Gesture Recognition Demo Implementation Details

Revision 0.70

June 2023

Texas Instruments, Incorporated
TI India, Bangalore -560093

Version History

Date	Owner	Version number	Comment
April 10, 2023	Akshay Chandrasekaran	0.5	Initial version
April 18, 2023	Akshay Chandrasekaran	0.6	Additions based on initial feedback
June 26, 2023	Akshay Chandrasekaran	0.7	Addition of virtual antenna array figures supported in release

Table of Contents

1 Purpose and Scope

This document provides the Gesture Recognition algorithmic and implementation details on xWRLx432 using MMWAVE L SDK. Sections 2 to 6 talk about the high algorithmic level details about the demo while Sections 8 through 16 describe the actual implementation of the application on the target. The document describes the Data Processing Units (DPU) used, feature extraction library, gesture classifier library, and the mode switch between gesture mode and presence mode. It also provides (Section 7) a quick start guide on how to use demo with TI EVM xWRLx432.

Algorithmic level details

2 Gesture Recognition Demo Setup

The gesture recognition demo setup is shown in Figure 1. The demo configuration is provided by the CLI configuration file. The PC visualizer is connected with the xWRLx432 EVM board via single UART port. The gesture recognition algorithm runs on the xWRLx432 radar sensor. Initially, to start the demo, a set of CLI configuration commands is sent to the EVM. Based on the configuration, the EVM sends through the same port various information to the visualizer, such as features extracted, presence/gesture mode, gesture detections and some statistics.

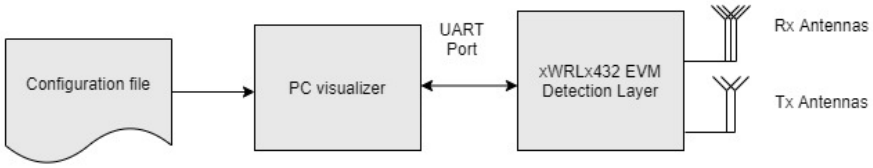


Figure 1 – Gesture recognition setup

The demo operates in two modes: (1) Presence Detect Mode and (2) Gesture Mode. The Presence Detect mode is a low power mode, where the device is monitoring the presence of motion in a predefined zone (currently set to distance from 20cm to 2.1m). On detection of presence, the device transitions to the Gesture Mode which detects gestures.

The processing layers of the gesture recognition demo (in the gesture mode) are shown in Figure 2.

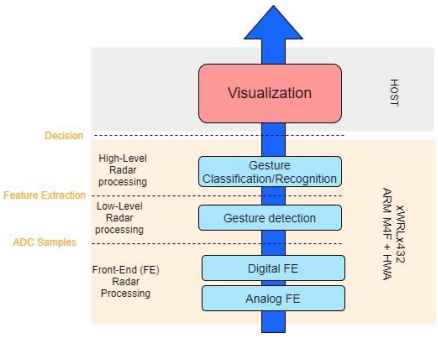


Figure 2 – The signal processing layers of the gesture recognition demo

3 Modulation Scheme, Chirp Configuration, Antenna geometry

Gesture recognition demo supports **TDM MIMO** mode of operation only.

3.1 Chirp configuration in the Gesture Mode

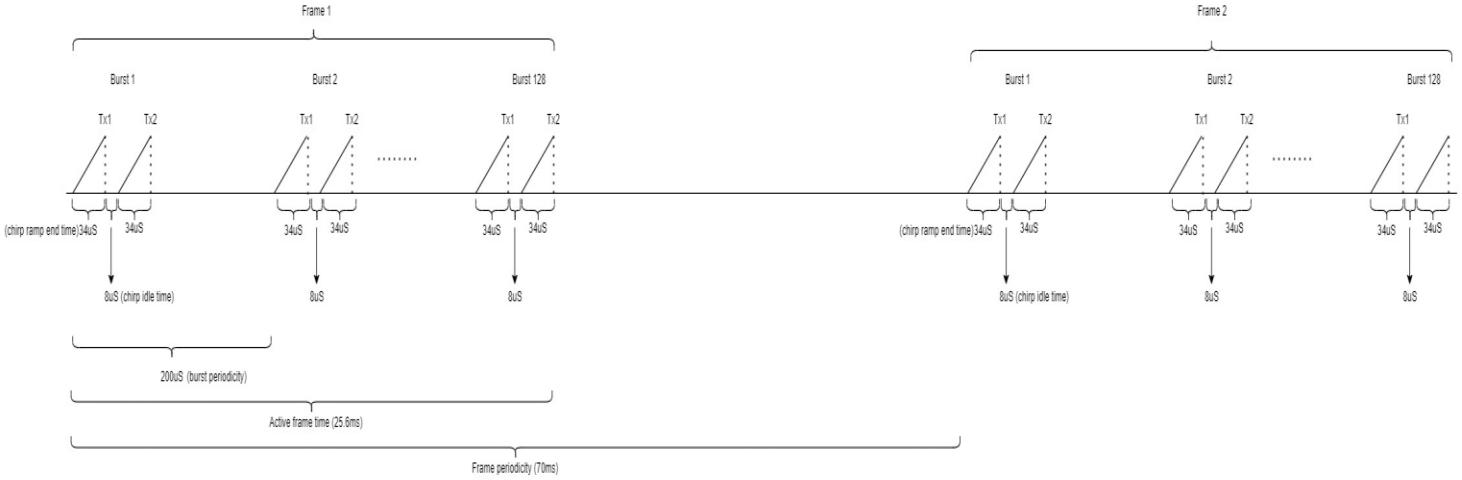


Figure 3 – Chirp config for gesture mode

Active frame time = 128 * 200uS = 25.6ms

Frame periodicity = 70ms

Ramp end time = 34uS

Idle time = 8uS

Chirps in burst = 2

Num of bursts = 128

Burst periodicity = 200uS

Num of ADC samples = 128 (real only architecture)

Sampling rate = 4.71Mhz

Slope = 102.98 Mhz/uS

Start frequency = 77GHz (1432), 59Ghz (6432)

3.2 Chirp Configuration in the Presence Detection Mode

The Chirp configuration in the Presence Detection Mode has a similar chirp profile as in the gesture mode, but uses only one burst per chirp and uses 1Tx-1Rx config. The gap between the bursts is appropriately set depending on the desired sensitivity to motion [default value is 1.5ms via CLI as of now]. The chirp configuration is shown below in Figure 4:

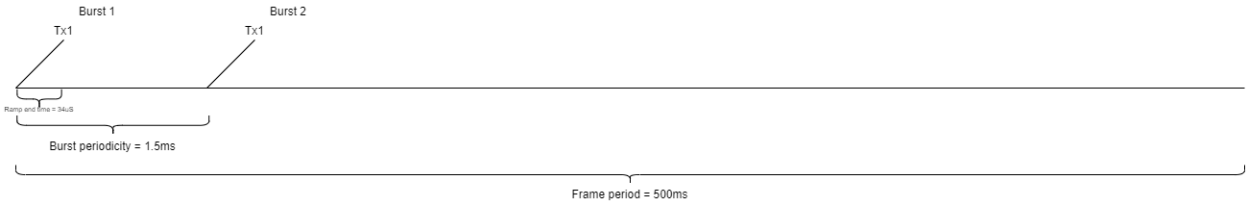


Figure 4 – Chirp config for presence mode

The antenna geometry is explained in detail in section 5 of [1].

4 Signal Processing Chain

4.1 Gesture mode

The top-level signal processing chain for gesture recognition implemented on xWRLx432 is shown in Figure 5.

