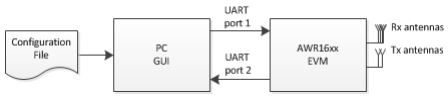
Millimeter Wave (mmw) Demo for XWR16XX

Introduction



The millimeter wave demo shows some of the capabilities of the XWR16xx SoC using the drivers in the mmWave SDK (Software Development Kit). It allows user to specify the chirping profile and displays the detected objects and other information in real-time.

Following is a high level description of the features of this demo:

- Be able to specify desired chirping profile through command line interface (CLI) on a UART port or through the TI Gallery App - **mmWave Demo Visualizer** - that allows user to provide a variety of profile configurations via the UART input port and displays the streamed detected output from another UART port in real-time, as seen in picture above.
- Some sample profile configurations have been provided in the demo directory that can be used with CLI directly or via **mmWave Demo Visualizer**:

```
mmw/profiles/profile_2d.cfg
mmw/profiles/profile_2d_srr.cfg
mmw/profiles/profile_heat_map.cfg
```

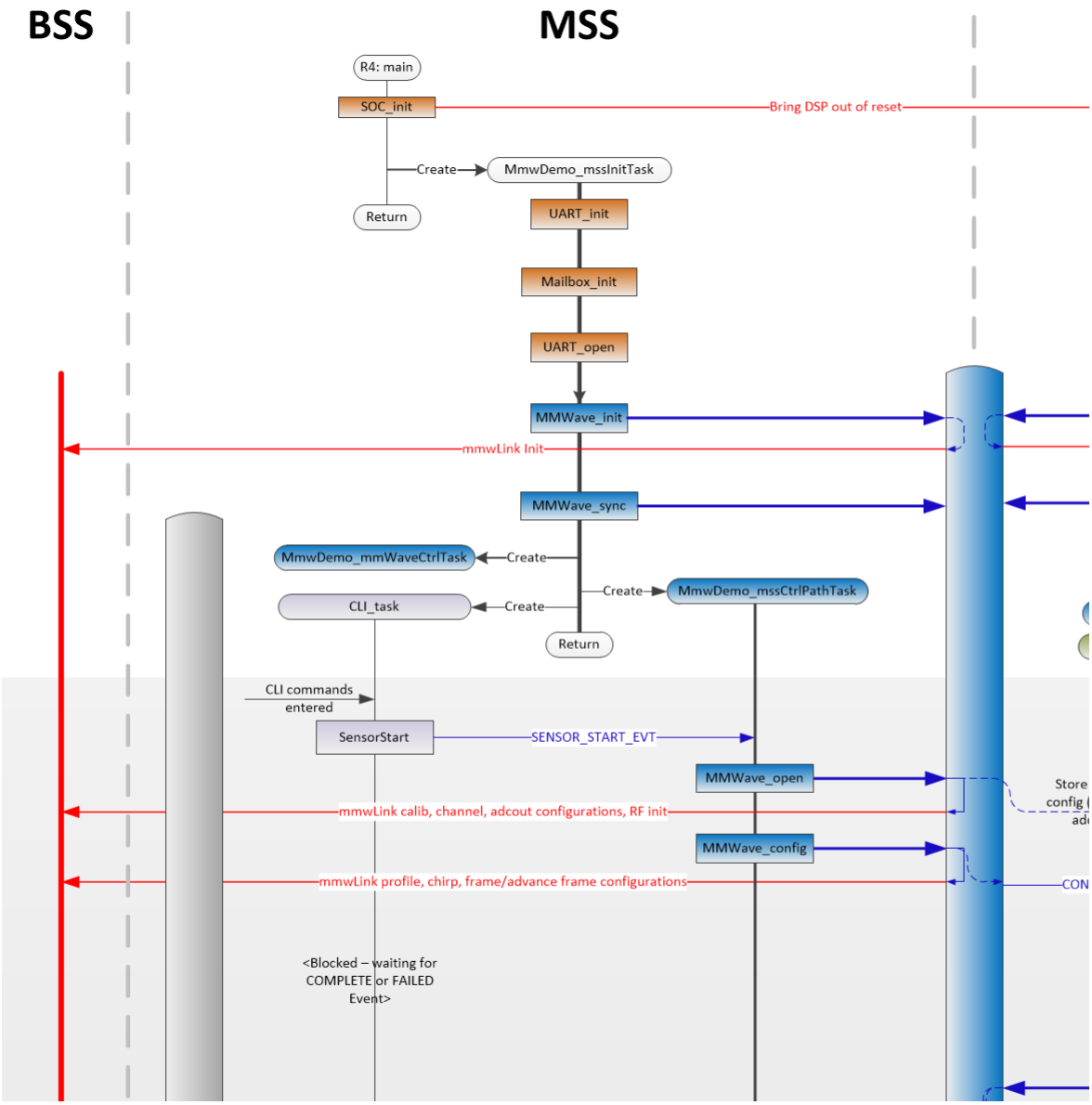
- Do 1D, 2D, CFAR and Azimuth processing and stream out velocity and two spatial coordinates (x,y) of the detected objects in real-time. The demo can also be configured to do 2D only detection (velocity and x,y coordinates).
- Various display options besides object detection like azimuth heat map and Doppler-range heat map.
- Illustrates how to configure the various hardware entities (EDMA, UART) in the AR SoC using the driver software.

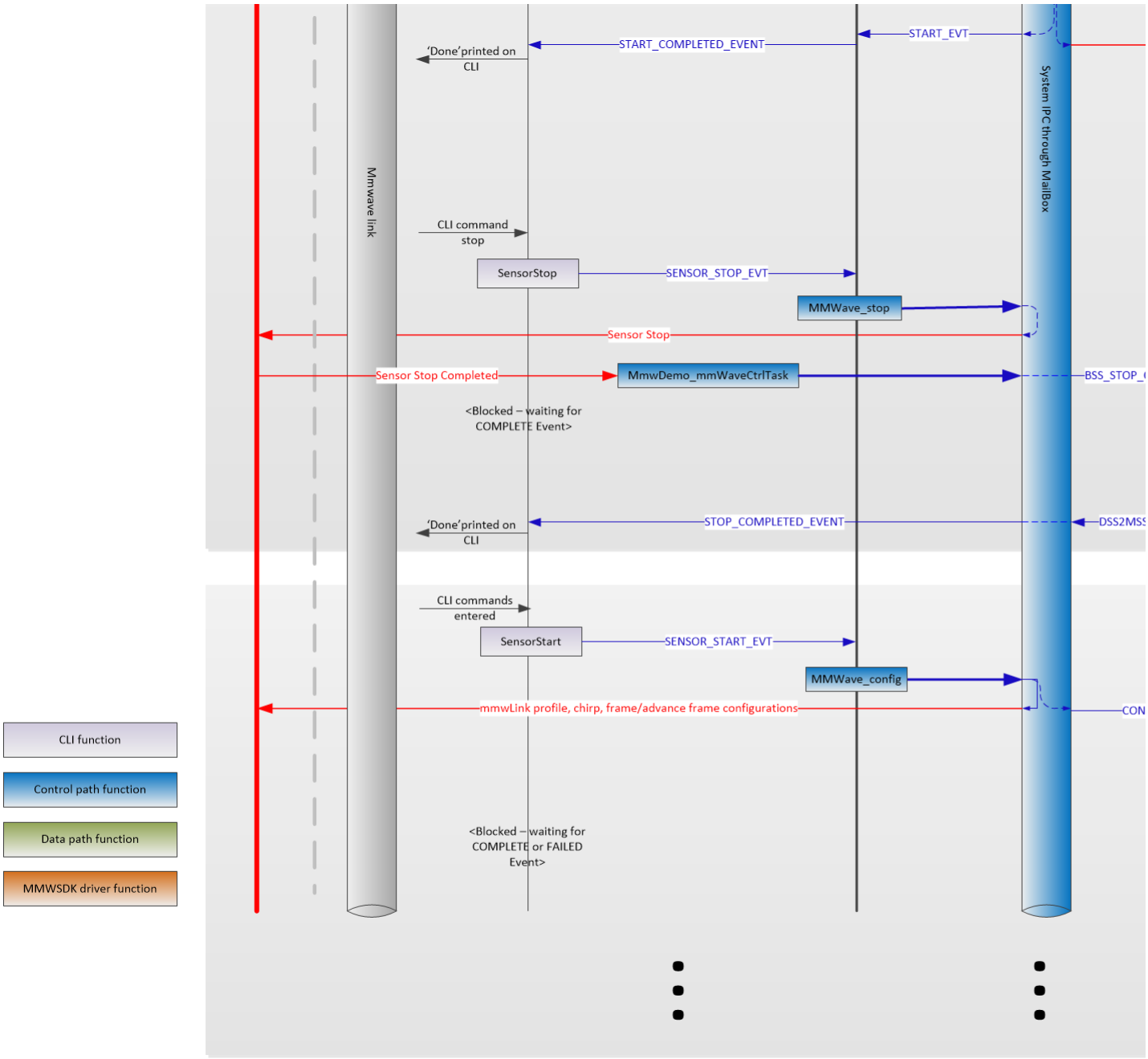
Limitations

- Because of UART speed limit (< 1 Mbps), the frame time is more restrictive. For example, for the azimuth and Doppler heat maps for 256 FFT range and 16 point FFT Doppler, it takes about 200 ms to transmit.
- Present implementation in this demo can resolve up to two objects in the azimuth dimension which have the same range and same velocity.
- Code will give an error if the requested memory in L3 RAM exceeds its size (**SOC_XWR16XX_DSS_L3RAM_SIZE**) due to particular combination of CLI configuration parameters.
- For most boards, a range bias of few centimeters has been observed. User can estimate the range bias on their board and correct using the calibration procedure described in **Range Bias and Rx Channel Gain/Offset Measurement and Compensation**.

System Details

The millimeter wave demo runs on both R4F (MSS) and C674x (DSS). System startup is described in the following diagram:





System Startup

Software Tasks on MSS

The following (SYSBIOS) tasks are running on MSS:

- **MmwDemo_mssInitTask**. This task is created and launched by **main** and is a one-time initialization task that performs the following sequence:
  1. Initializes drivers (<driver>_init).
  2. Initializes the MMWave module (MMWave_init)
  3. Creates/launches the following tasks (the **CLI_task** is launched indirectly by calling **CLI_open**).
- **MmwDemo_mmWaveCtrlTask**. This task is used to provide an execution context for the mmWave control, it calls in an endless loop the MMWave_execute API.
- **MmwDemo_mssCtrlPathTask**. The task is used to process data path events coming from the **CLI_task** or start/stop events coming from **SOC_XWR16XX_GPIO_1** button on EVM. It signals the start/stop completion events back to **CLI_task**.
- **CLI_task**. This CLI task takes user commands and posts events to the **MmwDemo_mssCtrlPathTask**. In case of start/stop commands, it waits for the completion events from **MmwDemo_mssCtrlPathTask** and on success, it toggles the LED **SOC_XWR16XX_GPIO_2**.
- **MmwDemo_mboxReadTask**. This task handles mailbox messages received from DSS.

Software Tasks on DSS

The following four (SYSBIOS) tasks are running on DSS:

- **MmwDemo_dssInitTask**. This task is created/launched by main (in **dss_main.c**) and is a one-time active task that performs the following sequence:
  1. Initializes drivers (<driver>_init).
  2. Initializes the MMWave module (MMWave_init)
  3. Creates/launches the other three tasks.
- **MmwDemo_mboxReadTask**. This task handles mailbox messages received from MSS.
- **MmwDemo_dssMMWaveCtrlTask**. This task is used to provide an execution context for the mmWave control, it calls in an endless loop the MMWave_execute API.
- **MmwDemo_dssDataPathTask**. The task performs in real-time:
  - Data path processing chain control and (re-)configuration of the hardware entities involved in the processing chain, namely EDMA.
  - Data path signal processing such as range, Doppler and azimuth DFT, object detection, and direction of arrival calculation.
  - Transfers detected objects to the HS-RAM shared memory and informs MSS that the data is ready to be sent out to through the UART output port. For format of the data on UART output port, see **MmwDemo_dssSendProcessOutputToMSS**. The UART transmission is done on MSS.

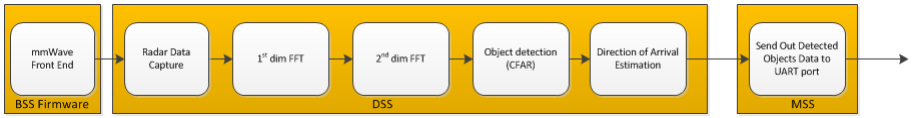
  The task pends on the following events:
  - **MMWDEMO_CONFIG_EVT**. This event is posted by **MmwDemo_dssMmwaveConfigCallbackFxn** which in called by MMWAVE_ API triggered when MSS issues MMWAVE_config
  - **MMWDEMO_BSS_STOP_COMPLETE_EVT**. This event is posted by **MmwDemo_dssMmwaveEventCallbackFxn** when the Frame Stop asynchronous event is received. Details as follows: Initially MSS

receives the CLI sensorStop command and issues a stop command to BSS. Once BSS fully stops the frame processing, BSS sends a "Frame Stopped Asynchronous event" to MSS. MSS then forwards the "Frame Stopped Asynchronous event" to DSS, where it is handled by **MmwDemo_dssMmwaveEventCallbackFxn**. This event causes DSS to go in **MmwDemo_DSS_STATE_STOP_PENDING** state. If DSS received the "Frame Stopped Asynchronous event" after the inter-frame processing is completed, it will post MMWDEMO_STOP_COMPLETE_EVT. Otherwise, it will wait to finish the inter-frame processing before posting MMWDEMO_STOP_COMPLETE_EVT. See MMWDEMO_STOP_COMPLETE_EVT event below for more details.

