AN-027 OPS243-A Rolling Buffer

In many applications there are fast, singular events for which data needs to be captured and processed by different means than the standard radar processing. A very fast example of this is capturing the muzzle velocity of a bullet (2,000 ft/s or 610 m/s). Another is capturing the speed of a bat hitting a baseball which may have an exit velocity of over 100mph.

These events can take as little as 10ms for the object of interest to leave the sensors field of view or detection range. However, the standard radar signal processing flow consists of data capture, processing, and outputting which when pushed to its fastest can take 5-20ms. To help capture these fast events, a new mode is implemented in the API for the OPS243-A called a rolling buffer. In this mode, the sensor is constantly capturing data but not processing it. Based on a trigger signal, the sensor will complete the data capture and output it for further processing. The new feature allows defining how much of the data captured comes from before or after the trigger event.

Table 1. Sample Time vs. Sample Rate and Max Detectable Speed

Sampling Rate	Sampling Time (ms)	Max Detectable Speed (m/s)	Max Detectable Speed (mph)	Notes
5ksps	819.2	15.5	34.7	
10ksps	409.6	31.1	69.5	Default API setting
20ksps	204.8	62.1	139.0	Vehicle, baseball setting
30ksps	136.5	93.2	208.5	Golf ball setting
50ksps	81.9	155.3	347.5	
100ksps	41.0	310.7	694.9	
250ksps	16.4	776.7	1737.4	Bullet muzzle velocity setting
500ksps	8.2	1,553.3	3474.7	Bullet muzzle velocity setting

Rolling Buffer Operation

The rolling buffer is implemented as a special mode in the API for the OPS243-A. In normal operation, OPS243-A performs a sequence of data capture, signal processing, and data output as shown on the left side of Figure 1. When enabling the rolling buffer, the sensor loops in data capture mode until a trigger signal is provided, after which it completes the data capture, and then moves to data output. Signal processing is done off sensor by the customer implementation with reference Python code provided on the OmniPreSense GitHub site.