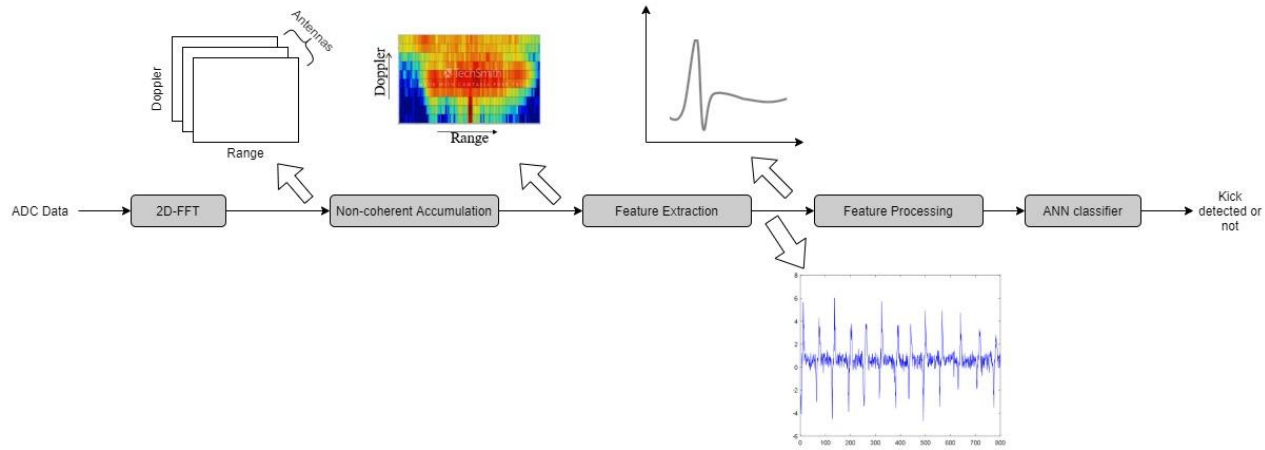
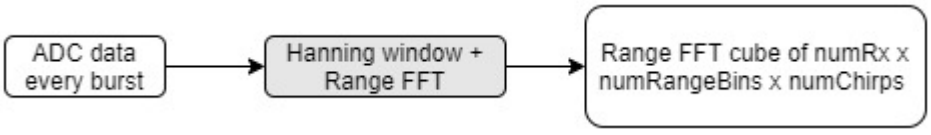


Figure 5 – Top level view of radar processing chain

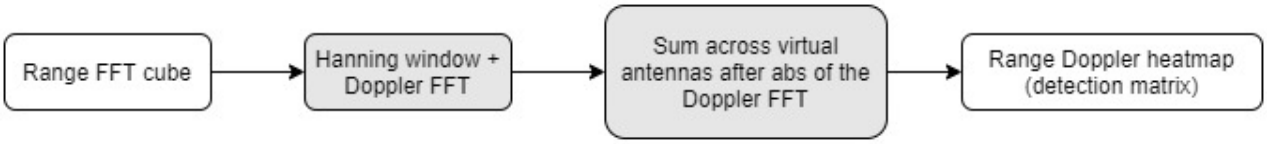
The signal processing is split among the following data processing functional block units (DPUs)

- Range DPU (*rangeproc*):



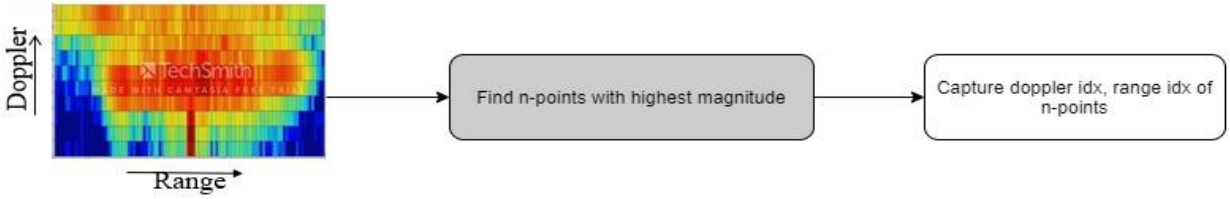
  - Range FFT (with windowing) for each antenna and chirp.
  - Output will be the Range FFT cube (num virtual antennas x num Range bins x num Chirps).

- Doppler DPU (*dopplerproc*):



- Doppler FFT per each antenna
- Summation of absolute value across virtual antennas of Doppler cube
- Output will be the detection matrix (num Range bins x num Doppler Bins)

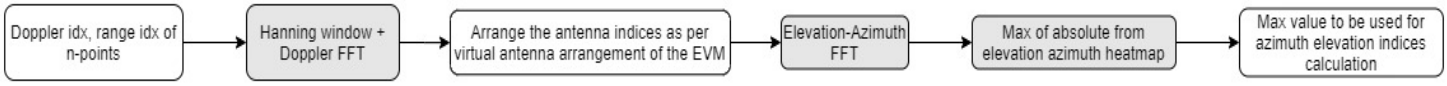
- Max of n-points:



Range Doppler heatmap

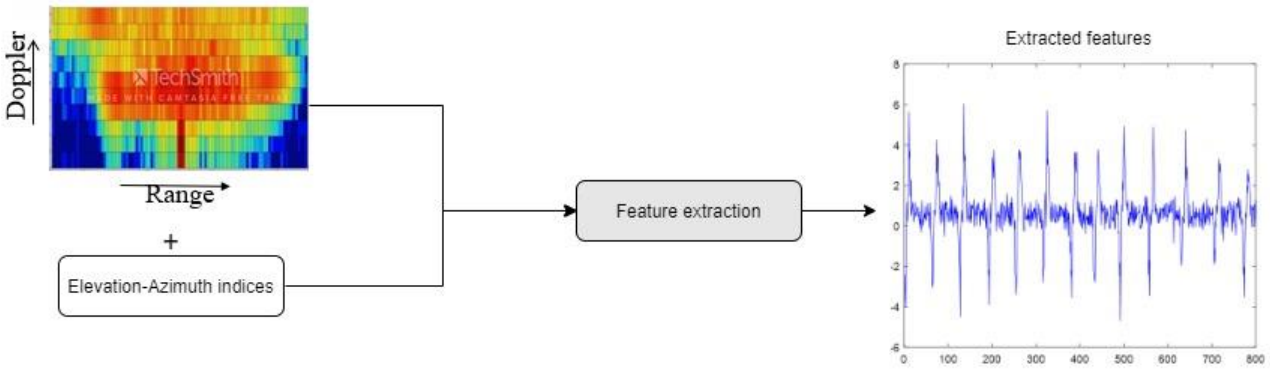
- Finding max of n-points will be part of the gesture recognition application code itself right after the range-Doppler heatmap creation.
- Find the n-points from range-Doppler heatmap of maximum magnitude (n is currently set to 25 in the demo].
- This is achieved by iteratively (n times) passing the range-Doppler heatmap through the HWA, finding the max values and zeroing the max values each time we find max value.
- The sub-set of range gates which contain these points will be subjected to recalculation of Doppler and angle processing in the AoA DPU.

- Angle of Arrival (AoA) DPU (*aoaproc*):



- From the previous step we have the range gates on which angle-FFT is to be performed.
- As we don't have enough space in xWRLx432 to store both the range and Doppler cube, we have to perform Doppler FFT for range gates of interest from previous step.
- Doppler compensation is performed after which the data is arranged as per antenna configuration to perform elevation-azimuth FFT.
- After this the corresponding elevation and azimuth of object is found out which will be used in feature extraction step.
- Note: (a) The angle-FFT processing of the SDK only supports the antenna configuration shown in section 5 of MotionPresenceDetectionDemo_documentation

- Feature Extraction (*rangeDopplerBased*):



  - Feature extraction from range-Doppler heatmap, angle related features using elevation, azimuth indices.
  - Feature extraction is done using the Feature extraction library (.lib) as a part of SDK (source/alg/featExtract/rangeDopplerBased)

- ANN Classifier (*gestureClassifier*):
  - Used to classify the gestures based on input features across frames as shown in Figure 6.
  - There is a normalization (standard normalization) performed before giving to ANN to increase its performance.
  - Post processing the data by counting number of gestures within certain number of frames above a certain threshold to reduce false alarms.
  - Classification is done using the Classification library (.lib) as a part of SDK (source/alg/classifier/gestureClassifier)