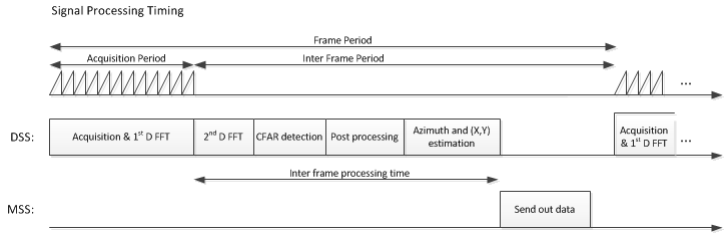
- **MMWDEMO_FRAMESTART_EVT**. This event originates from BSS firmware and indicates the beginning of the radar frame. It is posted by interrupt handler function **MmwDemo_dssFrameStartIntHandler**.
- **MMWDEMO_CHIRP_EVT**. This event originates from BSS firmware and indicates that the ADC buffer, (ping or pong) is filled with ADC samples. It is posted by **MmwDemo_dssChirpIntHandler**.
- **MMWDEMO_START_EVT**. This event is posted by **MmwDemo_dssMmwaveStartCallbackFxn** when MMWave_start is called from MSS (on CLI sensorStart 0 command which means starts with no reconfiguration)
- **MMWDEMO_STOP_COMPLETE_EVT**. This event is posted either when the MMWDEMO_BSS_STOP_COMPLETE_EVT is received after the inter-frame processing has ended or when the ongoing active frame finishes sending data over UART. This event now moves the DSS to MmwDemo_DSS_STATE_STOP state and executes **MmwDemo_dssDataPathStop** which further sends **MMWDEMO_DSS2MSS_STOPDONE** message back to MSS.

Data Path

Data Path - Overall



Top Level Data Path Processing Chain

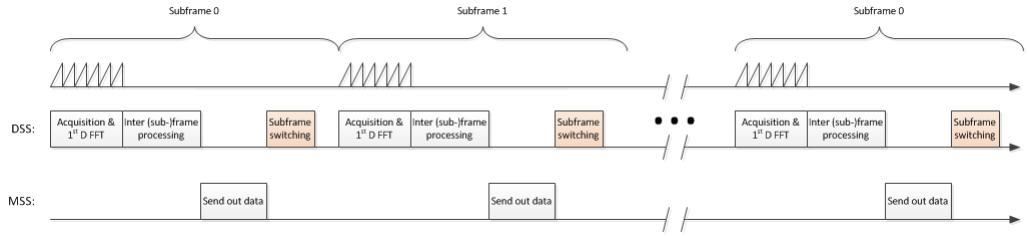


Top Level Data Path Timing

As seen in the above picture, the data path processing consists of:

- Processing during the chirps as seen in the timing diagram. This consists of
  - 1D (range) FFT processing performed by C674x that takes input from multiple receive antennas from the ADC buffer for every some number of chirps (corresponding to the chirping pattern on the transmit antennas), and
  - transferring output into the L3 RAM by EDMA. More details can be seen in **Data Path - 1st Dimension FFT Processing**
- Processing during the time between the end of chirps until the beginning of the next chirping period, shown as "Inter Frame Period" in the timing diagram. This processing consists of:
  - 2D (velocity) FFT processing performed by C674x that reads input from 1D output in L3 RAM in a transposed manner (using EDMA) and performs FFT to give a (range,velocity) matrix in the L3 RAM. The processing also includes the CFAR detection in Doppler direction. More details can be seen in **Data Path - 2nd Dimension FFT Processing**.
  - CFAR detection in range direction using mmWave library.
  - Peak Grouping if enabled.
  - Direction of Arrival (Azimuth) Estimation. More details can be seen at **Data Path - Direction of Arrival FFT Calculation** and **Data Path - (X,Y) Estimation**

Advanced Frame - Sub Frame Processing

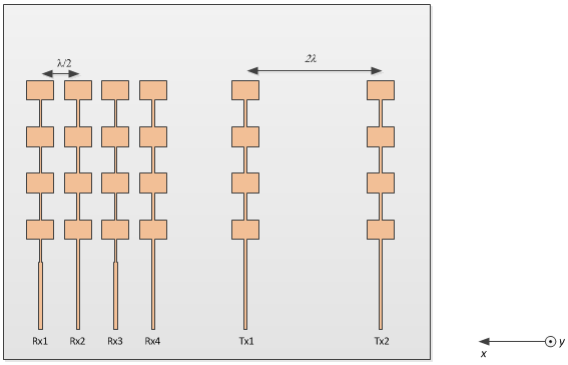


Top Level Data Path Timing for sub-frames

In advanced frame mode, sub-frame processing is supported for which the datapath is essentially same as described in **Data Path - Overall** except that there is sub-frame switching related processing required to prepare for next sub-frame as seen in above diagram. The details of what is involved in sub-frame switching are described in **Sub-frame Switching for Advanced Frame**. The data path for each sub-frame is independent of other sub-frames i.e there is no combining of information across sub-frames - each sub-frame's results are sent out to the host after the completion of its data path processing in real-time.

Antenna Configurations

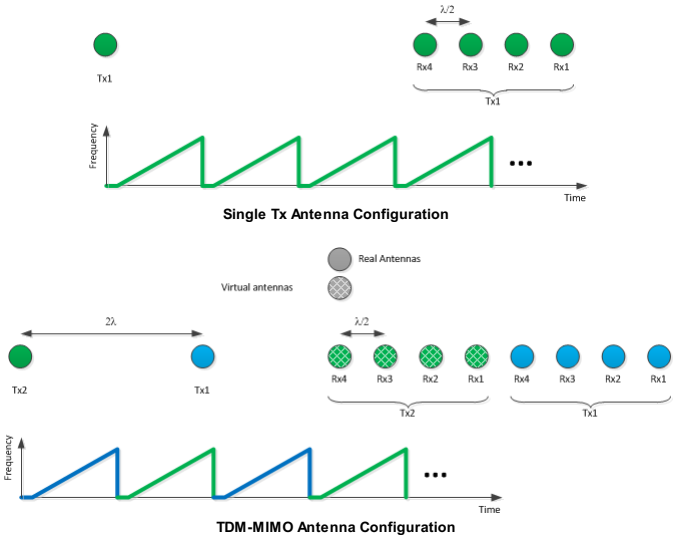
The following figure shows antenna layout as seen from the front of the EVM xWR16xx board alongside the x,y coordinate convention.



AWR16xx Antenna layout

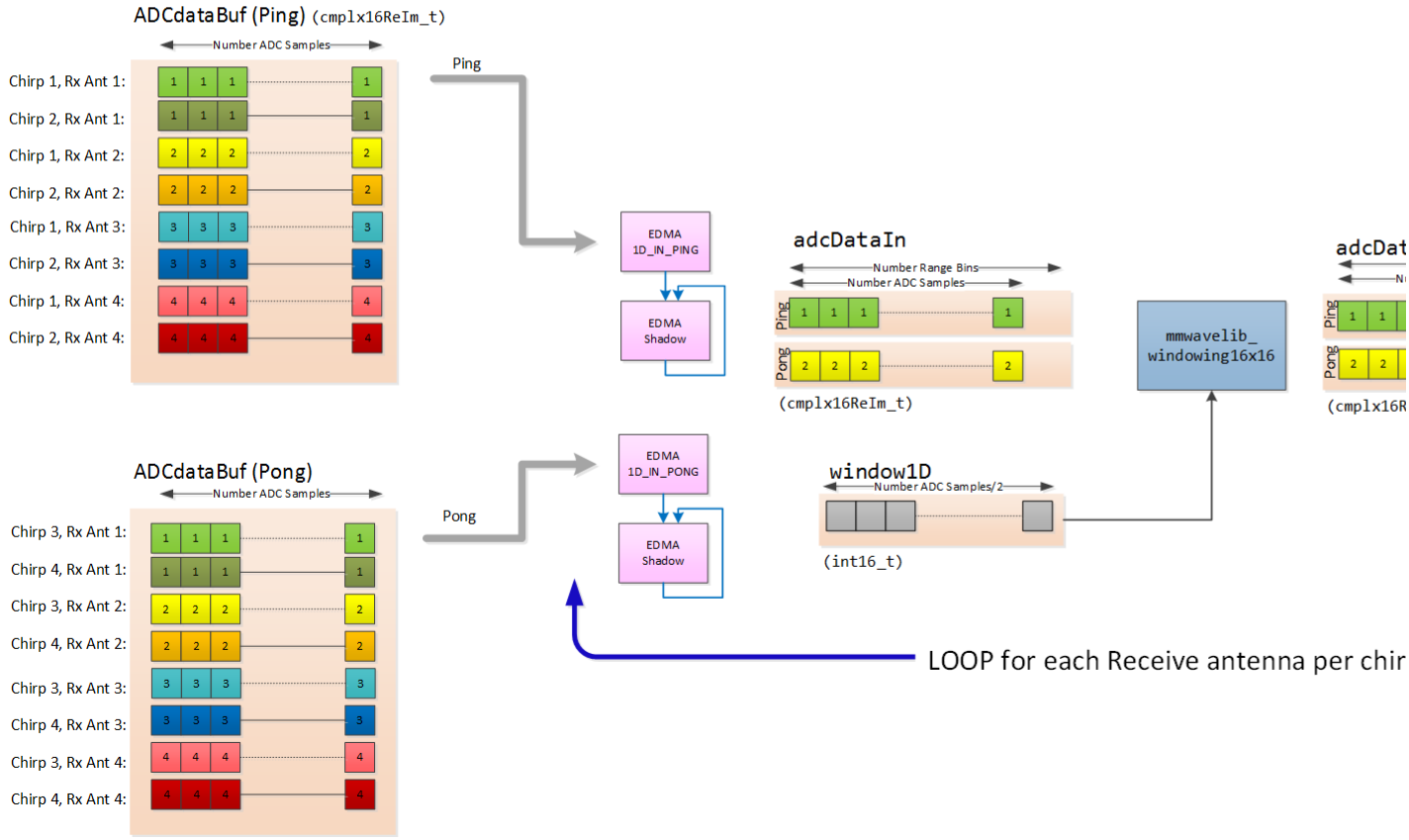
As seen in figures below, the millimeter wave demo supports two antenna configurations:

- Single transmit antenna and four receive antennas.
- Two transmit antennas and four receive antennas. Transmit antennas Tx1 and Tx2 are horizontally spaced at d = 2 Lambda, with their transmissions interleaved in a frame



Both configurations allow for azimuth estimation.

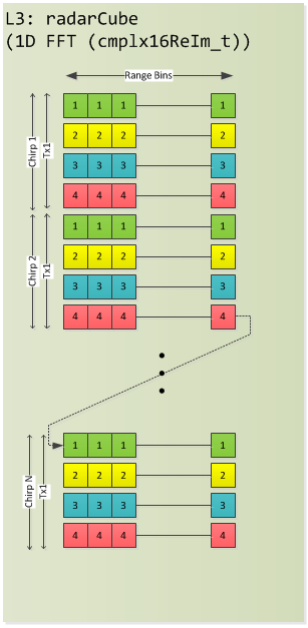
Data Path - 1st Dimension FFT Processing



Data Path 1D for TDM-MIMO configuration

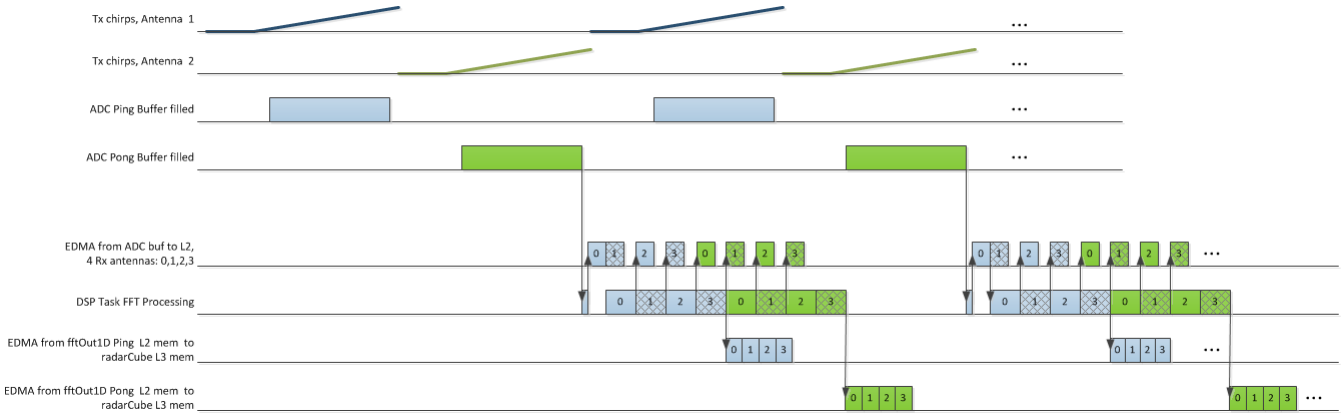
Above picture illustrates 1D chirp processing for the case with one chirp (interrupt) event for every two chirps and two transmit antennas, (TDM-MIMO case), as mentioned in **Antenna Configurations**. There are 4 rx antennas, the samples of which are color-coded and labeled as 1,2,3,4 with unique coloring for each of chirps that are processed in ping-pong manner. The 1D FFT chirp processing is triggered by hardware chirp event generated when the ADC has samples to process in the ADC buffer Ping or Pong memories. The hardware event triggers the registered chirp event interrupt handler function **MmwDemo_dssChirpIntHandler**, that in turn

posts **MMWDEMO_CHIRP_EVT** to **MmwDemo_dssDataPathTask**. The task initiates EDMA transfer of rx antenna samples in a ping pong manner to parallelize C674x processing with EDMA data transfer from ADC buffer to L2 memory. The processing includes FFT calculation using DSP library function with 16-bit input and output precision. Before FFT calculation, a Blackman window is applied to ADC samples using mmwlib library function. The calculated 1D FFT samples are EDMA transferred to the radar cube matrix in L3 memory. One column of the radar cube matrix contains 1D-FFT samples of chirps corresponding to the two transmit antennas and in this (TDM) case, all chirps corresponding to Tx1 are stored consecutively followed by those corresponding to Tx2. The reason for storing in this way instead of time of arrival order (Tx1,Tx2,Tx1,Tx2..) is to prevent EDMA jump for 1D output and 2D input from exceeding the EDMA jump limit. The EDMA jumps (source and destination B/C indices) are 16-bit signed, so when number of range bins is 1024 and number of receive antennas is 4, the jump becomes 1024*4*4(bytes/sample)*2(Tx1,Tx2 order) = 32768 which is -32768 as signed 16-bit. While the jump in 1D output can be overcome by setting the destination address in the compute loop every chirp output EDMA trigger (which is not too significant burden in cycles), the 2D cannot be overcome this way without breaking the very purpose of EDMA-CPU parallelism because the source address would have to be reprogrammed every sample! The jump is halved (16384) when storing all Tx1 consecutively followed by all Tx2 consecutively. Picture below illustrates the shape of radar cube matrix for one Tx antenna configuration, where one column contains 1D FFT samples of one chirp.



