MRR Software Description

# Inter Frame Processing

The input to inter-frame processing is the radarcube. The radarcube is the buffer to hold 1D FFT samples from every chirp and every combination of Rx and Tx antennas. In case of MRR there are 128 fast chirps and 128 slow chirps and 4 Rx Antennas and 1 Tx Antennas.

For every range bin do the following:

1. Perform FFT on the range in (across multiple chirps or Doppler)
2. Sum the abs values of the FFT from each antenna. And perform log2
3. Perform Doppler cfar on the rangebin
4. Prune the detected peaks by only keeping peaks
5. Restrict the number of detections to 3 based on magnitude of sumAbs (not SNR)
6. For every detected peak from Doppler cfar for a range gate do the following
   1. Load the detected peak details (detected Doppler index, range index, SNR) into the buffer detObj1Draw
   2. Using quadratic interpolation estimate the true peak
   3. Determine the velocity and use it for disambiguation
   4. If detected peaks are less than 3, then set veldisambFacValidit to -2
7. Keep track of all the Doppler lines that have a detection

For all the detected Doppler line do the following:

1. Find peaks along rangebin (range cfar)
2. For every detected peak do the following:
   1. Prune the detected peak (same routine as Doppler)
   2. For every peak detected in range CFAR (check is done if corresponding doppler peak exists, but that is unnecessary) find the following:
      1. Determine speed
      2. Determine range by subtracting range offset of 0.075
      3. Find SNR per antenna for both range and doppler
      4. Convert to oneQformat (7 fractional bits)
      5. Load results into detObj2DRaw. Max number of 2D detections is set to 200

For every 2D detected objects, discard the ones that don’t meet the range vs snr thresholds (needs more analysis)

Max number of 2D detections is 200

Questions to ask:

1. Why is TI finding SNR
2. Why log2

# FFT

The sensor computes fixed point FFT and does not perform scaling of output by k, where k bit-width by 2…

# Buffers

Buffers are allocated by calling the function the MmwDemo\_dataPathConfigBuffers(obj, SOC\_XWR18XX\_DSS\_ADCBUF\_BASE\_ADDRESS) in the MRR\_DSS\_initTask.

MAX\_VEL\_ENH\_PROCESSING is enabled by default for MRR. This sets numChirpTypes to 2. This appears to be fast and slow chirps used to extend the unambiguous Doppler velocity estimation.

There are the following buffers in MRR.

Shaded ones occupy the same memory

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Buffer Name | Size | Units | Comment | MemoryType |
| adcDataIn | 2\*numRangeBins | cmplx16ReIm\_t | Ping-Pong | L1 |
| dstPingPong | 2\*numDopplerBins | cmplx16ReIm\_t | ? | L1 |
| fftOut2D | numDopplerBins | cmplx16ReIm\_t | ? | L1 |
| windowingBuf2D | numDopplerBins | cmplx32ReIm\_t | ? | L1 |
| Log2Abs | numDopplerBins | unint16\_t | ? | L1 |
| sumAbs | numDopplerBins | unint16\_t | ? | L1 |
| sumAbsSlowChrip | 2\*numDopplerBins | unint16\_t |  | L1 |
| detObj2DRaw | maxNumObj2DRw | MmwDemo\_objRaw2D\_t |  | L1 |
| numVirtualAntennas | numVirtualAntennas | cmplx32ReIm\_t |  |  |
|  |  |  |  |  |
| fftOut1D | 2\*numRxAntennas\*  numRangeBins |  |  | L2 |
| cfarDetObjIndexBuf |  |  |  |  |
| cfarDetObjSNR |  |  |  |  |
|  |  |  |  |  |
| radaarCube | numRangeBins\*  numDopplerBins\*  numRxAntennas\*  numTxAntennas\*  numChirpTypes | cmplx16ReIm\_t |  | L3 |
| sumAbsRange | 2\*numRangeBins | Uint16\_t |  |  |

## fftout1D

For every Rx antenna, 1D FFT is computed and stored in the fftOut1D buffer. We need space for 4 antennas in MRR, however the space is twice because after every chirp the data is pushed into the radarcube. While the data is moved to the radar cube the data for the next chirp is moved into the other half of the 1D FFT. So, all even chirps are stored in the lower half of the buffer every odd chirps are stored in upper half.

|  |  |  |
| --- | --- | --- |
| Even/Odd Chrip | Rx Antenna | fftoutID Start Index |
| Even | 1 | 0 |
| Even | 2 | 256 (0+numRangeBins) |
| Even | 3 | 512 |
| Even | 4 | 768 |
| Odd | 1 | 1024 |
| Odd | 2 | 1280 |
| Odd | 3 | 1536 |
| Odd | 4 | 1792 |