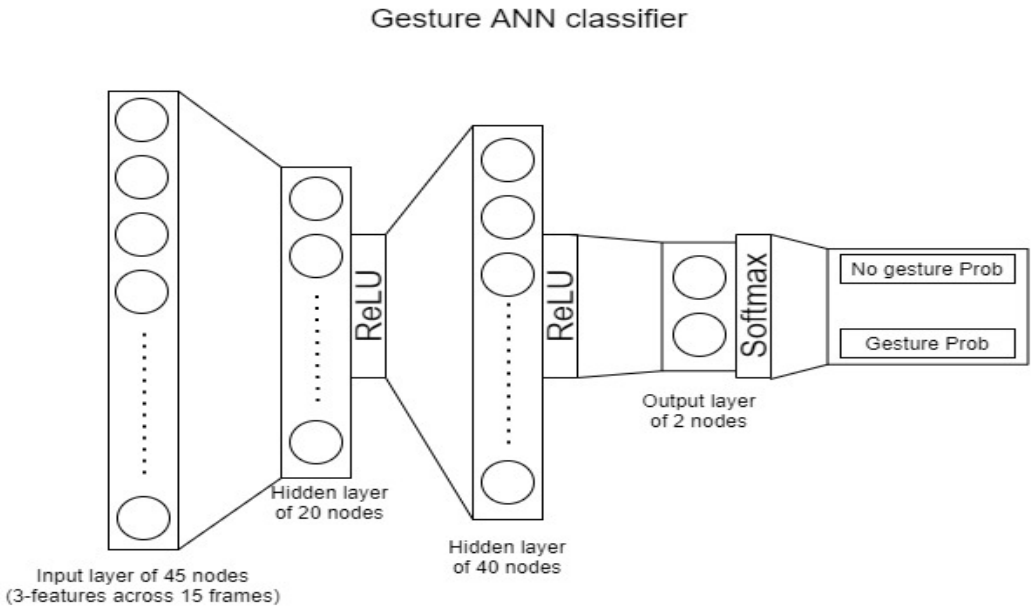
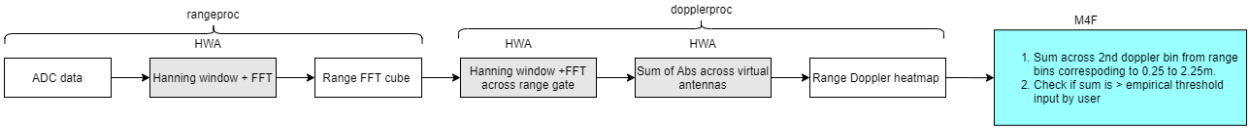


Figure 6 – ANN classifier diagram used for gesture recognition

4.2 Presence Detection Mode

Most of the time the device is expected to be in the Presence Detection Mode. This is a mode that is optimized for presence detection with minimal power. It uses only 2 bursts per frame and a low frame rate (currently set to 500ms). On detecting motion, the device transitions to the Gesture Mode. The signal processing steps are as shown in the figure below:



- As shown in the figure we perform range-FFT for the 2-bursts.
- Once we have the range-FFT cube, Doppler FFT is performed across 2-bursts to create range-Doppler heatmap.
- Sum across 1st Doppler bin (for range bins controlled by MACRO in meters *MIN_RANGE_METERS* and *MAX_RANGE_METERS)* and check if sum is greater than a particular threshold to make sure it goes into gesture mode

5 CLI commands

The parameters presented in this section are used to configure the gesture recognition demo and can be adjusted to match the use case. The user can set the configuration parameters using the configuration (i.e., cfg) files located in the gesture recognition demo application folder. An example configuration file for presence detection + gesture recognition mode is shown below. Each line in this file represents a CLI message that configures several parameters.

channelCfg 3 3 0

chirpComnCfg 21 0 0 128 1 34 0

chirpTimingCfg 8 30 0 102.98 76

frameCfg 2 0 200 128 70 0

sigProcChainCfg2 32 32 1 0.2 1.7 25

adcDataSource 0 radar_data_6432_100framesS_pres1.bin

adcLogging 0

presenceDetectCfg 1 3000 1500

lowPowerCfg 1

factoryCalibCfg 1 0 40 0 0x1ff000

sensorStart 0 0 0 0

All the CLI commands are same as for presence detect demo as mentioned in section 3 of [2] (as part of mmWave L SDK which can be used for tuning the parameters as required) except for the presenceDetectCfg and sigProcChainCfg2. Ideally the start frequency marked in blue (will change if its xWRL1432 device – operating range is 76 to 81GHz) to 76.

sigProcChainCfg2 CLI command has the structure:

<azimuthFftSize> <elevationFftSize> <motDetMode> <minDist(in meters)> <maxDist(in meters)> <maxNumPoints>

- <azimuthFftSize> is the size of azimuth FFT size to be applied in AOA DPU (best to keep <=32). For *LINEAR_ARRAY* along azimuth direction, this is set to 1.
- <elevationFftSize> is the size of elevation FFT size to be applied in AOA DPU (best to keep <=32). For *LINEAR_ARRAY* along elevation direction, this is set to 1.
- <motDetMode> is always set to 1 (major mode detection only) for gesture recognition demo
- <minDist> and <maxDist> corresponds to the distance in meters where the gesture would be detected. For the given CLI commands above, the range is set from 20cm to 1.5m (best to keep values 0.16-0.32 and 1.6-1.7 for min and max distance respectively).
- <maxNumPoints> corresponds to the n-points with maximum magnitude to be chosen from range-Doppler heatmap for elevation and azimuth indices calculation (best to keep it <=50)

presenceDetectCfg CLI command has the structure:

<presenceDetectEnable> <thresholdPresenceDetect> <burstPeriodicity>

- <presenceDetectEnable> - to enable or disable the presence mode (0-gesture mode, 1-presence->gesture and vice-versa, 2-presence mode only to test the threshold value)
- <thresholdPresenceDetect> - threshold value in presence mode above which the switch will happen to gesture mode (chosen empirically after setting in presence test mode)
- <burstPeriodicity> - burst periodicity (in uS) between 2-bursts to be used in presence mode