Radar cube matrix for single Tx antenna configuration

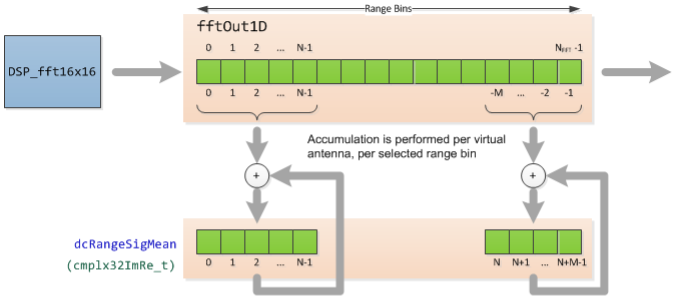
The timing diagram of chirp processing is illustrated in figure below.



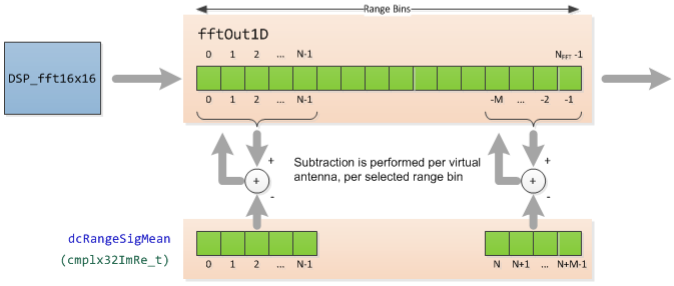
Data Path 1D timing diagram

Antenna coupling signature removal

During measurement:



After measurement:



Antenna coupling signature removal

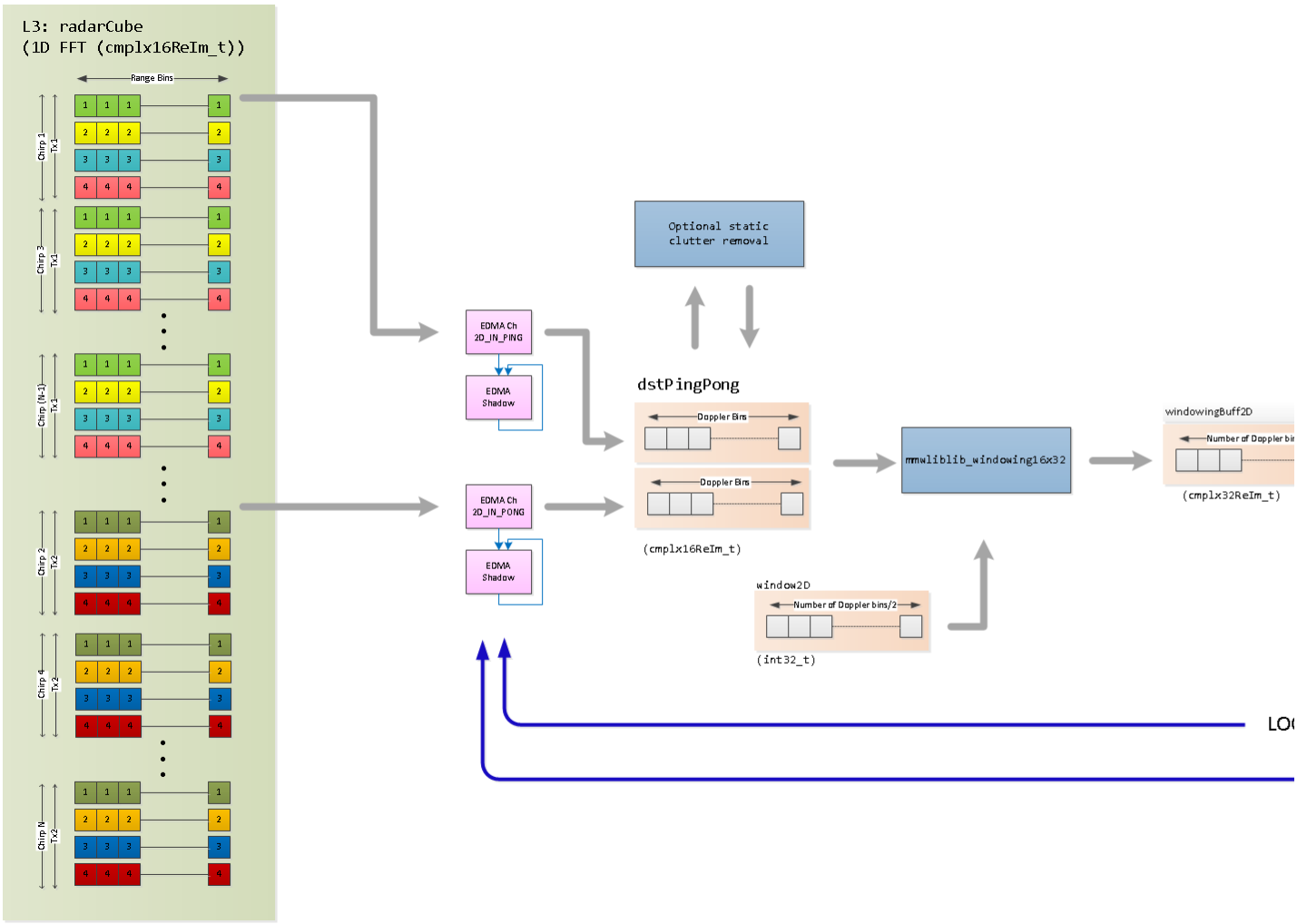
Antenna coupling signature dominates the range bins close to the radar. These are the bins in the range FFT output located around DC. This feature is under user control in terms of enable/disable and start/end range bins through a CLI command called calibDcRangeSig. During measurement (when the CLI command is issued with feature enabled), each of the specified range bins for each of the virtual antennas are accumulated over the specified number of chirps and at the end of the period, the average is computed for each bin/antenna combination for removal after the measurement period is over. Note that the number of chirps to average must be power of 2. It is assumed that no objects are present in the vicinity of the radar during this measurement period. After measurement is done, the removal starts for all subsequent frames during which each of the bin/antenna average estimate is subtracted from the corresponding received samples in real-time for subsequent processing.

Data Path - 2nd Dimension FFT Processing

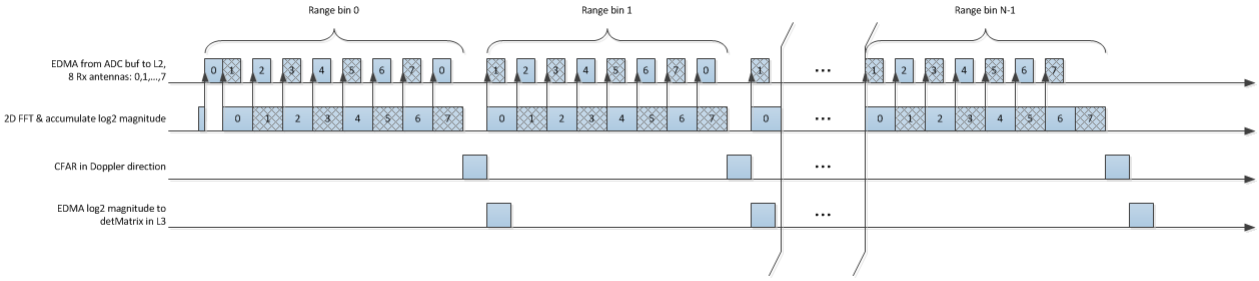
The 2D processing consists of the following steps:

1. For each range bin it performs:
    - Static clutter removal if enabled. The mean value of the input samples to the 2D-FFT is subtracted from the samples,
    - Windowing - samples are multiplied by a window function,
    - 2D-FFT on the samples of 1D-FFT output across chirps (samples are transposed by the EDMA before 2D FFT can be performed),
    - log2 magnitude of the output,
    - accumulation across all Rx antennas,
    - transfer of accumulated values to detection matrix in L3 using EDMA,
    - CFAR pre-detection in Doppler direction and saving of Doppler indices of detected objects for the final CFAR detection in the range direction.
2. Final CFAR detection in range direction at Doppler indices at which objects were detected in previous step
3. Peak grouping. Grouping options are specified by CLI CFAR configuration function and can be
    - in both range and Doppler direction,
    - only in range,
    - only in Doppler direction, or
    - none.

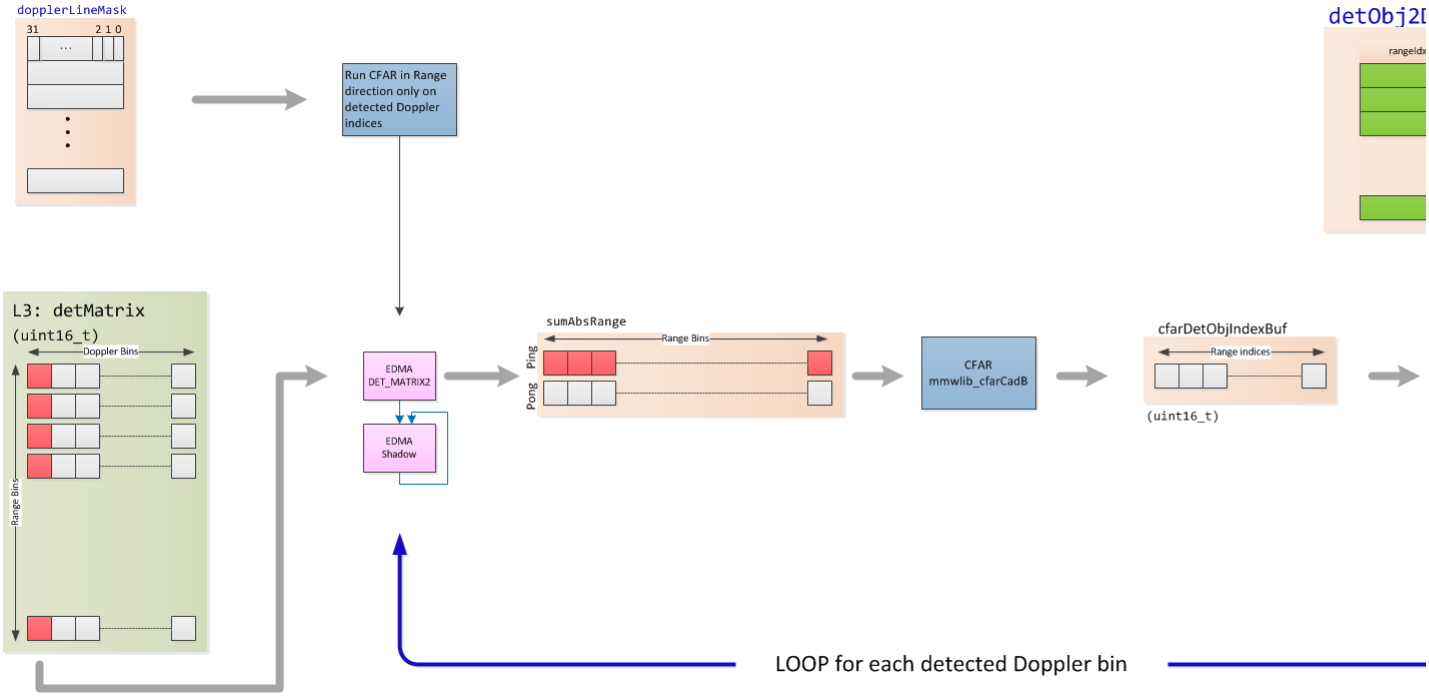
    The 2D processing is shown in figures below.



