in the previous figure.

If you would like to change the .cfg file or read more about these parameters, refer to `Additional Documentations/` on Canvas. Do not change parameters marked **. If you get the following error in your Industrial Visualizer's command line after you clicked "Start and send config", your .cfg file is not valid.



```
ERROR: No data detected on COM Port, read timed out
       Be sure that the device is in the proper mode, and that the cfg you are sending is valid
```