6 State machine for presence + gesture mode

State Machine Working for Presence + gesture mode

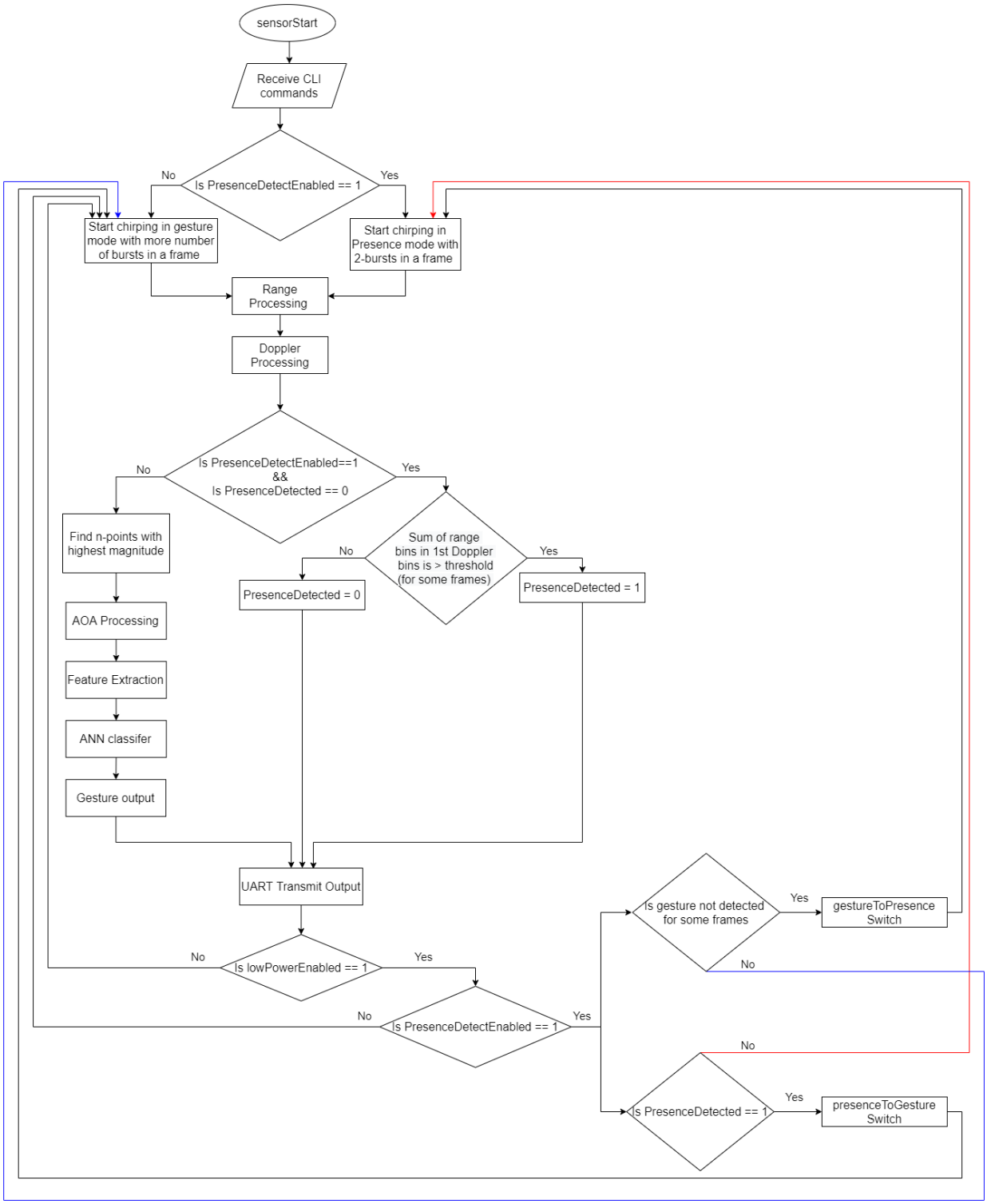


Figure 7 – Top level state machine working for presence + gesture mode operation

7 Quick start guide for running the demo

- Install the mmWave L SDK in the c:/ti/ folder preferably
- Make sure “make”/”gmake” is part of the environment variables
- Navigate to the folder of mmWave L SDK in the cmd prompt/windows shell where gmeake is added as environment variable
- Build the libs using the command: gmake -s -f makefile.xwrL64xx libs for 6432
  gmake -s -f makefile.xwrL14xx libs for 1432

- Build the demo using the command:
  gmake -s -C examples/mmw_demo/gesture_recognition/xwrL64xx-evm/m4fss0-0_freertos/ti-arm-clang all

- Once the appimage is created, flash using the visualizer in C:\ti\mmwave_lp_sdk\tools\visualizer\Low_power_visualizer_5.1.0.3
- Then send the CLI commands via tera-term (set the baud rate to 1250000)/GUI

Implementation Level details

8 Signal Processing Chain and Timing diagram

The top-level signal processing chain implemented on xWRLx432 is shown in Figure 8.

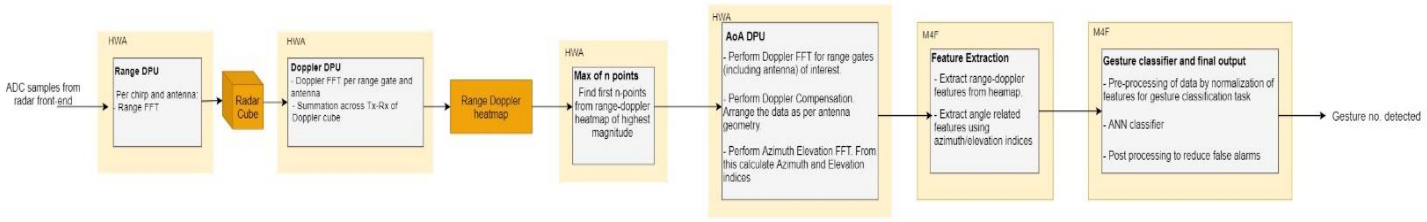


Figure 8 – High level view of radar signal processing chain

Most of the processing starting from the Range DPU through the AoA DPU is done in the HWA. The Feature Extraction and Gesture Classification are run on the M4F.

The high-level timing diagram of how various steps in signal chain happen in the chirp and frame time is shown in Figure 9.

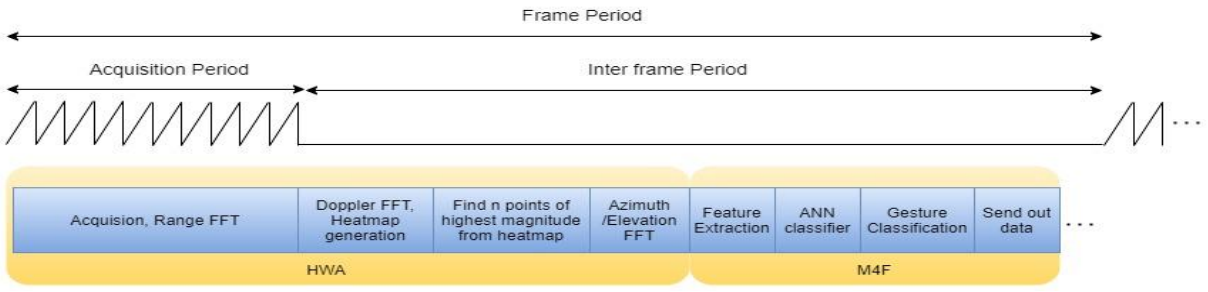


Figure 9 – Signal processing high level timing diagram

9 Data Processing Unit (DPU), Feature Extraction and Classifier

This section provides detailed description of DPU components, feature extraction library and gesture classifier used in the application.

9.1 Range processing DPU

The range processing DPU is explained in detail in [1]. Note that only major mode of operation in rangeproc DPU is supported for gesture recognition demo.

9.2 Doppler DPU

The top-level diagram of the Doppler DPU is shown in Figure 10. Based on the input radar cube this DPU creates the range-Doppler detection matrix (heatmap) [*gMmwMSSMCB.detMatrix.data*] that is used further in the chain ('max of n points' routine and Feature Extraction).

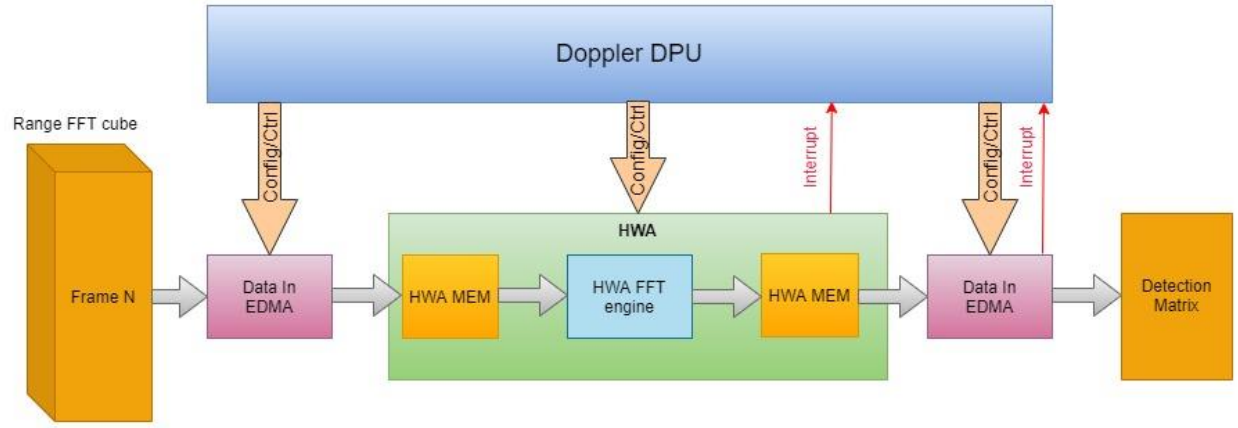


Figure 10 – Doppler DPU top level diagram

Detailed diagram of Doppler DPU implementation is shown in Figure 11.