2D-FFT Processing - Calculation of Detection Matrix and CFAR in Doppler direction



2D-FFT Processing - Calculation of Detection Matrix and CFAR in Doppler direction - timing diagram



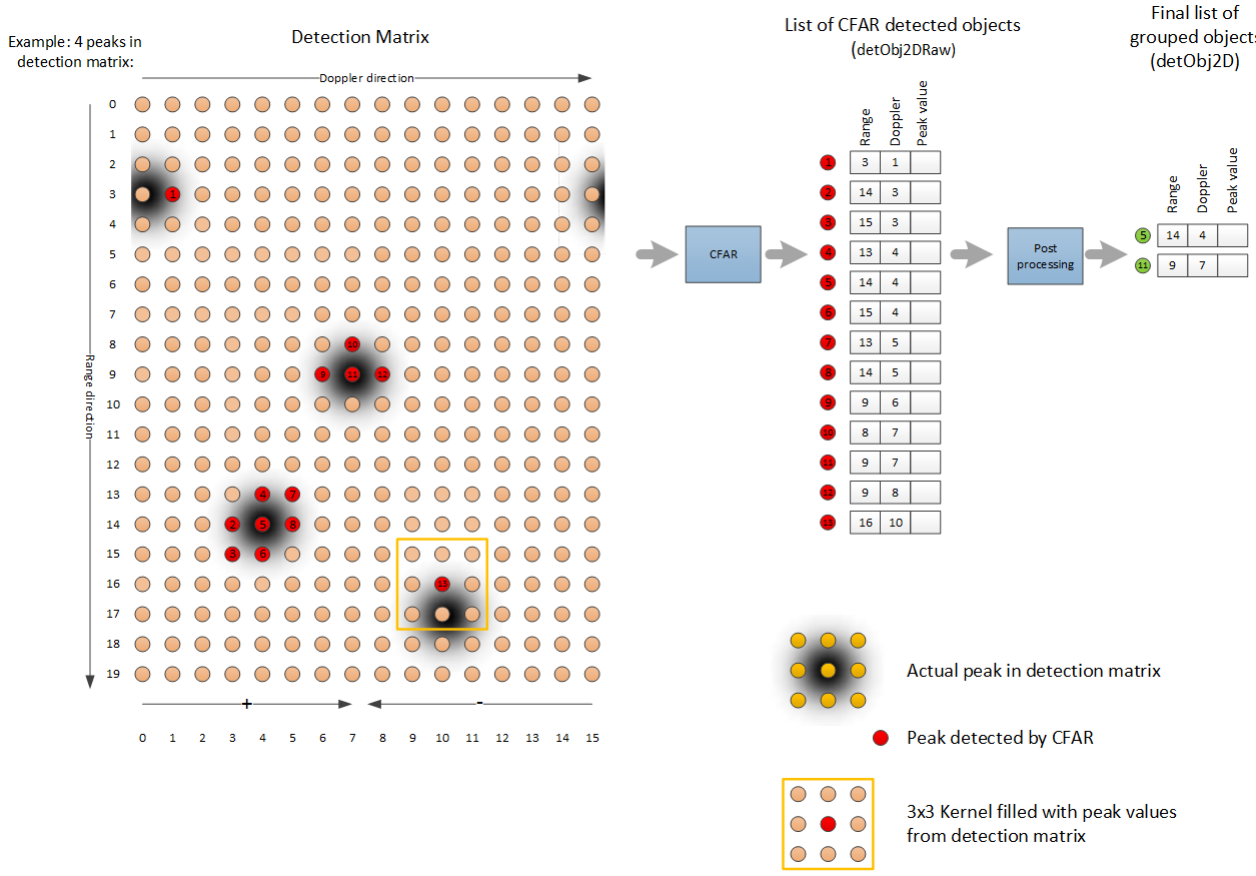
CFAR in range direction and peak grouping

Peak grouping

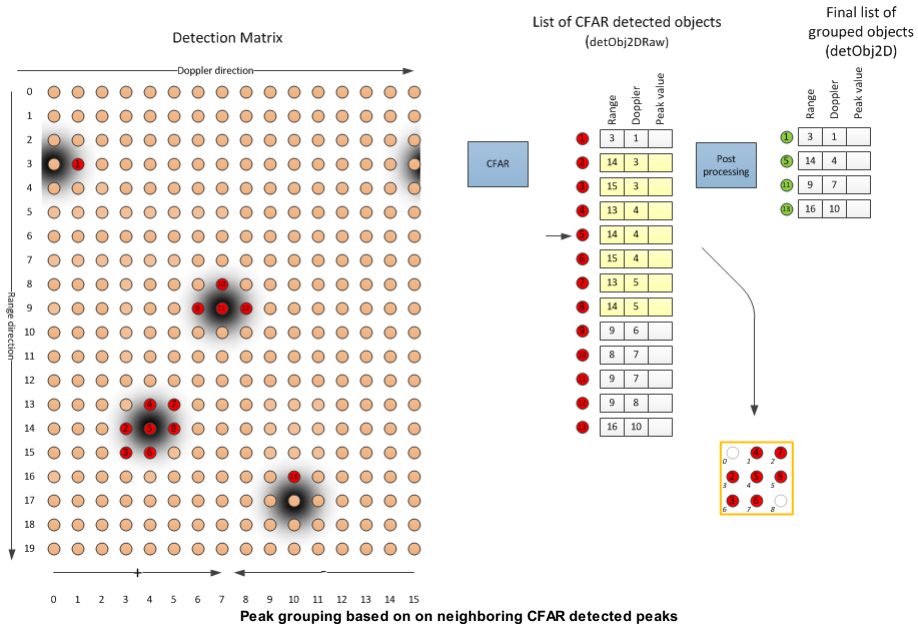
Two peak grouping schemes are implemented:

1. Peak grouping based on peaks of the neighboring bins read from detection matrix. For each CFAR detected peak, listed in **MmwDemo_DSS_DataPathObj::detObj2DRaw**, it checks if the peak is greater than its neighbors. If this is true, the peak is copied to the output list of detected objects **MmwDemo_DSS_DataPathObj::detObj2D**. The neighboring peaks that are used for checking are taken from the detection matrix **MmwDemo_DSS_DataPathObj::detMatrix** and are copied into 3x3 kernel regardless of whether they are CFAR detected or not.
2. Peak grouping based on peaks of neighboring bins that are CFAR detected. For each detected peak the function checks if the peak is greater than its neighbors. If this is true, the peak is copied to the output list of detected objects. The neighboring peaks that are used for checking are taken from the list of CFAR detected objects, (not from the detection matrix), and are copied into 3x3 kernel that has been initialized to zero for each peak under test. If the neighboring peak has not been detected by CFAR, it is not copied into the kernel.

Peak grouping schemes are illustrated in two figures below. The first figure, illustrating the first scheme, shows how the two targets (out of four) can be discarded and not presented to the output. For these two targets (at range indices 3 and 17 in figure below) the CFAR detector did not detect the highest peak of the target, but only some on the side, and these side peaks are discarded. The second figure, illustrating the second scheme, shows that all four targets are presented to the output, one peak per target, with the targets at range indices 3 and 17 represented with side peaks.



Peak grouping based on neighboring peaks from detection matrix

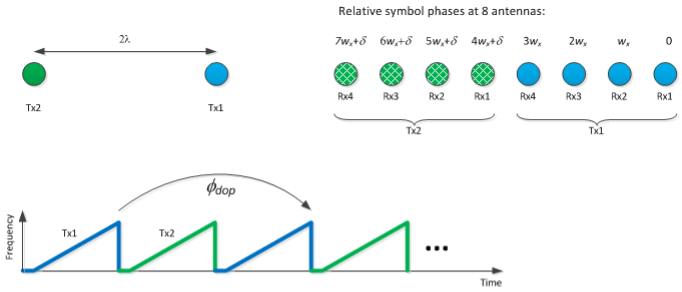


Peak grouping based on on neighboring CFAR detected peaks

Data Path - Direction of Arrival FFT Calculation

Because L3 memory is limited in size, the radar cube matrix stores only the 1D-FFT in 16-bit precision. Because of this, azimuth FFT calculation requires repeated 2D FFT calculation. Since for each detected object, we need 2D FFT at a single bin, instead of recalculating 2D-FFT, we calculate single point DFT at the bin index of each detected object. This calculation is repeated for each received antenna.

Compensation for the Doppler phase shift in the angle estimation is performed on the virtual antennas (symbols corresponding to the second Tx antenna in case of TDM-MIMO configuration). These symbols are rotated by half of the estimated Doppler phase shift between subsequent chirps from the same Tx antenna. The Doppler shift is calculated using the lookup table **MmwDemo_DSS_DataPathObj::azimuthModCoefs** Refer to the pictures below.



$\delta = \phi_{dop} / 2$ – Doppler induced phase shift between two sets of Rx antennas

ϕ_{dop} - Doppler induced phase shift between subsequent chirps of the same Tx antenna

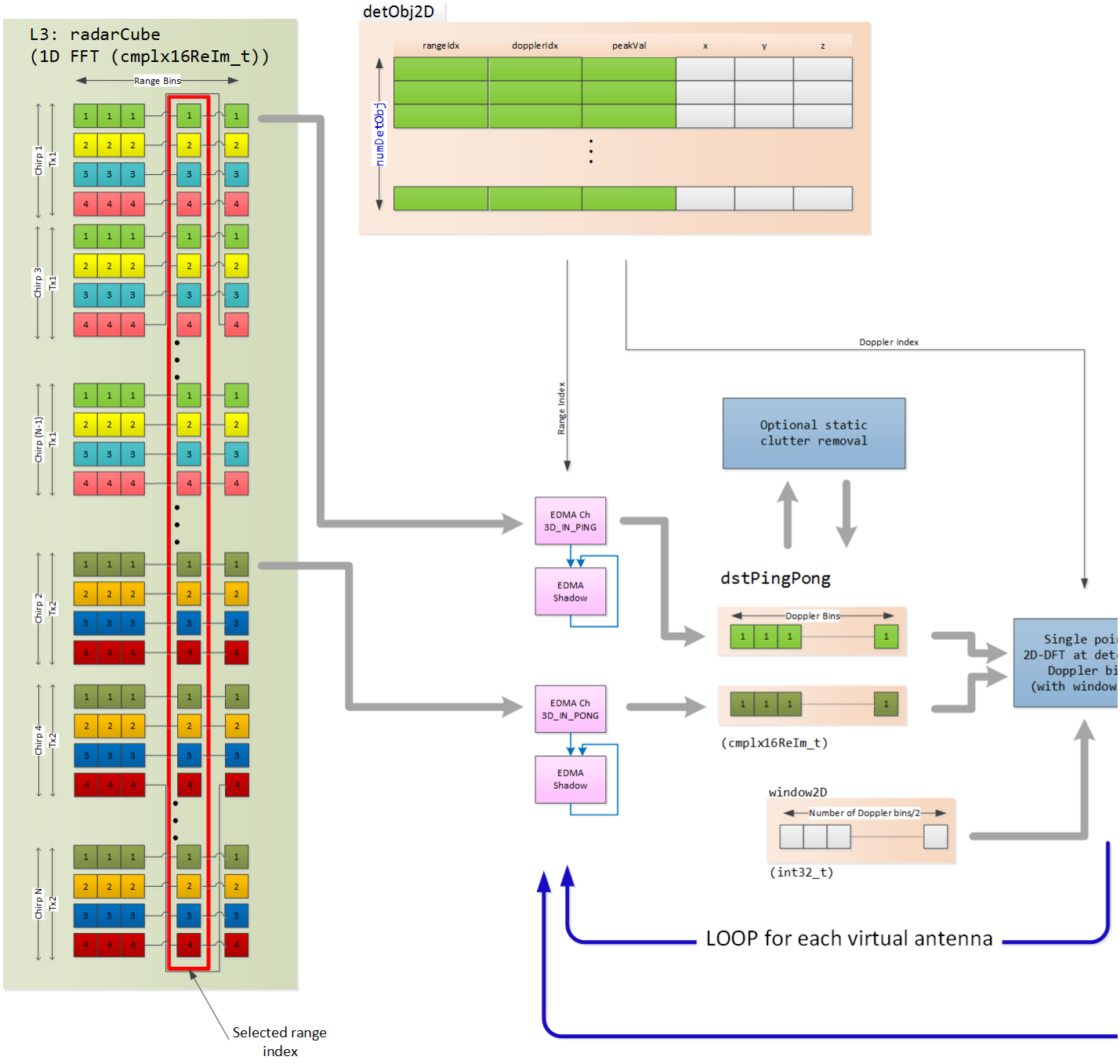
$\phi_{dop} = 2\pi\, i_{dop} / N_{FFT\_DOP}$, *idop* is detected Doppler bin index, $-N_{FFT\_DOP}/2 < i_{dop} < N_{FFT\_DOP}/2-1$

Doppler compensation:

$$X'(m,k) = X(m,k)e^{-jm\delta}, m = 1,..., N_{Tx} - 1, k = 0,..., N_{Rx} - 1$$

Figure_doppler: Doppler Compensation

Currently the size of Azimuth FFT is hardcoded and defined by **MMW_NUM_ANGLE_BINS**. The FFT is calculated using DSP lib function DSP_fft32x32. The output of the function is magnitude squared and the values are stored in floating point format.



Direction of arrival calculation

Data Path - (X,Y) Estimation

The (x,y) estimation is calculated in the function **MmwDemo_XYestimation**.