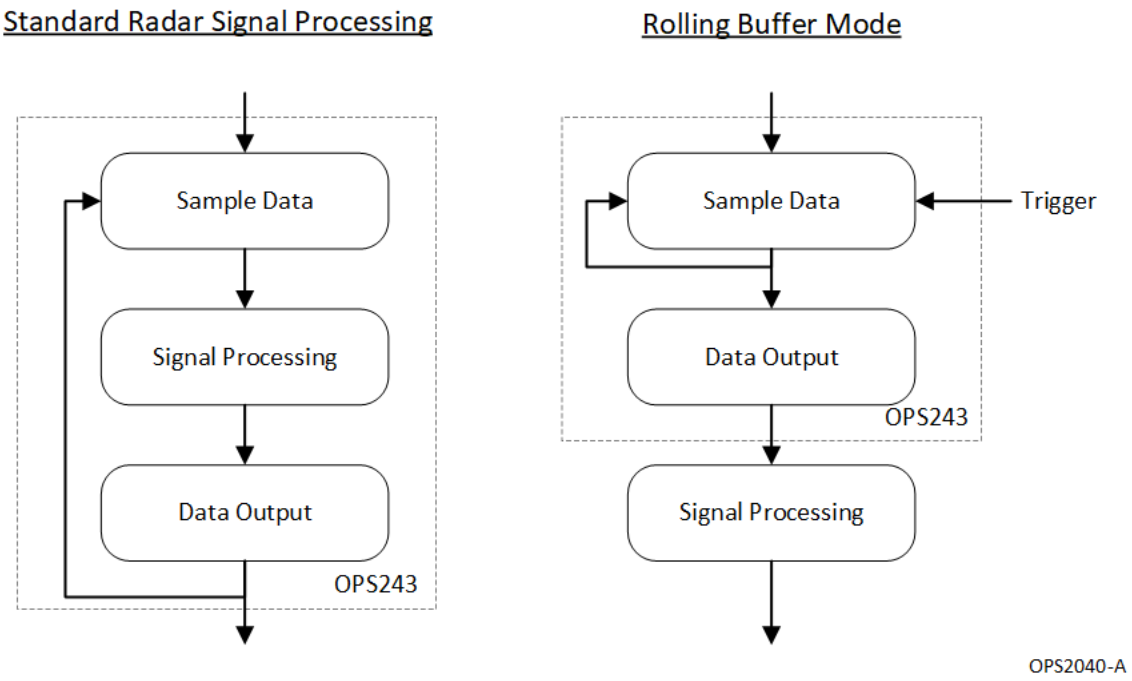

Figure 1. Radar Signal Processing

The rolling buffer is enabled using the GC command. Upon enabling, no further speed data is output and the sensor loops within the data sampling. The rolling buffer is captured at a fixed size of 4096 samples. As seen in Figure 3, the buffer is organized into 32 segments of 128 samples. These segments provide break points at which the trigger signal is captured and determine how much data captured is from before and after the trigger signal.

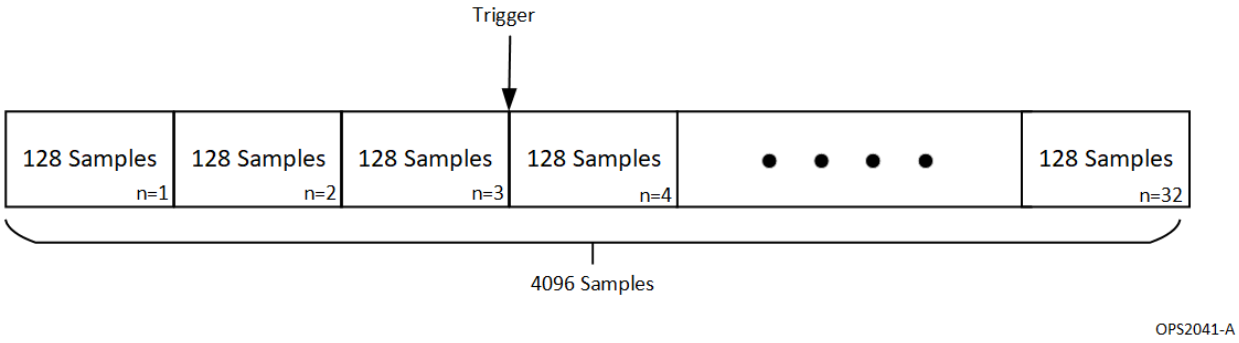

Figure 2. Rolling Buffer Architecture

The trigger signal can be sent via either an API command S! or via a 3.3V signal on pin 3 (HOST_INT) of the 10-pin header J3. The trigger signal is a low to high signal with edge detection used. An API command S#*n* with *n* being 0-32 determines how many prior data samples are included in the final data output. The default setting is n = 8 with 25% of the data (8 segments x 128 = 1024 samples) output being prior sampled data and 75% of the data being newly sampled data. Setting n = 0 outputs no historical samples and only new samples while n = 32 uses only the current samples captured.

Upon completing data sampling, the data is output as a large array of I & Q data along with two timestamps and pauses in Idle mode. The first timestamp is the time of the start of the samples and the second is the time of the trigger signal. A follow-on GC or PA command puts the sensor back into active rolling buffer mode for the next data capture. To get out of the rolling buffer mode, use the G0 command.

API Command Summary

- GC – enter rolling buffer mode, restart new rolling buffer sampling output
- G0 – normal radar signal processing
- S! – API enabled trigger signal
- S#n – set trigger signal old/new sample split, $0 \leq n \leq 32$
- PA – enable new rolling buffer sampling after GC has been set

Trigger Signal

As noted, there are two methods to trigger the sample data capture and output, a software or hardware trigger. The software trigger is a simple command from the processor board to the sensor, S!.

Hardware trigger signals are input on pin 3 of the 10-pin header J3. The signal level is 3.3V and the sensor requires a low to high transition to detect a trigger. Hardware which may be used for trigger signals includes a button press, a microphone, or an IR break beam sensor. Figure 3 provides the connection diagram for an SEN-14262 microphone from Sparkfun. The microphone Gate signal is connected to pin 3 of the OPS243-A, providing the trigger signal. When a loud sound such as a gunshot is heard, the Gate signal provides a low to high transition.