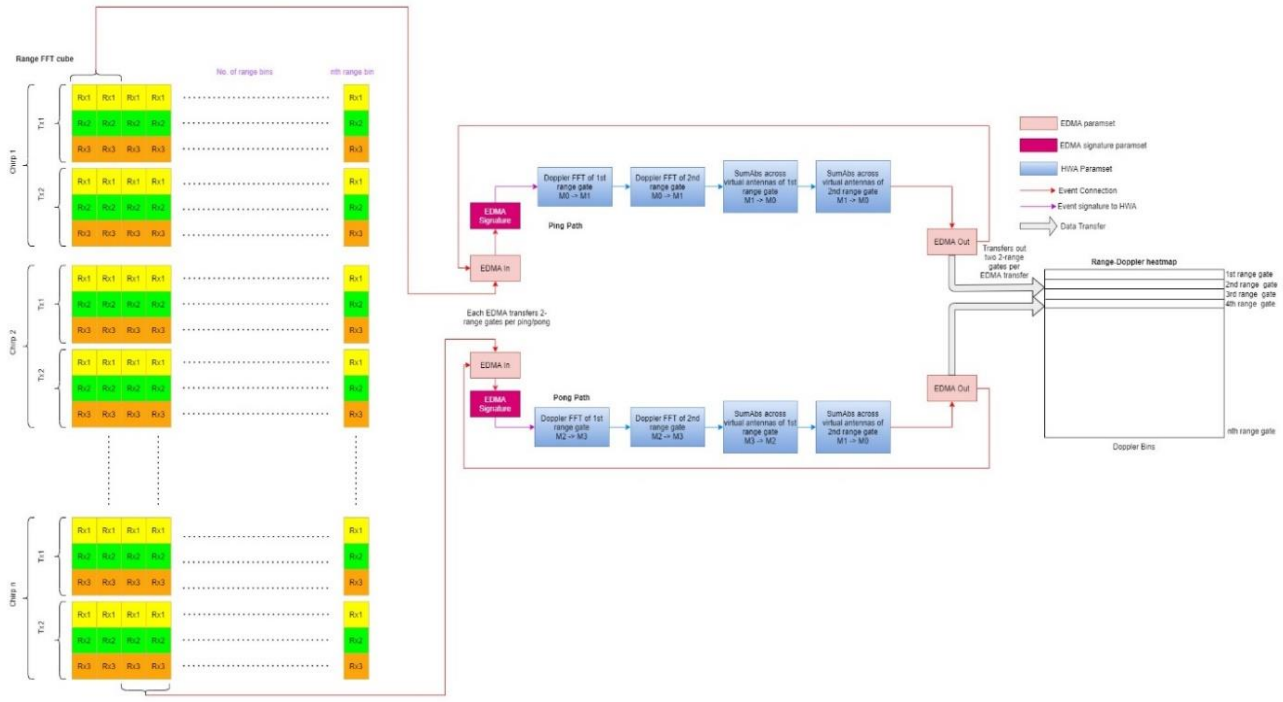


Figure 11 – Doppler DPU detailed diagram implementation with HWA and EDMA

The HWA processes data in the loop in a ping pong fashion with 4-range gates processed per loop (2-range gates per ping/pong path). Two range gates (complex 16-bit input range bin) are taken per ping/pong path considering 64-bit interconnect (to fill the bus fully for efficient transfer) between HWA memory and HWA_SS shared RAM. The input EDMA reads data from the radar cube (all chirps and antennas of the current 2-range gates) and copies to the HWA memory in linear fashion (top to bottom along the chirp dimension).

There are 4-paramsets in each ping/pong path which perform the following functions:

- Doppler FFT per antenna for 1st range gate,
- Doppler FFT per antenna for 2nd range gate,
- Sum of Abs across virtual antennas for the 1st range gate,
- Sum of Abs across virtual antennas for the 2nd range gate,

The output of the Doppler DPU is the detection matrix (range-Doppler heatmap) with dimensions [num of range bins x num of Doppler bins]. Note that the detection matrix is stored in M4F RAM for feature extraction process, AoA DPU computations and effective memory usage in xWRLx432 devices.

9.2.1 Doppler DPU API

9.2.1.1 DPU_DopplerProcHWA_init()

This function allocates memory for the Doppler DPU instance and initializes it to zero. It also constructs the semaphores used for processing.

9.2.1.2 DPU_DopplerProcHWA_config()

Based on the configuration parameters, this function configures hardware accelerator FFT engine and the input and output EDMA channels to bring data in and out of HWA memory. The function is normally called once during initialization, before the sensor start command is issued to the RF. In the low power deep sleep mode, the function is called per frame.

9.2.1.3 DPU_DopplerProcHWA_process()

This function is called per frame after the range DPU is completed for creating the range-Doppler heatmap.

The function performs the following steps:

- configures the common registers of the HWA, such as start and end param set, and the number of loops,
- triggers the input EDMA,
- pends on the semaphores for the end of the processing as the entire DPU processing is performed autonomously by HWA/EDMA after the 1st EDMA trigger.

Then the function exits.

9.2.1.4 DPU_DopplerProcHWA_deinit()

It frees the resources used for the DPU.

9.3 Presence Detection Mode Implementation

This is an optional mode for power saving with slow chirping (2-bursts per frame with frame period as high as 250ms). In this mode, objective is to find if there is any presence in front of radar; full chirping starts only when presence is detected in front of the radar. The signal processing chain is same till the Doppler processing after which the energy (sum of abs) is computed for the non-zero Doppler bin as shown below in Figure 12:

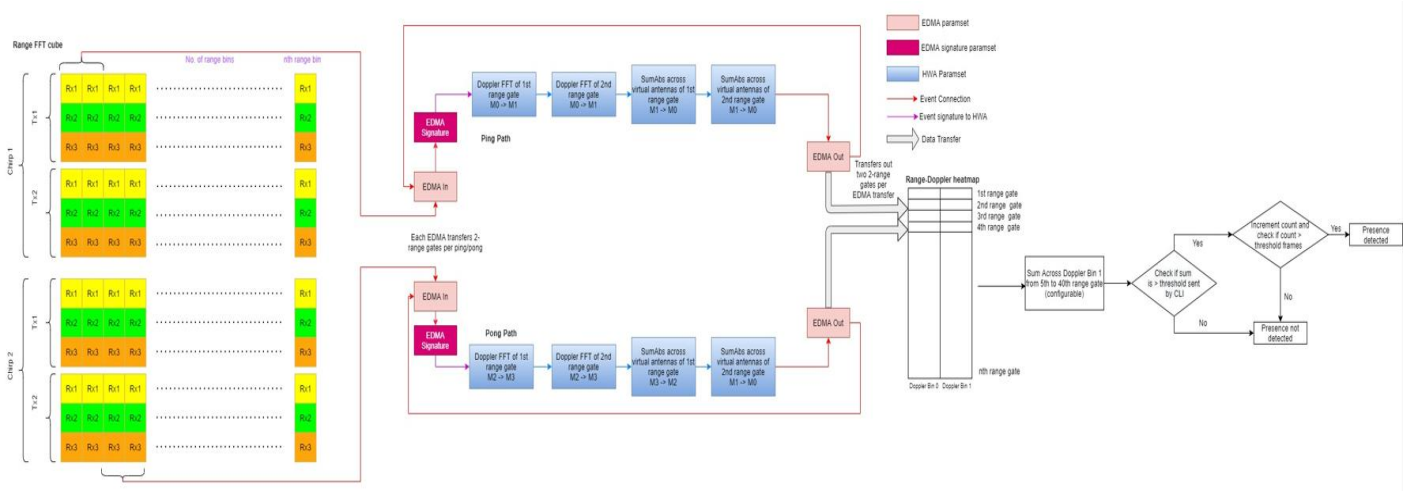


Figure 12 – Doppler DPU for presence detection followed by presence detect logic

As shown in the figure above, the following algorithm is used for presence detection:

- In the detection matrix, sum across Doppler bin 1 [in function *presence_detection()*] for few number of range gates (*MIN_RANGE_BIN_PRESENCE* to *MAX_RANG_BIN_PRESENCE*)
- If the sum is greater than threshold (*gMmwMssMCB.presenceDetectCfg.thresholdRange*) input via CLI by user, then the count (*sumFramesPresenceDetect*) is incremented.
- If this count is greater than *NUM_FRAMES_PRESENCE*, then presence detect flag is set and all the steps of further signal processing chain is skipped and it directly enters *powerManagementTask*

Thus, in this mode, the chirping and processing time is very less (<1% duty cycle) and hence the deep sleep time is more in this mode (more power saving).