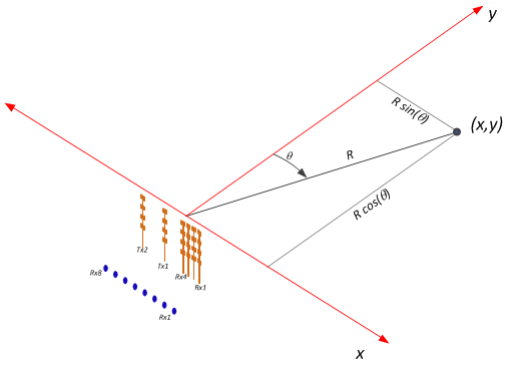


Figure A: Coordinate Geometry

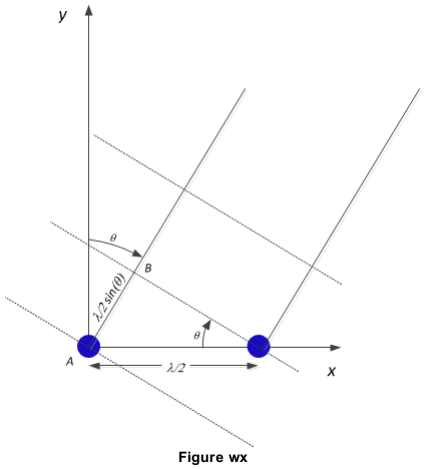


Figure wx

Figure_geometry shows orientation of x,y axes with respect to the sensor/antenna positions. The objective is to estimate the (x,y) coordinates of each detected object. w_x is the phase difference between consecutive receive azimuth antennas of the 2D FFT. The phases for each antenna are shown in the Figure_doppler. Figure_wx shows that the distance AB which represents the relative distance between wavefronts intersecting consecutive azimuth antennas is $AB = \frac{\lambda}{2}\sin(\theta)$. Therefore $w_x = \frac{2\pi}{\lambda} \cdot AB$, and therefore $w_x = \pi \sin(\theta)$. Note that the phase of the left-ward antenna is advanced compared to the right-ward antenna and antenna indices increment from right to left (which is the order in ADCbuf and all of processing) so phase increments as $+w_x$. For a single obstacle, the signal at the 8 azimuth antennas will be (A_1 and ψ are the arbitrary starting amplitude/phase at the first antenna):

$$A_1 e^{j\psi}[1 \; e^{jw_x} \; e^{j2w_x} \; e^{j3w_x} \; e^{j4w_x} \; e^{j5w_x} \; e^{j6w_x} \; e^{j7w_x}]$$

An FFT of the above signal will yield a peak at w_x . If k_{MAX} is the index of the peak in log magnitude FFT represented as signed index in range $[-\frac{N}{2}, \frac{N}{2} - 1]$, then w_x will be

$$w_x = \frac{2\pi}{N} k_{MAX}$$

Calculate range (in meters) as:

$$R = k_r \frac{c \cdot F_{SAMP}}{2 \cdot S \cdot N_{FFT}}$$

where, c is the speed of light (m/sec), k_r is range index, F_{SAMP} is the sampling frequency (Hz), S is chirp slope (Hz/sec), N_{FFT} is 1D FFT size. Based on above calculations of R and w_x, the (x, y) position of the object can be calculated as seen in the Figure_geometry,

$$x = R\sin(\theta) = R\frac{w_x}{\pi}, y = \sqrt{R^2 - x^2}$$

The computed (x, y) and azimuth peak for each object are populated in their respective positions in MmwDemo_DSS_DataPathObj::detObj2D. Note the azimuth peak (magnitude squared) replaces the previous CFAR peak (sum of log magnitudes) in the structure. To be able to detect two objects at the same range-doppler index but at different angle, search for the 2nd peak in the azimuth FFT and compare its height relative to the first peak height, and if detected, create new object in the list with the same range/Doppler indices, and repeat above steps to calculate (x,y) coordinates. To enable/disable the two peak detection or to change the threshold for detection, refer to MMWDEMO_AZIMUTH_TWO_PEAK_DETECTION_ENABLE and MMWDEMO_AZIMUTH_TWO_PEAK_THRESHOLD_SCALE.

Velocity disambiguation

A simple technique for velocity disambiguation is implemented. It corrects target velocities up to $2v_{max}$, and allows for correct calculation of X/Y coordinates for target velocities even greater than $2v_{max}$. The technique consists of the following steps applied after the CFAR detection phase. For each detected point, assuming doppler correction of virtual antennas is already done:

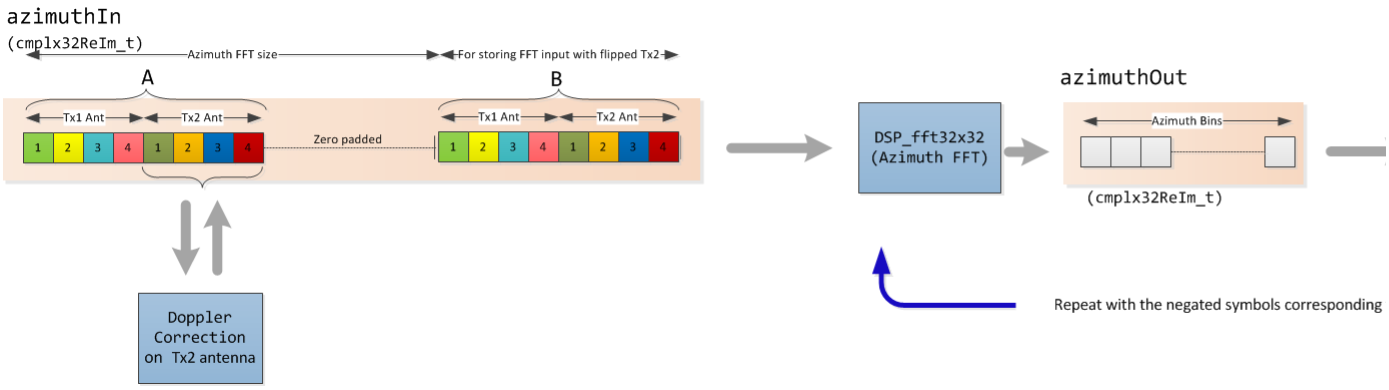
1. Copy the doppler corrected antenna data A into the upper address area (labeled B in the figure below) of the azimuthIn. This is done because the DSPlib FFT function overwrites the input buffer with reversed index and we would need to later compute the FFT on sign flipped symbols corresponding to Tx2.
2. Calculate azimuth FFT (in area A) and compute the magnitude squared of the FFT output.
3. Save result azimuthMagSqr to set 0, "uncorrected set", (see figure below).
4. Flip the signs of the symbols corresponding to Tx2 antenna transmission in area B. Copy B to A and zero pad.
5. Repeat step 4.
6. Save result azimuthMagSqr to set 1, "corrected set".
7. Search for maximum over both sets, and select the set where the maximum occurred.
8. If the maximum occurred in "corrected set", correct the estimated velocity as:

   $v_{corr} = v_{est} + 2v_{max}$ (if $v_{est} < 0$)

   $v_{corr} = v_{est} - 2v_{max}$ (if $v_{est} > 0$)

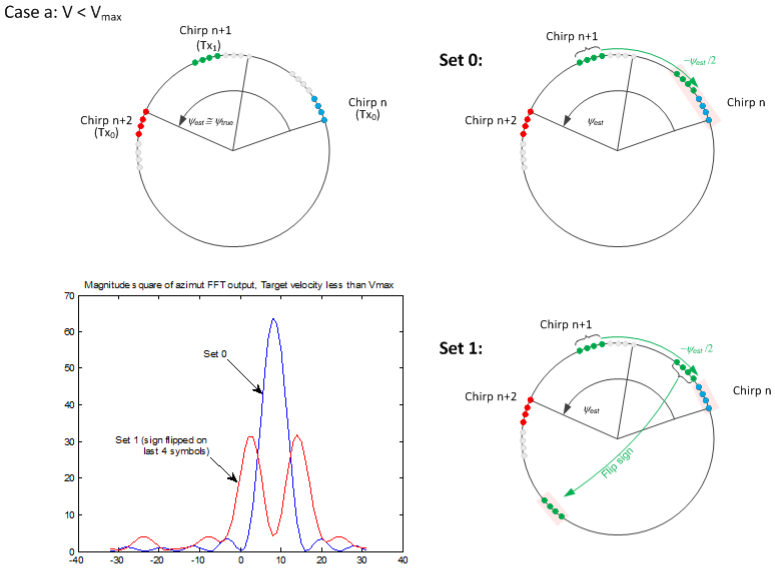
   otherwise no correction is required.

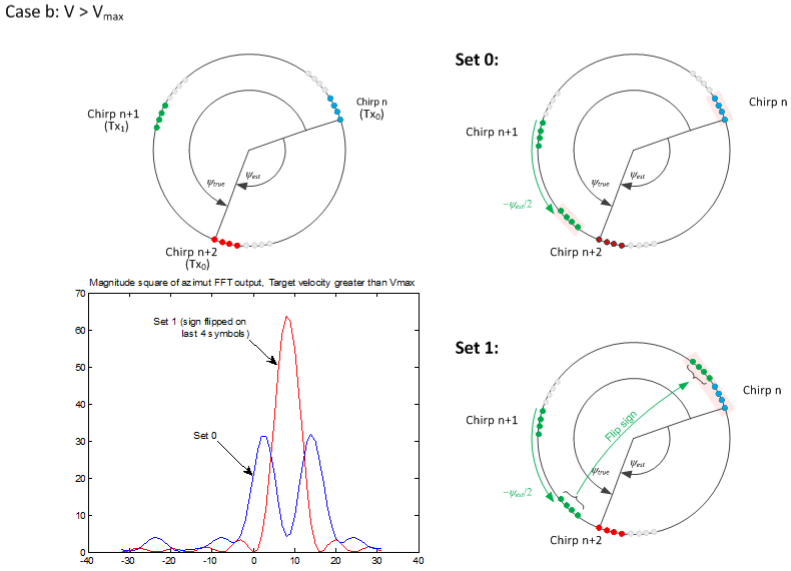
9. Calculate X/Y coordinates using index i_{max} from the selected set.



Extending maximum velocity - data path

Figure below illustrates this technique on one example with a target moving from the sensor at positive azimuth angle at a speed less than Vmax (case a), and at the speed greater than Vmax (case b). Figure shows 2D-FFT antenna symbols for 3 consecutive chirps: n, n+1 and n+2 (blue, green and red dots). The symbols of the chirp n and n+1 are used for azimuth FFT calculation. The Doppler shift, the angle ψ_{true} between chirps n and n+2, (successive chirps of the same Tx antenna), is estimated from 2D-FFT as ψ_{est}. In case a, $\psi_{true} < \pi$, and $\psi_{est} \approx \psi_{true}$. In case b, $\psi_{true} > \pi$, (Doppler velocity is aliased), and ψ_{est} is estimated as a negative value, (target approaching the sensor). Doppler compensation rotates the symbols of chirp n+1 by $-\psi_{est}/2$ to align them with the symbols of chirp n. The azimuth FFT is calculated on these symbols, and the output is placed to set 0. The symbols of the chirp n+1 are then sign flipped, and the azimuth FFT is again calculated and the output is placed to set 1. In case a, V<Vmax, the maximum peak in the FFT output in set 0 is larger than in set 1, as opposed in case b, V>Vmax, where the maximum peak in FFT output in set 1 is larger than in set 0.

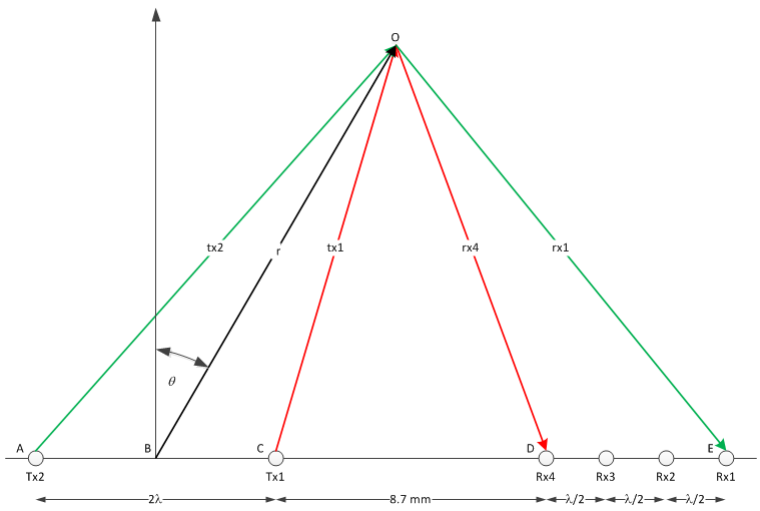




Example of target moving from sensor under angle at V<Vmax (Case a) and V>Vmax (Case b)

Near Field Correction

Problem Statement and Algorithm Description



Near Field Geometry

It was assumed in **Data Path - (X,Y) Estimation** that the object was located in the far field so that the rays between the object and the multiple TX/RX antennas are parallel. However for very close by objects this assumption (of parallel lines) is not valid as seen in the above figure and can induce a significant phase error when processed using regular FFT techniques. It can be shown that:

- The phase error is most pronounced in the phase difference between the path from Tx1(C)->object(O)->Rx4(D) (red lines) and Tx2(A)->object(O)->Rx1(E) (green lines). This corresponds to the phase difference between virtual antenna 4 and 5
- The phase error decreases as the range of the object increases.
- For a fixed range, the phase error is most severe for bore-sight objects and decreases to zero at grazing angles.