# Edma

The following is the working principle of the EDMA. The following parameters (aka paramset) are key to understand the data movement from source to destination

|  |
| --- |
| aCount |
| bCount |
| srcBIdx |
| dstBIdx |
| cCount |
| srcCIdx |
| dstCIdx |

|  |  |
| --- | --- |
| 1 | Move aCount number of bytes from Src to Dst |
| 2 | Increment Src by srcBIdx |
| 3 | Increment Dst by dstBIdx |
| 4 | Perform steps 1 through 3 for bCount times |
| 5 | Increment Src by srcCIdx |
| 6 | Increment Dst by dstCIdx |
| 7 | Perform steps 1 through 6 for cCount times |

|  |  |
| --- | --- |
| MRR\_SF0\_EDMA\_CH\_2D\_IN\_PING | Moves 1D FFT for each range bin to dstPingPong |
| MRR\_SF0\_EDMA\_CH\_2D\_IN\_PONG | Moves 1D FFT for each range bin to dstPingPong |

|  |  |  |
| --- | --- | --- |
|  | MRR\_SF0\_EDMA\_CH\_2D\_IN\_PING |  |
| srcBuff | Radarcube[0] | Pointer to source buffer |
| dstBuff | dstPingPong[0] | Pointer to destination buffer |
| aCount | numDopplerbins\*4 | Acnt of transfer (i.e number of contiguous bytes per transfer) |
| bCount | (numRangebins\*numRxAnt\*numTxAnt\*numChirpType)/2 | Number of EDMA transfers (each of ACnt bytes) |
| srcBIdx | numDopplerBins\*4\*2 | Specifies the byte offset between the source pointer of subsequent EDMA transfers |
| dstBIdx | 0 | Specifies the byte offsets between the destination pointer of subsequent EDMA transfer |

# 2D Window



# Radar Cube

# CFAR Parameter

CFAR input is log2 of the abs values. Need to understand how left and right average is done by adding logarithmic numbers. Shouldn’t we convert to linear scale and then add and divide?

|  |  |
| --- | --- |
| CFAR Configuration in Doppler direction | |
| cfarCfgDoppler.thresholdScale | 2389 |
| cfarCfgDoppler.noiseDivShift | 4 should be log2(2\*winLen) |
| cfarCfgDoppler.guardLen | 4 |
| cfarCfgDoppler-winLen | 8 |

## ThresholdScale

Thresholdscale is calculated based SNR and fractional bits.

# Doppler Lines

A structure called MmDemo\_1D\_DopplerLines is used…

|  |  |
| --- | --- |
| dopplerLineMask | Each bit indicates a doppler bin. So a 32 bit word can record 32 bins. For 128 doppler bins, 128/32 words are needed. The SW performs 128/32 as 122>>5 |
| currentIndex |  |
| dopplerLineMaskLen | Number of integers in dopplerLineMask |

# Angle Processing

/\* Calculate one bin DFT, at detected doppler index. \*/

mmwavelib\_dftSingleBinWithWindow(

(uint32\_t \*)&obj->dstPingPong[pingPongId(rxAntIdx) \* obj->numDopplerBins],

(uint32\_t \*) obj->azimuthModCoefs,

obj->window2D,

(uint64\_t \*) &obj->antInp[rxAntIdx],

obj->numDopplerBins,

DOPPLER\_IDX\_TO\_UNSIGNED(obj->detObj2D[detIdx2].dopplerIdx, obj->numDopplerBins));

Appendix A: Input Signal Modelling

Simulating returns after de-chirping

|  |  |
| --- | --- |
| fr | Frequency of de-chirped signal |
| Ts | ADC Sample Time |
| InitPhase |  |
| DeltaPhase |  |

Appendix B: Radar Cube Extraction

A snapshot of the MRR project’s radar cube is passed from DSS and MSS and then through UART to a connected computer.

RadarCube is 262155 complex numbers. This 256\*128\*4\*2 (number of rangebins\*number of doppler bins\*number of rx antennas\*number of chirp type)

Following steps describe how to extract radarcube for offline processing

|  |  |  |
| --- | --- | --- |
| Steps | Description | File path |
| 1 | Load the radarcapture binary into the sensor | [Link](https://github.com/mohandas98/UavRadarProject_MA/tree/main/CCS/MRR/RadarCubeCapture) |
| 2 | Point the radar |  |
| 3 | Follow the instruction in the readme | [Link](https://github.com/mohandas98/UavRadarProject_MA/blob/main/Matlab/RadarCubeCapture/Readme.txt) |