Note that this mode is entered only when the presence detect mode is enabled via CLI. When this mode is enabled, *presenceDetectChirpParams()*[function which gets the chirp parameters for presence mode and also stores the chirp params of gesture mode in another struct to be used for chirp config switching] is the first function called before execution of *CLI_MMWStart()* which creates other tasks related to datapath processing chain.

9.4 Finding n-points from range-Doppler heatmap of highest magnitude

This functionality is not built as a DPU but is included as a bunch of routines directly in the application (maxOfDetectionMatrix.c). Using the HWA, it picks out n-points with the highest magnitude from the detection matrix (range Doppler heatmap). Figure 13 below shows a high-level overview of this functionality.

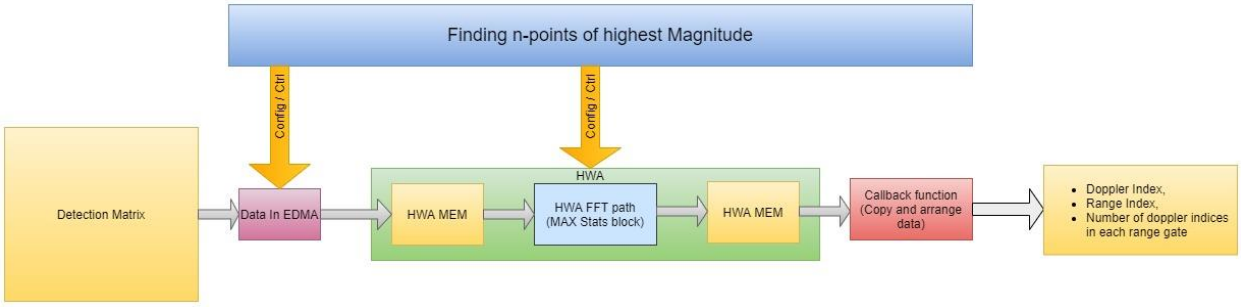


Figure 13 – Top level flow for finding max of n-points from range-Doppler heatmap

Figure 14 below shows a more detailed description of the functionality:

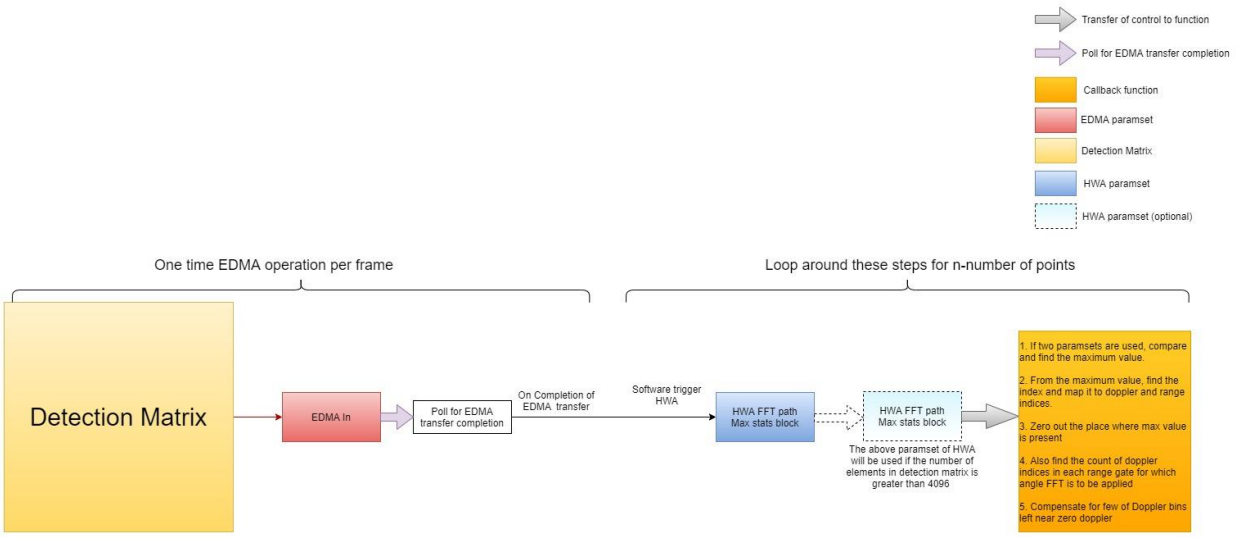


Figure 14 – n-points with maximum magnitude detailed diagram

As mentioned in the above diagram, the detection Matrix is taken into HWA memory by EDMA (once per frame). This is confirmed by polling for EDMA transfer completion bit.

The first 3-banks of the HWA server as the input (for the detection matrix) while the 4th bank is where the output (maximum values along with their indices) is stored. Once EDMA transfer is completed, there is a necessity to loop through the detection matrix n-number of times (successively zeroing out the maximum value identified after each iteration) to find n-points of highest magnitude. The HWA supports a maximum ACNT of 4096. So, if the number of elements in the detection matrix is greater than 4096, we use two paramsets (with the first and second paramset, processing the 1st half and 2nd half of samples respectively).

Once the HWA gives out the max values along with indices, additional processing is done (*maxOfDetectionMatrix_copyDataOut()*) to copy out the data in the right format

Three arrays are output (which enables the AoA DPU to reference the right sections of the Radar Cube for Angle estimation):

- range index: Range index of each of the n detected points, stored in a 1D-array of size n-points.
- Doppler index: For each range bin, it lists the Doppler indices of all the points detected at this range index. Stored in 2D-array of size n-points x num_range bins.
- angle index Doppler count: Lists the number of detected points per range bin. Stored in 1D-array of size 1 x num_range_bins.

9.4.1 API for max of n-points in detection matrix

9.4.1.1 maxOfDetectionMatrix_config()

Configures the HWA and EDMA paramsets. The function is normally called once during initialization, before the sensor start command is issued to the RF. In the low power deep sleep mode, the function is called per frame.

9.4.1.2 maxOfDetectionMatrix_HWACommonConfig()

Resets the Doppler index array, range index array, angle index Doppler count arrays to zero. This function gets called every frame.

9.4.1.3 maxOfDetectionMatrix_copyDataOut()

This does the following operations:

- If two paramsets were used to find max values (with 1st half and 2nd half of samples respectively processed via the first and second paramset)
    - Compute the maximum of the max values returned by each of the paramsets
    - The max indices returned by the HWA are compensated to reflect the true index in range-Doppler heatmap.
- Store the indices corresponding to the maximum. Populate the 3 arrays: range index, Doppler index, angle index Doppler count
- Compensate the indices for the Region of interest (see Section 9.6)

9.5 Angle of Arrival (AoA) DPU

The top-level diagram of the AoA DPU is shown in Figure 15.