This phase error manifests in a spurious peak in the azimuth FFT which results in ghost object detection when multi object beam forming feature is enabled. In order to mitigate this error, the phase error between the physical and virtual sets of antennas needs to be corrected based on the geometry and the range at which the object has been detected. This is done in the following manner:

Let x be the 1x8 vector corresponding to the 8 virtual antennas. Let F denote the 64-point FFT of x; i.e F=fftshift (fft(x,64)); Similarly let F_1 and F_2 denote the 64-point FFT's of x(1:4) and x(5:8). Then it can be shown that:

$$F(k) = F_1(k) + F_2(k)e^{-j2\pi k4/64}; -32 \le k \le 31;$$

The above equation can be modified to incorporate the effect of the near-field induced phase error (which occurs at the boundary between the virtual antennas corresponding to TX1 and TX2). The modified equation is:

$$F(k) = F_1(k) + F_2(k)e^{-j2\pi k4/64}e^{-j\phi(k,r)}; -32 \le k \le 31;$$

where $\phi(k,r)$ is the near-field induced phase error which depends on the angle k and range r. This quantity can be computed from the geometry in the above figure as follows:

$$\triangle OBC : tx1 = \sqrt{r^2 + \overline{\mathrm{BC}}^2 - 2r\overline{\mathrm{BC}}\cos(\pi/2 - \theta)}$$

$$\triangle OBA : tx2 = \sqrt{r^2 + \overline{\mathrm{BA}}^2 - 2r\overline{\mathrm{BA}}\cos(\pi/2 + \theta)}$$

$$\triangle OBD : rx4 = \sqrt{r^2 + \overline{\mathrm{BD}}^2 - 2r\overline{\mathrm{BD}}\cos(\pi/2 - \theta)}$$

$$\triangle OBE : rx1 = \sqrt{r^2 + \overline{\mathrm{BE}}^2 - 2r\overline{\mathrm{BE}}\cos(\pi/2 - \theta)}$$

With $theta = \arcsin(2k/64)$, the above simplifies to:

$$tx1 = \sqrt{r^2 + \overline{\mathrm{BC}}^2 - rk(\overline{\mathrm{BC}}/16)}$$

$$tx2 = \sqrt{r^2 + \overline{\mathrm{BA}}^2 - rk(-\overline{\mathrm{BA}}/16)}$$

$$rx4 = \sqrt{r^2 + \overline{\mathrm{BD}}^2 - rk(\overline{\mathrm{BD}}/16)}$$

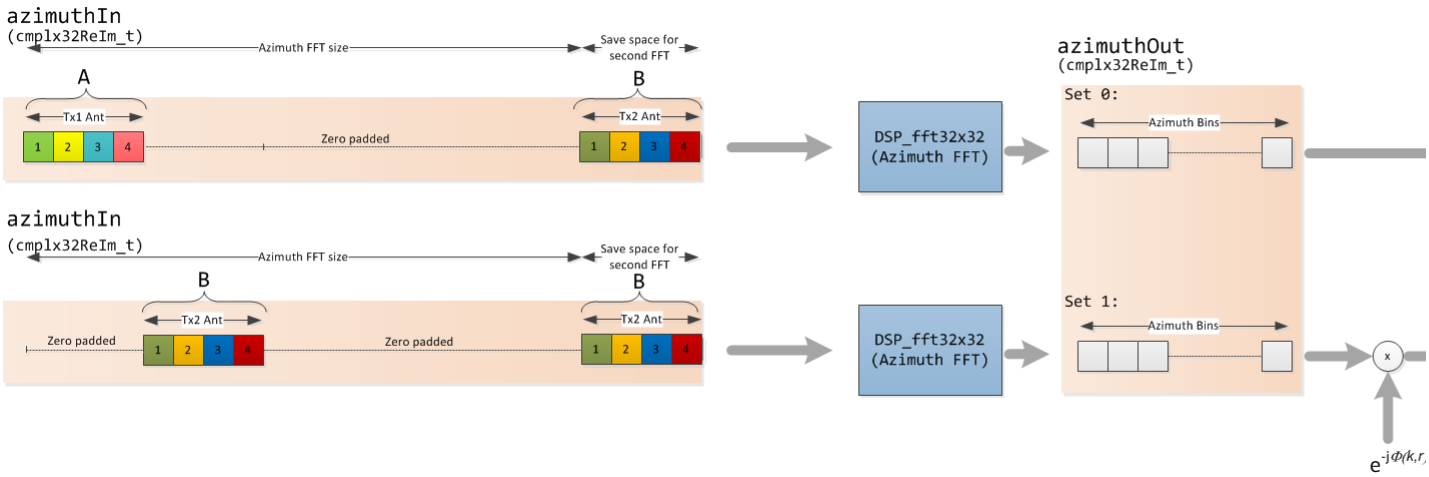
$$rx1 = \sqrt{r^2 + \overline{\mathrm{BE}}^2 - rk(\overline{\mathrm{BE}}/16)}$$

Note that $\overline{\mathrm{BC}}$, $\overline{\mathrm{BA}}$, $\overline{\mathrm{BD}}$, $\overline{\mathrm{BE}}$ are constants and therefore above calculations involve 4*(3 multiplications + 2 additions/subtractions + square root).

As mentioned in **Data Path - (X,Y) Estimation**, for far field the phase difference between consecutive antennas is $w_x = \pi\sin(\theta)$. So the phase error is calculated as:

$$\begin{aligned}\phi(k,r) &= \frac{2\pi}{\lambda}\{(tx2 + rx1) - (tx1 + rx4)\} - \pi\sin(\theta) \\ &= \frac{2\pi}{\lambda}\{(tx2 + rx1) - (tx1 + rx4)\} - \pi k/32\end{aligned}$$

Implementation Details



Near Field Implementation

Referring to the figure above, for each detected point, assuming doppler correction of virtual antennas is already done:

1. Copy the antenna data corresponding to Tx2 to the upper address area (labeled B in the figure) of the azimuthIn and zero-pad between A (data for Tx1) and B. The purpose of this step is to preserve the Tx2 symbols for the next computation. Compute the FFT of the first 64 samples of azimuthIn and store in lower address (Set 0 in the picture) of azimuthOut. This is the $F_1(k)$.
2. Copy the Tx2 data stored at the upper address from above step into the position B shown in the lower path, which is basically restoring B back to its original position. Zero-pad the first 4 antennas (corresponding to Tx1) and zero-pad the remaining FFT input data after B. Compute the FFT of the first 64 samples of azimuthIn and store in upper address (Set 1 in the picture) of azimuthOut. This will be $F_2(k)e^{-j2\pi k4/64}$.
3. Apply the correction $e^{-j\phi(k,r)}$ on Set 1 over all $k \in [-32, 31)$ and add with Set 0 in place i.e output of the sum will replace Set 0.
4. Proceed to do the rest of the azimuth calculations (magnitude square etc).

NOTE:

1. The range bias estimated from the calibration procedure is subtracted from the range at which object is detected before calculating the phase correction. The function to estimate the phase correction is **MmwDemo_nearFieldCorrection**.
2. The near field correction is currently in exclusion with velocity disambiguation because of implementation complexities and also because it is unlikely to have objects at high velocities in the near field.

User Interface

A CLI command "nearFieldCfg" has been provided with parameters to enable/disable the feature and provide start and end range index over which the feature is to be activated. The phase error is highest at boresight ($\theta = 0$). The following data at the boresight, which can be computed using the formulae in **Problem Statement and Algorithm Description**, can be used as a guidance to decide the maximum range up to which to enable the feature.

Distance (cm)	Phase Error (degrees)
5	159.8
10	82.8
20	41.8
40	20.9
80	10.5
100	8.4
200	4.2
400	2.1
1000	0.8

NOTE:

1. The range index does not exclude the range bias. E.g if the range bias is 6 cm and the range step is 2 cm, then 4 cm from the sensor will be (6+4)/2 or 5 range bins or range index of 4.
2. Because of several computations involved in the near field correction for each detected point in the near field (most notably 64*(4 square root + 1 sin + 1 cosine)), the DSP will consume non-trivial MIPS for this feature. Presently it takes about 35000 cycles per detected point in the enabled near field range which is 58 us (at 600 MHz DSP speed).
3. Because of reasons not yet fully understood, the feature does not work reliably below about 5 cm and therefore is recommended to be disabled below this range (using the start index parameter of the configuration).
4. The near field correction is not applied on azimuth-range heatmap data shipped out of the target.

Sub-frame Switching for Advanced Frame

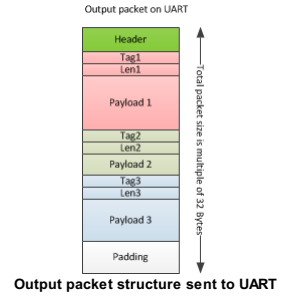
There is some compute overhead for switching sub-frames beyond the book-keeping aspects for the data path object (which are multi-instantiated). Because of the potential resource constraints of EDMA (channels, params etc), and memory limitations, some computations are redone for each sub-frame depending on its configuration. These are:

- Reconfiguration of ADCbuf driver - **MmwDemo_dssDataPathConfigAdcBuf**
- Reprogramming of the EDMA channels and Param sets - **MmwDemo_dataPathInitEdma**
- Regeneration of tables for FFT (twiddle tables), single-point DFT and FFT window - **MmwDemo_dataPathConfigFFTs**. This regeneration is particularly high in cycles if not optimized so certain optimizations were performed to efficiently generation these tables.

One can see the cycles related to switching to the next sub-frame by examining in real-time mode the element **MmwDemo_timingInfo::subFrameSwitchingCycles** in the data path object instance corresponding to the current sub-frame. Note that the buffer layout as described in section **EDMA versus Cache based Processing** is calculated during configuration time for all sub-frames corresponding to their configurations and so does not need to be part of sub-frame switching, because the memory savings related to not multiplying the buffer pointer storage for all sub-frames is modest. The above functions are called as part of a convenient reconfiguration function - **MmwDemo_dssDataPathReconfig**.

Output information sent to host

Output packets with the detection information are sent out every frame through the UART. Each packet consists of the header **MmwDemo_output_message_header_t** and the number of TLV items containing various data information with types enumerated in **MmwDemo_output_message_type_e**. The numerical values of the types can be found in **mmw_output.h**. Each TLV item consists of type, length (**MmwDemo_output_message_tl_t**) and payload information. The structure of the output packet is illustrated in the following figure. Since the length of the packet depends on the number of detected objects it can vary from frame to frame. The end of the packet is padded so that the total packet length is always multiple of 32 Bytes.



Output packet structure sent to UART

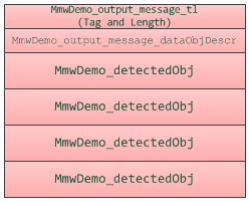
The following subsections describe the structure of each TLV.

List of detected objects

Type: (**MMWDEMO_OUTPUT_MSG_DETECTED_POINTS**)

Length: (size of **MmwDemo_output_message_dataObjDescr_t**) + (Number of detected objects) x (size of **MmwDemo_detectedObj_t**)

Value: List descriptor (**MmwDemo_output_message_dataObjDescr_t**) followed by array of detected objects. The information of each detected object is stored in structure **MmwDemo_detectedObj_t**. When the number of detected objects is zero, this TLV item is not sent. The whole detected objects TLV structure is illustrated in figure below.



Detected objects TLV

Note: If user is interested in parsing the UART stream to find range and velocity from their indices in the structure **MmwDemo_detectedObj_t** in meters and meters/second respectively, then refer to the following:

1. In order to convert range index to range in meters, use the range formula mentioned in **Data Path - (X,Y) Estimation**, where R is the range and k_r is the range index. In terms of CLI's profile configuration:
   range in m = range index * (speed of light(3e8) * (digOutSampleRate * 1e3) / (2 * (freqSlopeConst * (1e6/1e-6)) * numRangeBins)) - range bias
   where the parameter names digOutSampleRate (ksps) and freqSlopeConst (MHz/us) are defined in CLI's profile configuration. Range bias is in meters and measured using the procedure described in **Range Bias and Rx Channel Gain/Offset Measurement and Compensation**.
2. In order to convert doppler index to velocity in meters/sec,
   velocity in m/s = doppler index * (speed of light(3e8) / (2 * (startFreq * 1e9) * ((idleTime + rampEndTime) * 1e-6) * numChirpsPerFrame))
   where the parameter names startFreq (GHz), idleTime (us) and rampEndTime (us) above are described in CLI's profile configuration. and numChirpsPerFrame is the total number of physical chirps in one frame [the quantity (idleTime + rampEndTime) * numChirpsPerFrame therefore represents the total chirping duration in a frame]. This quantity is related to profile configuration as follows:
   numChirpsPerFrame = (chirp end index - chirp start index + 1) * numLoops, the parameters are from frame configuration (frameCfg).
   If using advanced frame feature, then
   numChirpsPerFrame = numOfChirps * numLoops, the parameters are from sub-frame configuration (subFrameCfg).
   Note doppler index is signed as documented in **MmwDemo_detectedObj_t** so velocity in m/s will also be signed. Positive velocity means target is moving away from the sensor and negative velocity means target is moving towards the sensor.

Range profile

Type: (**MMWDEMO_OUTPUT_MSG_RANGE_PROFILE**)

Length: (Range FFT size) x (size of uint16_t)

Value: Array of profile points at 0th Doppler (stationary objects). In XWR16xx the points represent the sum of log2 magnitudes of received antennas, expressed in Q8 format. In XWR14xx the points represent the average of log2 magnitudes of received antennas, expressed in Q9 format.

Noise floor profile

Type: (**MMWDEMO_OUTPUT_MSG_NOISE_PROFILE**)

Length: (Range FFT size) x (size of uint16_t)

Value: This is the same format as range profile but the profile is at the maximum Doppler bin (maximum speed objects). In general for stationary scene, there would be no objects or clutter at maximum speed so the range profile at such speed represents the receiver noise floor.

Azimuth static heatmap

Type: (**MMWDEMO_OUTPUT_MSG_AZIMUT_STATIC_HEAT_MAP**)

Length: (Range FFT size) x (Number of virtual antennas) (size of **cmplx16ImRe_t_**)

Value: Array **MmwDemo_DSS_DataPathObj::azimuthStaticHeatMap**. The antenna data are complex symbols, with imaginary first and real second in the following order:

```
Imag(ant 0, range 0), Real(ant 0, range 0),...,Imag(ant N-1, range 0),Real(ant N-1, range 0)
...
Imag(ant 0, range R-1), Real(ant 0, range R-1),...,Imag(ant N-1, range R-1),Real(ant N-1, range R-1)
```

Based on this data the static azimuth heat map is constructed by the GUI running on the host.