Note, sound is relatively slow for the speeds being detected and the trigger signal timing should be adjusted accordingly. If the microphone/OPS243-A were 2m/6.6 ft. from the sound source, the sound will take ~6.6ms to travel to the sensor and trigger the data capture. If sampling at 250ksps, the 128 sample segment takes 0.5ms to capture. For gunshots, the bullet would already be 0.4m away when the sound reaches the sensor. If the goal is to have some speed data from before the event, say 2ms, adjust the trigger signal to allow for this time and the sound delay. In this case, a total of 8.6ms ahead of the true sound event which is equivalent to ~18 sets of 128 samples. For this example, set S#18 to adjust for this time.

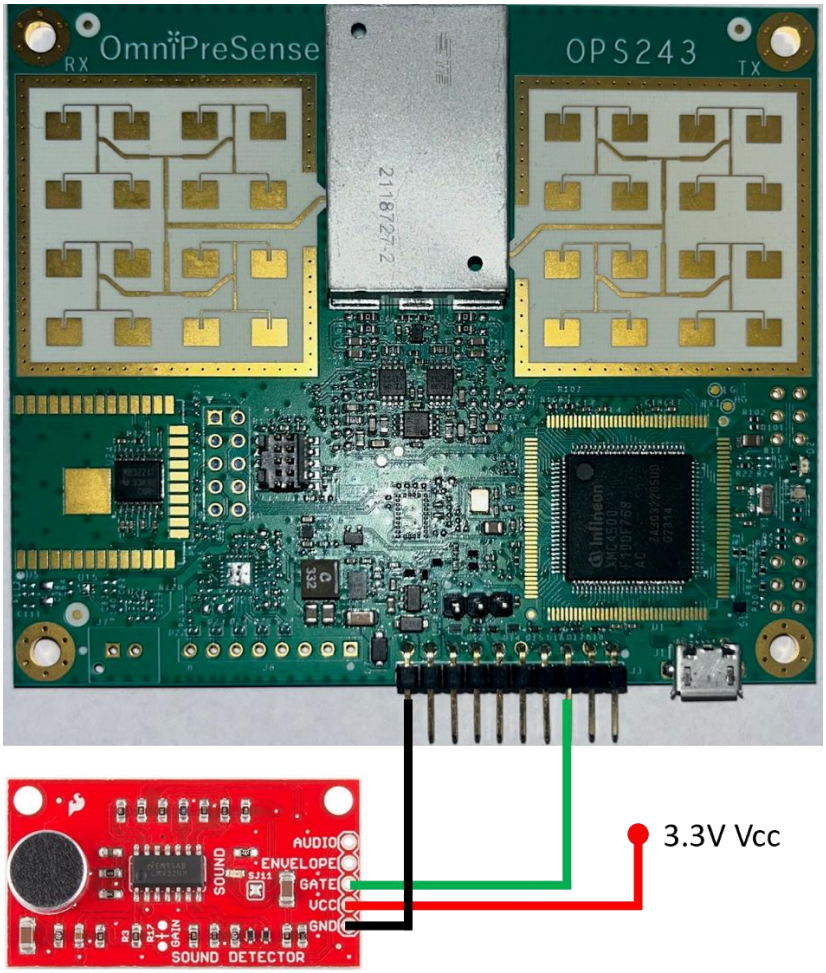

Figure 3. Hardware Microphone Trigger Signal

Python Reference Rolling Buffer and Post Processing Code

Once data is captured the customer can implement their own signal processing methods dependent on their application. Reference Python code is available which enables the rolling buffer mode, sets the trigger, reads out the data, performs signal processing/FFT, and provides final summary output data along with a plot of the data. The Python code flow is shown in Figure 4.

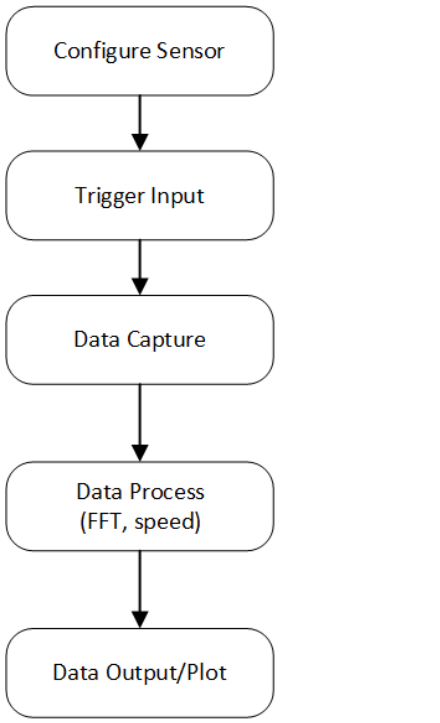

Figure 4. Reference Python Code Architecture

The code does an initial check of the sensor configuration to capture relevant settings such as sample rate. It then sets the sensor into the rolling buffer mode and awaits a trigger signal. The trigger is provided by entering the command Trig upon which the data is captured, processed, and output.