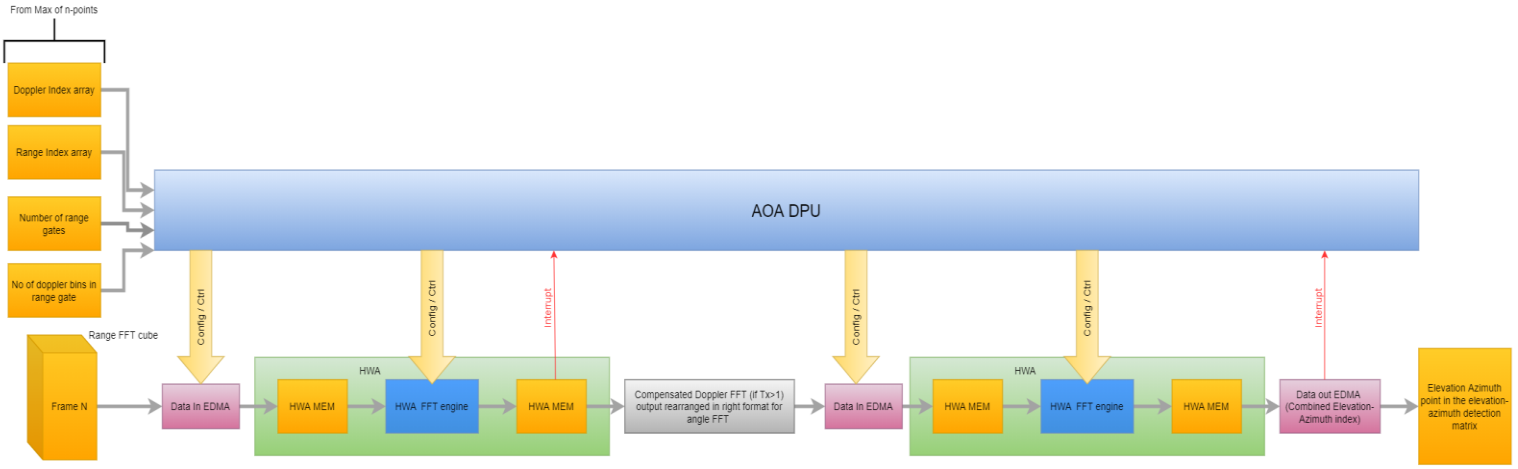


Figure 15 – Top-level diagram of AOA DPU

The AOA DPU recalculates the Doppler-FFT for the range gates (we don't store the entire Doppler cube to save memory) corresponding to the n detected points identified in the 'Max n-points' block. It also does Doppler compensation (if the number of Tx > 1), and arranges data for angle FFT computation. Once all

the data corresponding to the n detected points are arranged as per antenna config, then azimuth elevation FFT is performed on it.

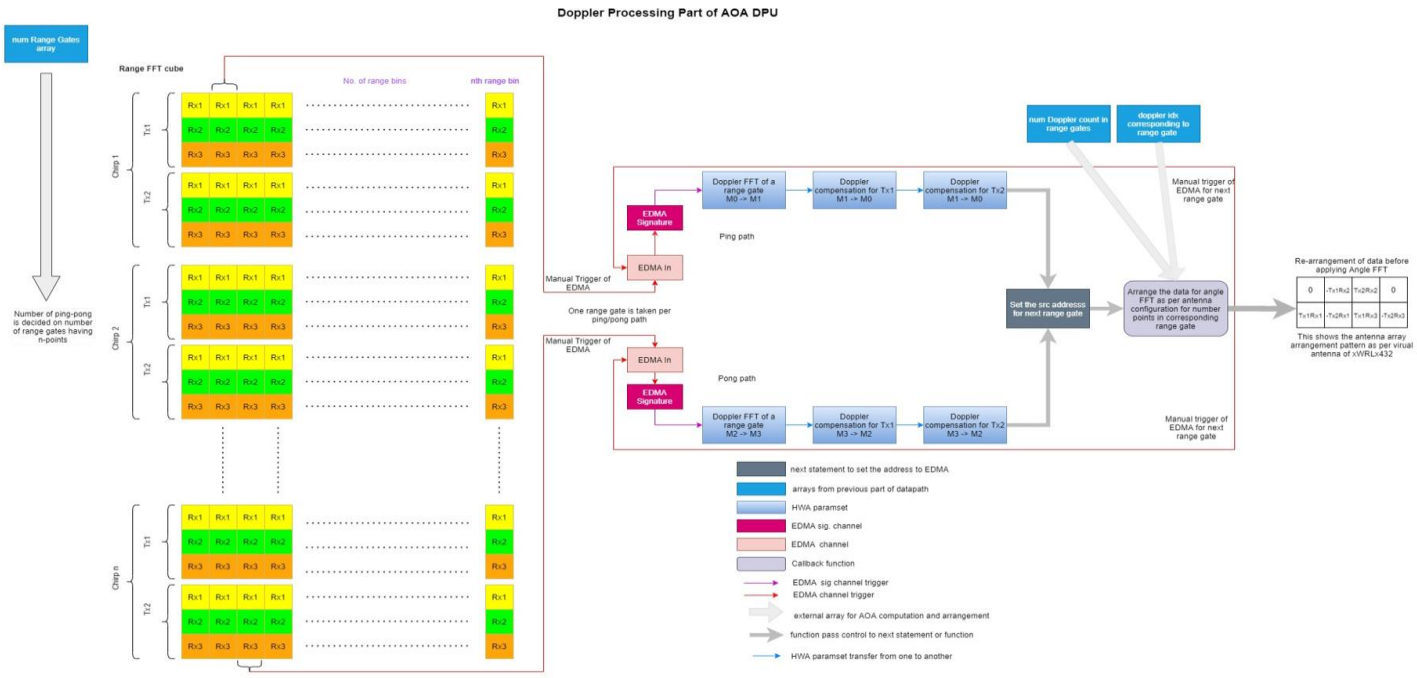


Figure 16 – AOA DPU – detailed diagram Doppler Processing

The Figure 16 elaborates on the Doppler processing part of AOA DPU. Based on which all of the range gates have n-points as calculated from detection matrix, we apply Doppler processing and Doppler compensation (if num of Tx used is > 1). Then move to the Doppler index in the range gate and arrange the antenna as per antenna geometry of the device before applying angle FFT. Antenna geometry is explained in detail in section 5 of [1]. This step of arranging data prior to angle-FFT is done for all detected points in the corresponding range gate. Note that to go to the next range gate, we will need to correspondingly set the source address of EDMA to pick up the next range gate of interest. Also, note that only one range gate is brought in per EDMA transfer in ping/pong path; unlike Doppler DPU where two range gates are taken in ping/pong path. The 3-paramsets used in the ping/pong path does the following operation:

1. Doppler FFT for range gate.
2. Doppler compensation for Tx1 (taken in as such as phase compensation need not be applied here)
3. Doppler compensation for Tx2 (use vector multiplication feature of HWA) to achieve this

4.

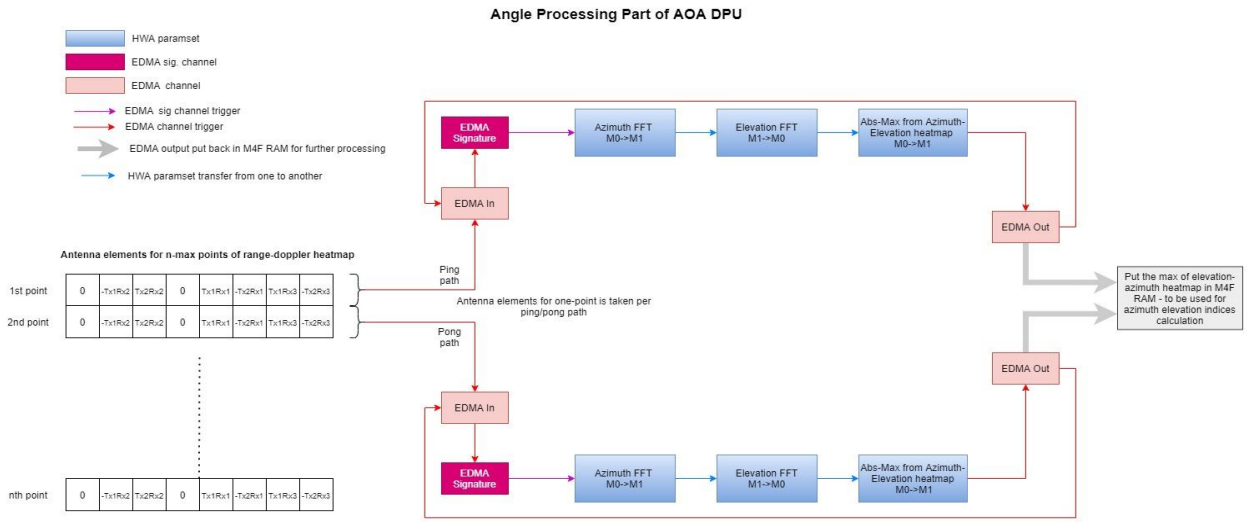


Figure 17 – AOA DPU – detailed diagram AOA part

Figure 17 illustrates the angle estimation portion of the DPU. For each of the n detected points, the azimuth elevation FFT is computed, followed by the index of the bin with the maximum magnitude in the azimuth-elevation heatmap (Max-Abs operation). This will be used for estimating elevation-azimuth indices [*compute_elev_azim_idx()*] to be subsequently used in feature extraction. As shown in the diagram, each ping/pong path processes one of the detected points. The EDMA output is the index corresponding to the max-value of azimuth-elevation heatmap. The 3-paramsets used in the ping/pong path perform the following operations:

1. Azimuth FFT
2. Elevation FFT
3. Max of absolute value from azimuth-elevation heatmap

9.5.1 AOA DPU API

The AOA DPU detection APIs are listed below.