Range/Doppler heatmap

Type: (**MMWDEMO_OUTPUT_MSG_RANGE_DOPPLER_HEAT_MAP**)

Length: (Range FFT size) x (Doppler FFT size) (size of uint16_t)

Value: Detection matrix **MmwDemo_DSS_DataPathObj::detMatrix**. The order is :

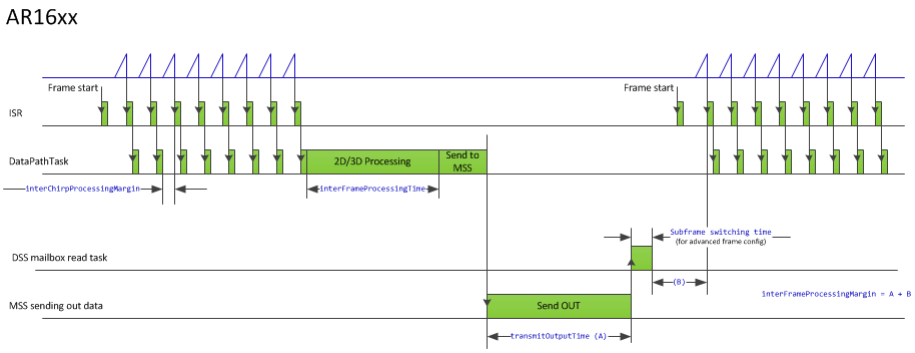
```
X(range bin 0, Doppler bin 0),...,X(range bin 0, Doppler bin D-1),
...
X(range bin R-1, Doppler bin 0),...,X(range bin R-1, Doppler bin D-1)
```

Stats information

Type: (**MMWDEMO_OUTPUT_MSG_STATS**)

Length: (size of **MmwDemo_output_message_stats_t**)

Value: Timing information enclosed in **MmwDemo_output_message_stats_t**. The following figure describes them in the timing diagram.

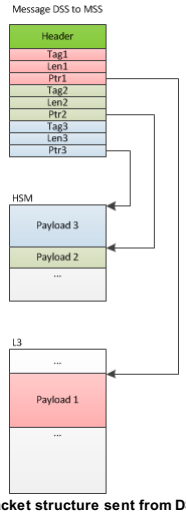


Margins and DSS CPU loading

The CLI command "guMonitor" specifies which TLV element will be sent out within the output packet. The arguments of the CLI command are stored in the structure **MmwDemo_GuiMonSel_t**.

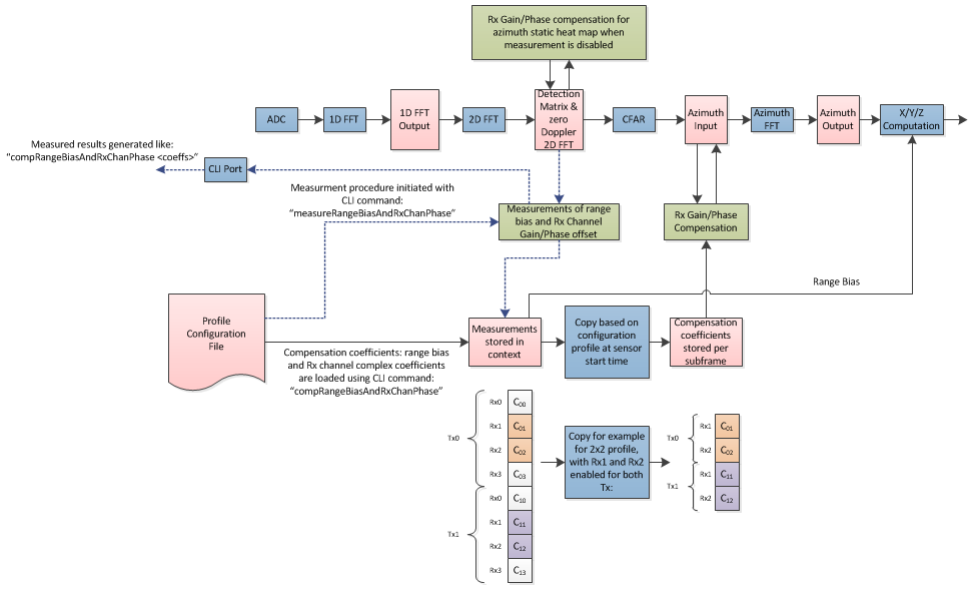
DSS to MSS packet structure

The packet construction is initiated on DSS side at the end of interframe processing. The packet is sent from DSS to MSS via mailbox. The structure of the packet on the mailbox is illustrated in the following figure. The packet header is followed by the number of TLV elements (**MmwDemo_msgTlv_t**) where the address fields of these elements point to payloads located either in HS-RAM or in L3 memory.



Output packet structure sent from DSS to MSS

Range Bias and Rx Channel Gain/Offset Measurement and Compensation



Measurement and compensation of range bias and Rx channel gain/offset

Because of imperfections in antenna layouts on the board, RF delays in SOC, etc, there is need to calibrate the sensor to compensate for bias in the range estimation and receive channel gain and phase imperfections. The calibration procedure is as follows:

1. Set a strong target like corner reflector at the distance of X meter (X less than 50 cm is not recommended) at boresight.
2. Set the following command in the configuration profile in .../demo/xwr16xx/mmw/profiles/profile_calibration.cfg, to reflect the position X as follows:

   ```
   measureRangeBiasAndRxChanPhase 1 X D
   ```

   where D (in meters) is the distance of window around X where the peak will be searched. The purpose of the search window is to allow the test environment from not being overly constrained say because it may not be possible to clear it of all reflectors that may be stronger than the one used for calibration. The window size is recommended to be at least the distance equivalent of a few range bins. One range bin for the calibration profile (profile_calibration.cfg) is about 5 cm. The first argument "1" is to enable the measurement. The stated configuration profile (.cfg) must be used otherwise the calibration may not work as expected (this profile ensures all transmit and receive antennas are engaged among other things needed for calibration).
3. Start the sensor with the configuration file.
4. To estimate the range bias, peak search is done after the 2D FFT in the 0th Doppler of the detection matrix. The peak position is then used to compute the square root of the sum of the magnitude squares of the virtual antennas (taken from **MmwDemo_DSS_DataPathObj::azimuthStaticHeatMap**) for the peak and its two nearest neighbors. These 3 magnitudes and their positions are used to do parabolic interpolation to find the more accurate peak from which the range bias is estimated as peak - X. The rx channel phase and gain estimation is done by finding the minimum of the magnitude squared of the virtual antennas and this minimum is used to scale the individual antennas so that the magnitude of the coefficients is always less than or equal to 1. The complex conjugate of the samples scaled in this way is stored in a common storage area (common across sub-frames) in Q15 format (**MmwDemo_CliCommonCfg_t::compRxChanCfg**). Refer to the function **MmwDemo_rangeBiasRxChPhaseMeasure** which performs the measurements and as seen in the above picture, the measurement results are written out on the CLI port in the format below:

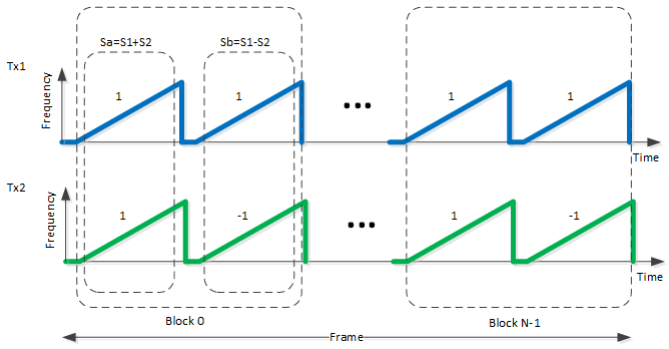
   ```
   compRangeBiasAndRxChanPhase <rangeBias> <Re(0,0)> <Im(0,0)> <Re(0,1)> <Im(0,1)> ... <Re(0,R-1)> <Im(0,R-1)> <Re(1,0)> <Im(1,0)> ... <Re(T-1,R-1)> <Im(T-1,R-1)>
   ```
5. The command printed out on the CLI now can be copied and pasted in any configuration file for correction purposes. During the parsing of the configuration file, based on the antenna profile, the measurement result stored as described in previous step in the common storage is copied to individual sub-frame data path object (each sub-frame may have separate configuration) so that all antennas enabled for that configuration are contiguous in storage as seen in the example shown in picture **Figure_calibration**. This contiguous storage of the samples themselves as opposed to storing indices to look-up into the common (full) storage is required for computational efficiency in the DSP. that is used during angle estimation to apply the correction by multiplying the received samples with the stored conjugate numbers. If compensation is not desired, the following command should be given

   ```
   compRangeBiasAndRxChanPhase 0.0   1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
   ```

   Above sets the range bias to 0 and the phase coefficients to unity so that there is no correction. Note the two commands must always be given in any configuration file, typically the measure commmand will be disabled when the correction command is the desired one.

BPM Scheme

Similar to TDM-MIMO, in BPM scheme a frame consists of multiple blocks, each block consisting of 2 chirp intervals. However, unlike in TDM-MIMO where only one TX antenna active per chirp interval, both Tx antennas are active in each chirp interval (see figure below).



BPM Scheme Antenna configuration

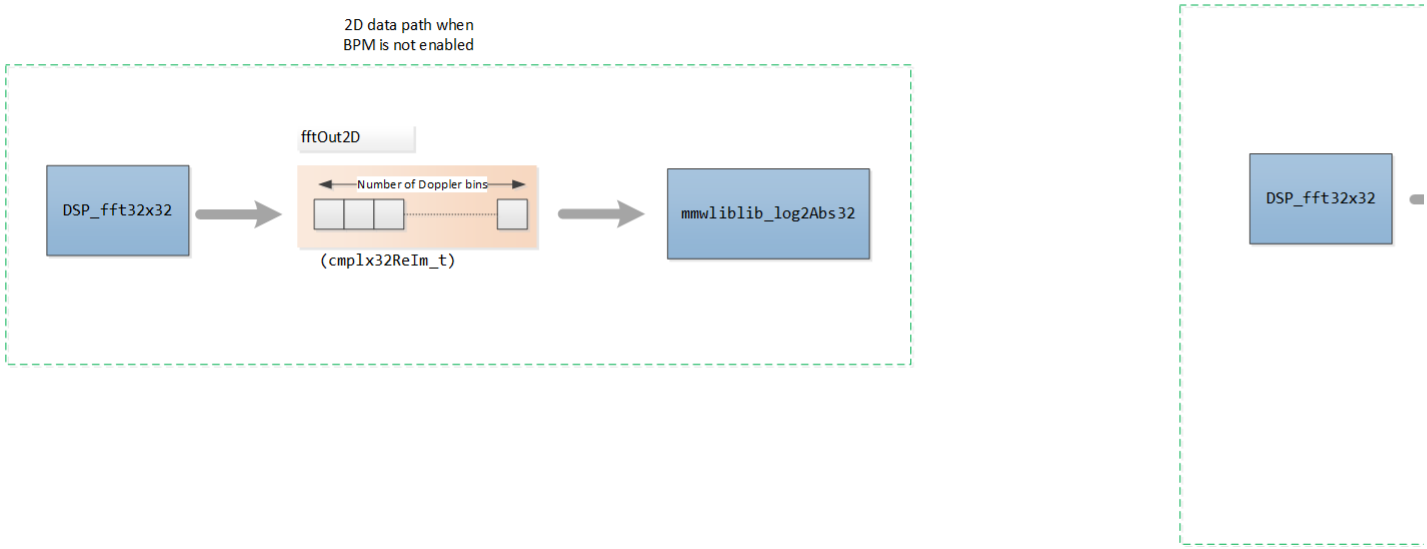
Let S1 and S2 represent chirp signals from two Tx antennas. In the first interval a combined signal Sa=S1+S2 is transmitted. Similarly in the second interval a combined signal Sb=S1-S2 is transmitted. Using the corresponding received signals, (S'a and S'b), at a specific received RX antenna, the components from the individual transmitters are separated out using S'1=(S'a+S'b)/2 and S'2=(S'a-S'b)/2.
With simultaneous transmission on both Tx antennas the total transmitted power per chirp interval is increased, and it can be shown that this translates to an SNR improvement of 3dB.

Data Path changes for BPM

When BPM is enabled the following changes are done in the data path.

2D Processing changes for BPM:

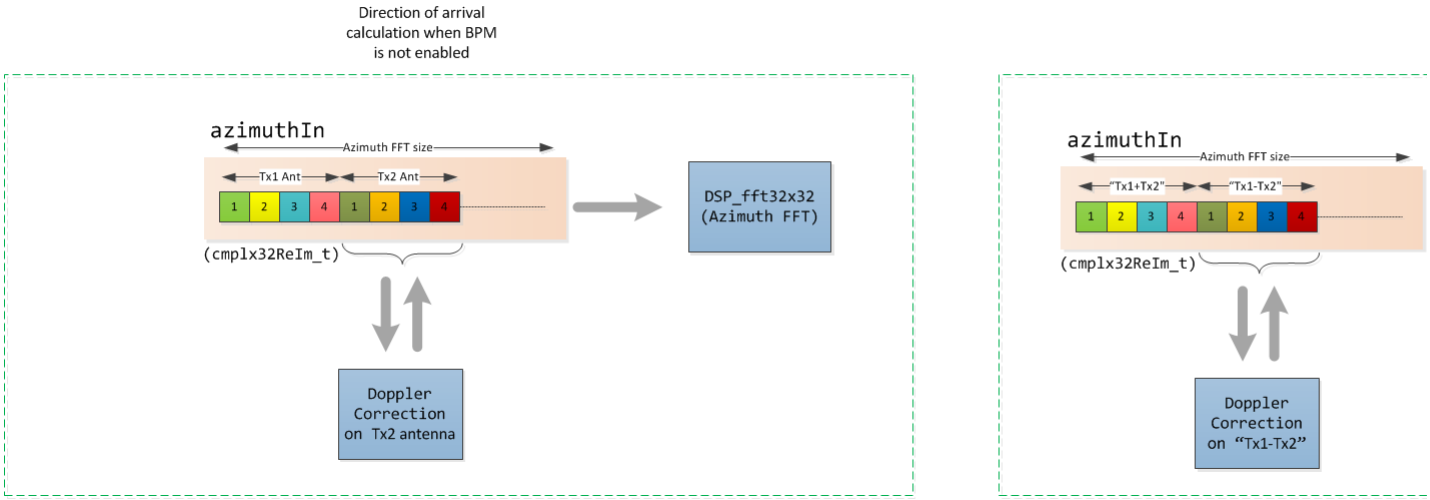
In the 2D processing chain, when BPM is enabled, Doppler compensation and BPM decoding are done after the 2D FFT. Note that the decoded data is not stored in the radar cube, therefore BPM decoding needs to be done again (on a much smaller set of samples) during the direction of arrival computation. The following figure shows the required changes in the 2D processing. When BPM is enabled the fftOut2D buffer is doubled in size to accommodate both Ping (Tx1+Tx2) and Pong (Tx1-Tx2) so that BPM can be decoded.