The detailed flow of the post processing is shown in Figure 5. The pre-FFT signal processing includes level shifting the data, converting to voltage levels, and applying a Hann Window function individually on the I & Q sample arrays. The I & Q sample arrays are then combined into a new array of complex value, ready to run through the FFT.

Typically, it's advantageous to have many speed reports to understand how the object is moving. Instead of doing a single 4096 FFT, the Python code takes 128 samples at a time to process through the FFT, providing a final output of 32 speed reports over the sample time. To increase speed resolution, the 128-sample arrays are processed with an FFT size of 4096. The results of the FFT are converted scanned for the top 5 magnitudes with these converted to speed values and output. A final step in the Python code plots the results of the speeds calculated over time (Figure 6).

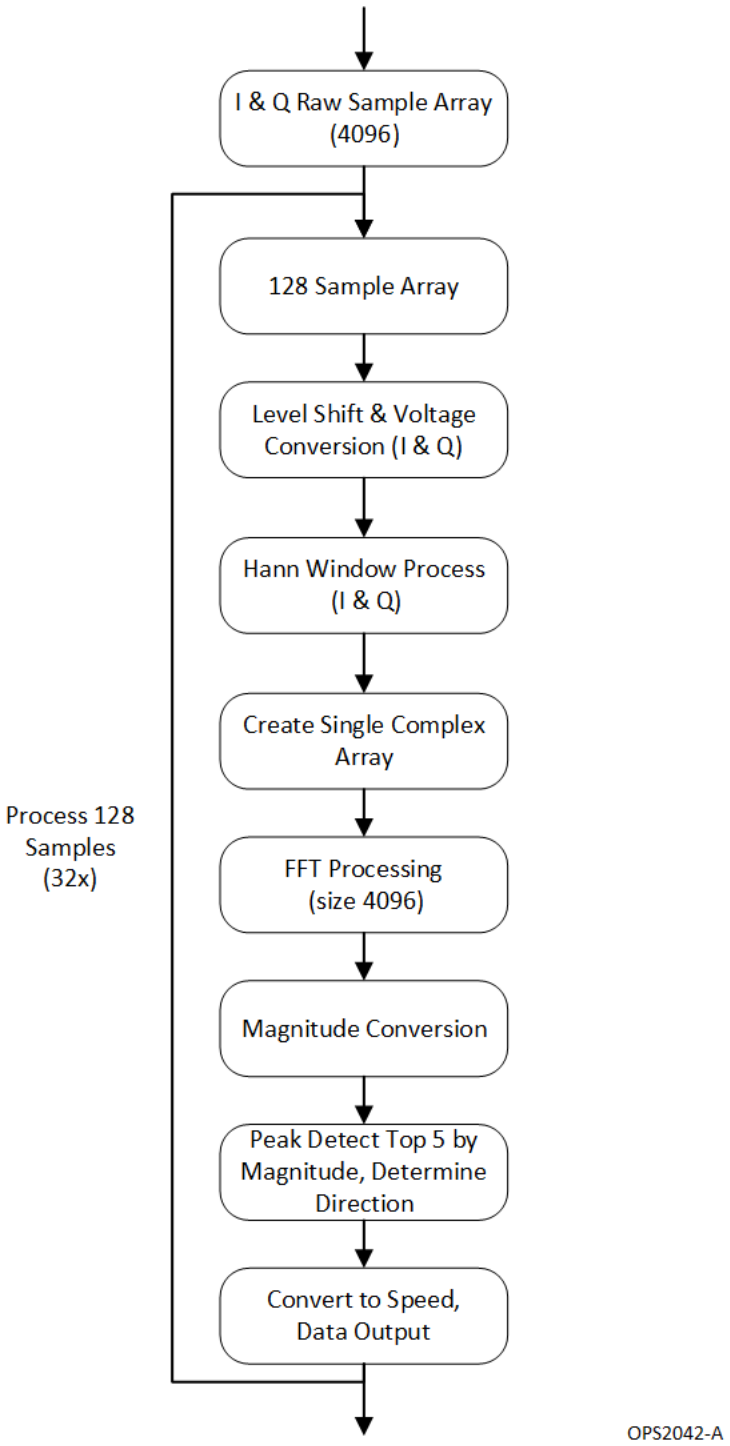

Figure 5. Python Code Post Processing

Example Python Rolling Buffer Results

An example output from the Python code is listed in Appendix A. Basic information about the sensor configuration and processing is provided along with the speed results from each set of 128 samples processed.

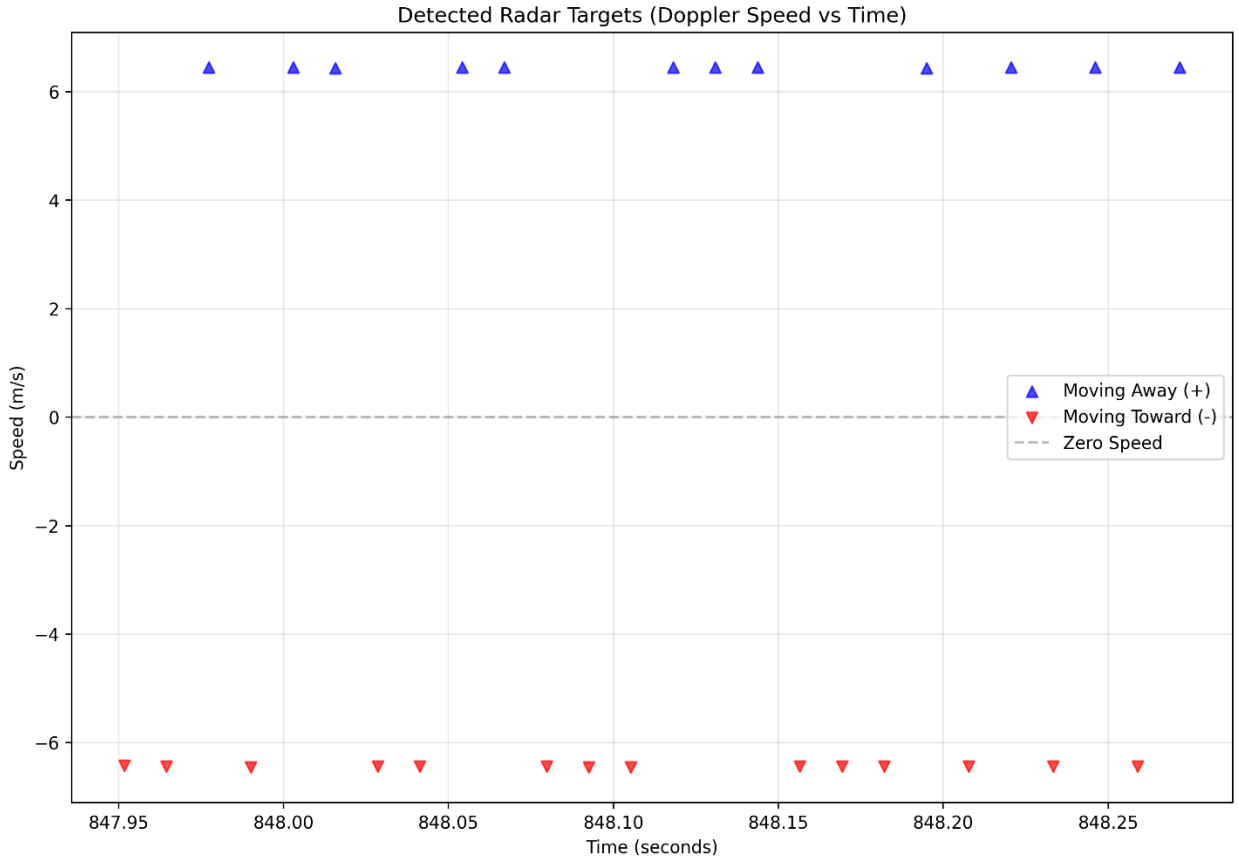

Figure 6. Python Code Speed Data Plot (1024 Hz Tuning Fork or 6.3 m/s)

Appendix A: Example Python Code Output

The results from the Python code are shown below from a 1,024 Hz tuning fork.

```
INFO: Triggering data capture...
INFO: 1st Sample Time: 2059.659
INFO: Trigger time: 2059.761
INFO: Processing 32 blocks of data, trigger_time=2059.761
{"Time": 2059.659, "Block": 0, "Magnitude": [46], "Speeds": [-6.44]}
{"Time": 2059.672, "Block": 1, "Magnitude": [60], "Speeds": [-6.44]}
{"Time": 2059.685, "Block": 2, "Magnitude": [62], "Speeds": [6.44]}
{"Time": 2059.697, "Block": 3, "Magnitude": [59], "Speeds": [6.44]}
{"Time": 2059.71, "Block": 4, "Magnitude": [47], "Speeds": [-6.43]}
{"Time": 2059.723, "Block": 5, "Magnitude": [39], "Speeds": [6.41]}
{"Time": 2059.736, "Block": 6, "Magnitude": [40], "Speeds": [-6.44]}
{"Time": 2059.749, "Block": 7, "Magnitude": [47], "Speeds": [6.46]}
{"Time": 2059.761, "Block": 8, "Magnitude": [48], "Speeds": [-6.44]}
{"Time": 2059.774, "Block": 9, "Magnitude": [45], "Speeds": [-6.44]}
{"Time": 2059.787, "Block": 10, "Magnitude": [44], "Speeds": [6.44]}
{"Time": 2059.8, "Block": 11, "Magnitude": [43], "Speeds": [6.44]}
{"Time": 2059.813, "Block": 12, "Magnitude": [43], "Speeds": [6.44]}
{"Time": 2059.825, "Block": 13, "Magnitude": [43], "Speeds": [-6.44]}
{"Time": 2059.838, "Block": 14, "Magnitude": [45], "Speeds": [6.44]}
{"Time": 2059.851, "Block": 15, "Magnitude": [44], "Speeds": [-6.44]}
{"Time": 2059.864, "Block": 16, "Magnitude": [41], "Speeds": [6.44]}
{"Time": 2059.877, "Block": 17, "Magnitude": [39], "Speeds": [-6.44]}
{"Time": 2059.889, "Block": 18, "Magnitude": [39], "Speeds": [-6.44]}
{"Time": 2059.902, "Block": 19, "Magnitude": [43], "Speeds": [-6.44]}
{"Time": 2059.915, "Block": 20, "Magnitude": [45], "Speeds": [-6.44]}
{"Time": 2059.928, "Block": 21, "Magnitude": [45], "Speeds": [-6.44]}
{"Time": 2059.941, "Block": 22, "Magnitude": [46], "Speeds": [-6.44]}
{"Time": 2059.953, "Block": 23, "Magnitude": [46], "Speeds": [-6.44]}
{"Time": 2059.966, "Block": 24, "Magnitude": [43], "Speeds": [-6.46]}
{"Time": 2059.979, "Block": 25, "Magnitude": [36], "Speeds": [6.46]}
{"Time": 2059.992, "Block": 26, "Magnitude": [34], "Speeds": [6.46]}
{"Time": 2060.005, "Block": 27, "Magnitude": [38], "Speeds": [6.46]}
{"Time": 2060.017, "Block": 28, "Magnitude": [42], "Speeds": [6.44]}
{"Time": 2060.03, "Block": 29, "Magnitude": [41], "Speeds": [6.46]}
{"Time": 2060.043, "Block": 30, "Magnitude": [41], "Speeds": [6.46]}
{"Time": 2060.056, "Block": 31, "Magnitude": [41], "Speeds": [-6.46]}
INFO: Processing complete: 32 blocks with detections
INFO: Plot saved as: radar_plot_20250615_152730.png
INFO: Ready for next command...
Enter command:
```

Revision History

Version	Date	Description
A	June 15, 2025	Initial release.
B	February 6, 2026	Updated API command to send sensor into Rolling Buffer mode to GC.