2D processing changes for BPM

Direction of Arrival Processing changes for BPM:

In the direction of arrival processing, when BPM is enabled, Doppler compensation and BPM decoding are done after the 2D FFT and before the azimuth FFT. The following figure shows the required changes in the direction of arrival processing.



Direction of arrival computation changes for BPM

Data Path Design Notes

CFAR processing:

For most scenarios, detection along range dimension is likely to be more difficult (clutter) than detection along the Doppler dimension. For e.g a simple detection procedure (CFAR-CA) might suffice in the Doppler dimension, while detection along range dimension might require more sophisticated algorithms (e.g. CFAR-OS, histogram based or other heuristics). So detection along Doppler dimension first might be computationally cheaper: the first selection algorithm is less complex, and the subsequent more sophisticated algorithm runs only on the points detected by the first algorithm. So in this implementation, we run Doppler CFAR first and then run range CFAR on the detected Doppler indices. However, we currently use the same type of algorithm (CFAR-CA) for the range direction as the Doppler direction. The range CFAR algorithm could be replaced by a more sophisticated algorithm like CFAR-OS to get the benefit of this way of processing.

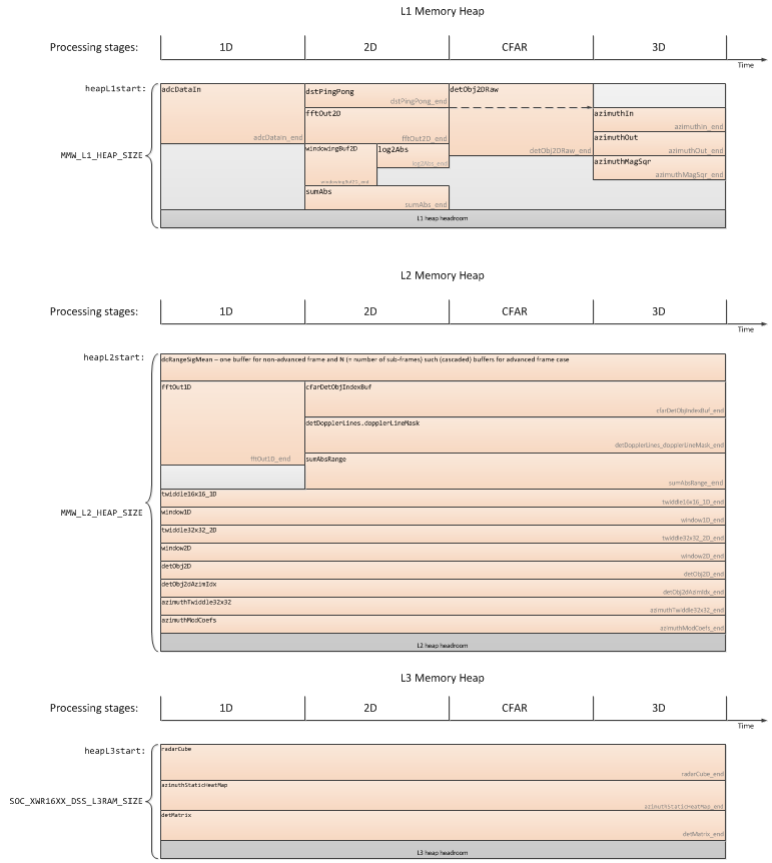
Scaling

Most of the signal processing in the data path that happens in real-time in 1D and 2D processing uses fixed-point arithmetic (versus floating point arithmetic). While the C647X is natively capable of both fixed and floating, the choice here is more for MIPS optimality but using fixed-point requires some considerations for preventing underflow and overflow while maintaining the desired accuracy required for correct functionality.

- 1D processing: If the input to the FFT were a pure tone at one of the bins, then the output magnitude of the FFT at that bin will be $N/\lceil 2^{log_4(N)-1} \rceil$ (N is the FFT order) times the input tone amplitude (because tone is complex, this implies that the individual real and imaginary components will also be amplified by a maximum of this scale). Because we do a Blackman window before the FFT, the overall scale is 0.42 of the FFT scale. For simplicity we will assume worst scale of 0.5 (this will be the case if someone choose Hanning window for example for 1D). This means for example for 256 point FFT, the windowing + FFT scale will be 16. Therefore, the ADC output when it is a pure tone should not exceed +/-2^15/16 = 2048 for the I and Q components. The XWR16xx EVM when presented with a strong single reflector reasonably close to it (with Rx dB gain of 30 dB in the chirp profile) shows ADC samples to be a max of about 3000 and while this exceeds the 2048 maximum, is not a pure tone, the energy of the FFT is seen in other bins also and the solution still works well and detects the strong object.
- 2D processing: For the 2D FFT, given that the input is the output of 1D FFT that can amplify its input as mentioned in previous section, it is more appropriate to use a 32x32 FFT which helps prevent overflow and reduce quantization noise of the FFT.

EDMA versus Cache based Processing

The C647X has L1D cache which is enabled and processing can be done without using EDMA to access the L2 SRAM and L3 memories through cache. However, the latency to L3 RAM is much more than L2SRAM and this causes cycle waits that are avoided by using EDMA to prefetch or postcommit the data into L2SRAM. The L1D is configured part cache (16 KB) and part SRAM (16 KB). Presently the L3 RAM is not declared as cacheable (i.e the MAR register settings are defaulted to no caching for L3 RAM). Various buffers involved in data path processing are placed in L1SRAM and L2SRAM in a way to optimize memory usage by overlaying between and within 1D, 2D and 3D processing stages. The overlay choices are based on a variety of configurations that this demo supports, so the choice may not necessarily be optimal for a specific/known configuration. The overlays are documented in the comments in the body of the function **MmwDemo_dataPathConfigBuffers** and can be disabled using a compile time flag for debug purposes. The function uses a macro **MMW_ALLOC_BUF** to facilitate expressing the cascading and parallelization of buffers. It creates a local variable <name>_end automatically to be used for subsequent cascaded allocation. Figure below also documents the same overlay scheme. The figure is not to scale as actual sizes will vary based on configuration.



Data buffers overlay arrangement

Streaming data over LVDS

The LVDS streaming feature enables the streaming of HW data (a combination of ADC/CP/CQ data) and/or user specific SW data through LVDS interface. The streaming is done mostly by the CBUFF and EDMA peripherals with minimal CPU intervention. The streaming is configured through the lvdsStreamCfg CLI command. The LVDS streaming configuration includes the following 3 items:

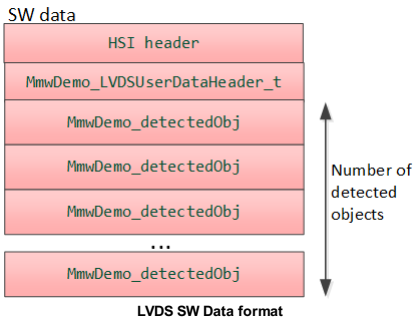
a) Enable/Disable HSI header. This controls if HSI header is enabled or disabled for all active streams (HW and/or SW). Note that if SW stream is enabled HSI header must be enabled.

b) HW streaming data format. This controls which type of HW data will be streamed (ADC/CP/CQ). The following values are allowed for this field
0 - HW streaming disabled
1 - ADC data streamed
2 - CP_ADC data streamed
3 - ADC_CP data streamed
4 - CP_ADC_CQ data streamed

c) Enable/Disable SW streaming.

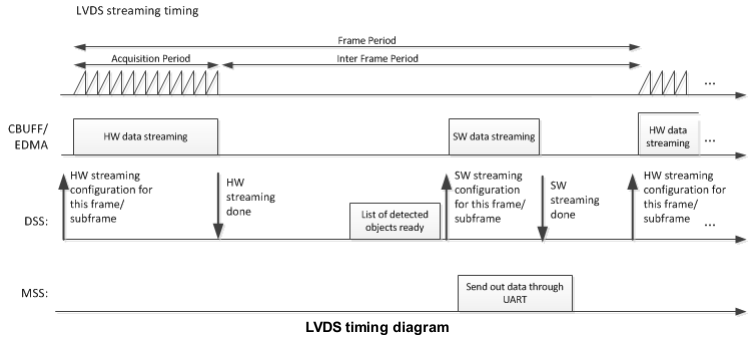
When HW data LVDS streaming is enabled the ADC/CP/CQ data is streamed per chirp as soon as the data is available in the HW buffers. When SW data streaming is enabled, it is streamed during inter frame period after the list of detected objects for that frame is computed. The SW data streamed is composed of 3 blocks:
i) HSI header (**HSIHeader_t**): refer to HSI module for details.
ii) User data header: contains the frame number, number of detected objects and 2 reserved bytes **MmwDemo_LVDSUserDataHeader**
iii) List of detected objects in that frame (one instance of **MmwDemo_detectedObj_t** per detected object).

The format of the SW data streamed is shown in the following figure:



LVDS SW Data format

The following figure shows a timing diagram for the LVDS streaming (the figure is not to scale as actual durations will vary based on configuration).



LVDS timing diagram

Memory Usage

MSS Memory usage summary

The table below shows the usage of various memories available on the device's MSS across the demo application and other SDK components. The table is generated using the demo's MSS map file and applying some mapping rules to it to generate a condensed summary. For the mapping rules, please refer to demo_mss_mapping.txt. The numeric values shown here represent bytes. Refer to the xwr16xx_mmw_demo_mss_mem_analysis_detailed.txt for detailed analysis of the memory usage across drivers and control/alg components and to demo_mss_mapping_detailed.txt for detailed mapping rules.

```
                                OVERVIEW
Memory                           Used            Total        Percent Used
-------------------------------------------------------------------------
DATA_RAM                        59764           196608              30.40%
HS_RAM                              0            32768               0.00%
L3_RAM                              0           786432               0.00%
PROG_RAM                        87078           261888              33.25%
VECTORS                            60              256              23.44%


                          Type            DATA_RAM       HS_RAM       L3_RAM     PROG_RAM      VECTORS
-------------------------------------------------------------------------------------------------------
APP                       code               10592            0            0        37565           60
APP                       heap               43008            0            0            0            0
BIOS                      code                   0            0            0        20547            0
COMPONENTS_CORE           code                1488            0            0        23400            0
COMPONENTS_OPTIONAL       code                 580            0            0         4134            0
linker-generated          linker              4096            0            0          438            0
linker-generated          unknown                0            0            0          994            0
```

DSS Memory usage summary

The table below shows the usage of various memories available on the device's DSS across the demo application and other SDK components. The table is generated using the demo's DSS map file and applying some mapping rules to it to generate a condensed summary. For the mapping rules, please refer to demo_dss_mapping.txt. The numeric values shown here represent bytes. Refer to the xwr16xx_mmw_demo_dss_mem_analysis_detailed.txt for detailed analysis of the memory usage across drivers and control/alg components and to demo_dss_mapping_detailed.txt for detailed mapping rules.

```
                                OVERVIEW
Memory                           Used            Total        Percent Used
-------------------------------------------------------------------------
PAGE 0:
HSRAM                           32768           32768             100.00%
L1DSRAM                         16384           16384             100.00%
L1PSRAM                         15392           16384              93.95%
L2SRAM_UMAP0                   120549          131072              91.97%
L2SRAM_UMAP1                   131072          131072             100.00%
L3SRAM                           2208          786432               0.28%
PAGE 1:
L3SRAM                         786432          786432             100.00%


                          Type              HSRAM      L1DSRAM      L1PSRAM L2SRAM_UMAP0 L2SRAM_UMAP1       L3SRAM
-------------------------------------------------------------------------------------------------------------------
ALG                       code                  0            0         6944           64           96            0
APP                       code                  0            0         8352        24735        54992         1648
APP                       heap              32768        16384            0        72712            0       786432
BIOS                      code                  0            0           16         7480        25600           64
COMPONENTS_CORE           code                  0            0           48        11656        33088          480
COMPONENTS_OPTIONAL       code                  0            0            0         2100        17280            0
linker-generated          linker                0            0           32         1802           16           16
```

Note on L3 memory and overlay

A quick look at the L3_SRAM column will show that the total of that column exceeds the total physical memory available on the device. The reason is that we use the code-data overlay mechanism to virtually extend the available memory on the device. One-time startup code is overlaid with the radar cube. At startup, the application code accesses these functions to perform one-time setup functionality. Beyond that point, application code does not have a need to access these functions again and hence switches to access radarCube placed at the exact same location. Refer to the linker command file of the demo on the mechanics of this overlay technique.

L1P/L3 overlay in 16xx DSS

Currently bootloader does not allow loading in L1PSRAM and hence we use the overlay and copy table functionality of the linker to specify the load address as L3_SRAM but run address as L1P_SRAM for some of the functions that have time-critical operations (for e.g., mmWaveLib functions). In the table we show such functions as placed in L1P_SRAM and do not show it under L3_SRAM column since the startup code takes care of copying from L3_SRAM and placing it in L1P_SRAM and that memory in L3_SRAM is available for use for other purposes.