9.5.1.1 DPU_AoAProcHWA_init()

This function allocates memory for the AOA DPU instance and initializes it to zero. It also constructs the semaphores used for processing.

9.5.1.2 DPU_AoAProcHWA_config()

Based on the configuration parameters, this function configures hardware accelerator FFT path and the input and output EDMA channels to bring range gates in HWA memory, and take out max of azimuth-elevation indices. The function is normally called one time before the sensor start command is issued to the RF. In the low power deep sleep mode, the function is called per frame.

9.5.1.3 DPU_AoAProcHWA_Dopplerprocess()

This function does the following operations:

- Sets the common config parameters of HWA like num loops based on number of range gates having maximum of n-points, param start, param stop etc.
- It also sets number of times EDMA in operation is to be performed for Doppler recalculation
- Sets EDMA source address from which range gates have to be taken into HWA (Manual trigger of EDMA is required as range gates may not be continuous for Doppler recalculation).
- Calls the function copyData_after_2DFFT to arrange data in the right order for angle processing
- It also takes care of processing odd number of range gates using ping/pong path by performing one additional ping path computation once.

9.5.1.4 DPU_AoAProcHWA_AoAprocess()

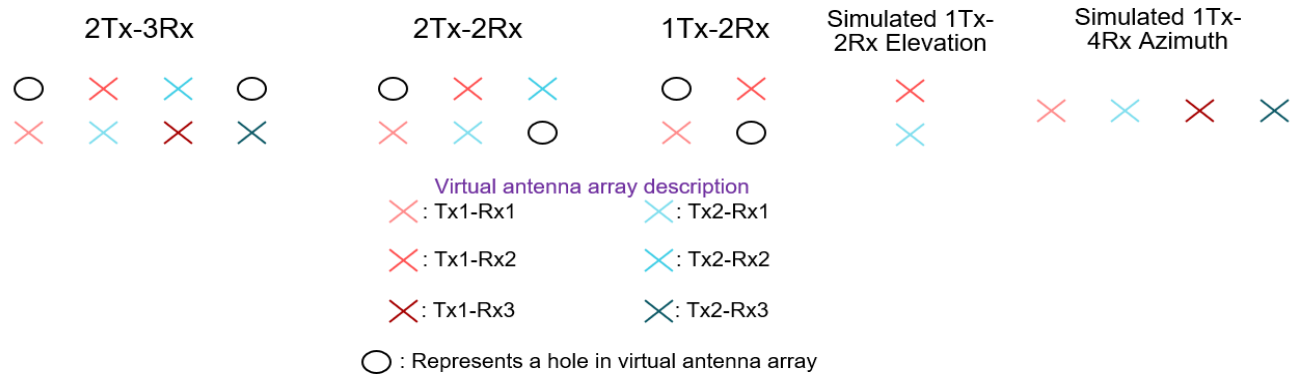
This function does the following operations:

- Sets the common config parameters of HWA like num loops based on n-number of points, param start, param stop etc.
- Triggers the ping and pong EDMA for the azimuth-elevation index calculation
- Pends on the semaphores for the end of the processing as this part of DPU processing is performed by HWA/EDMA in synchronization after 1st EDMA trigger.

9.5.1.5 copyData_after_2DFFT()

This function does the following set of operations:

- Retrieves the number of points in the range gate corresponding to Doppler index for which antenna arrangement is to be done.
- Gets the current range gate index and corresponding Doppler index
- Jump to address where the antenna is arranged in HWA memory
- Phase changes in the virtual array is implemented (this is due to LO phase inversion that happens in the device for Tx2, Rx2).
- This function has multiple antenna arrangements (controlled by MACRO in the code) available including the 2Tx-3Rx(*ANT_2TX_3RX*), 2Tx-2Rx(*ANT_2TX_2RX*), 1Tx-2Rx(*ANT_1TX_2RX*), 1Tx-2Rx_elev(*ANT_1TX_2RX_ELEV* – kind of simulated linear array 1Tx-2Rx for elevation direction), 2Tx-2Rx_Azim(*ANT_2TX_2RX_AZIM* – simulate the linear virtual for azimuth direction) and if linear array is to be exercised *LINEAR_ARRAY* needs to be set to 1.
- The figure depicting the same is given below for each virtual antenna configuration supported.



9.5.1.6 DPU_AoAProcHWA_deinit()

It frees the resources used for the DPU.

9.6 Feature Extraction

The feature extraction operation is a part of library in alg folder (mmwave_lp_sdk\source\alg\featExtract\rangeDopplerBased) of SDK. In the gesture recognition demo, we specifically use *rangeDopplerBased* feature extraction. The features are extracted from range-Doppler heatmap along with elevation and azimuth indices calculated as shown in Figure 18.

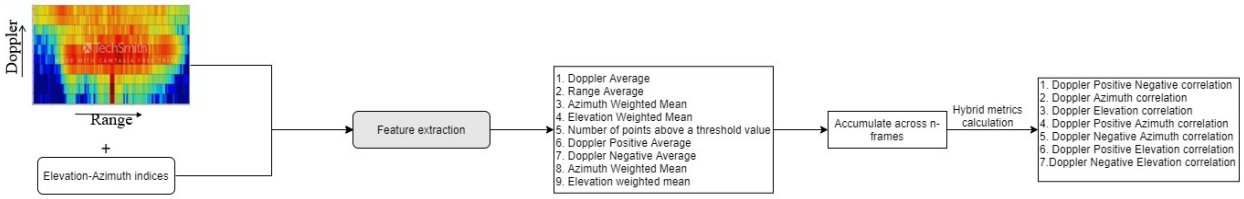


Figure 18 – Feature extraction diagram showing the features extracted by the framework

In this gesture recognition framework, there are a total of 16-features extracted. These are listed below:

- **Doppler Average** (*DopplerAvg*): Weighted average of the Doppler across the heatmap.
- **Doppler Positive Average** (*dopAvgPos*): Weighted average of the Doppler – averaged only across positive Dopplers in the heatmap (region A_+, in Figure 20).
- **Doppler Negative Average** (*dopAvgNeg*$^-$): Weighted average of the Doppler – averaged only across negative Dopplers in the heatmap (region A_-, in Figure 20).
- **Range Average** (*rangeAvg*): Weighted average of the range across the heatmap.
- **Number of detected points** (*numPoints*): number of cells in the heatmap that have a magnitude above a certain threshold.
- **Azimuth Weighted Mean** (*azimWtMean*): Select the N cells of the heat-map with the highest magnitude and for each cell compute the azimuth bin (via angle-FFT). The average azimuth is the weighted average of all these azimuth bin indices.
- **Elevation Weighted Mean** (*elevWtMean*): Select the N of the heat-map with the highest magnitude and for each cell compute the elevation bin (via angle-FFT). The average elevation is the weighted average of all these elevation bin indices.
- **Azimuth Weighted dispersion (***azimWtDisp*): This captures the dispersion (or weighted standard deviation of the cells that were used to compute the Azimuth Weighted Mean.
- **Elevation Weighted dispersion (***elevWtDisp*): This captures the dispersion (or weighted standard deviation of the cells that were used to compute the Elevation Weighted Mean.

- **Doppler Azimuth Correlation** (*dopAzimCorr*): Doppler Azimuth correlation which show how Doppler and Azimuth values are related across the frames. The correlation is calculated by the formula as shown below
- **Doppler Elevation Correlation** (*dopElevCorr*): Doppler Elevation correlation which show how Doppler and Elevation values are related across the frames. The correlation is calculated by the formula as shown below
- **Doppler Positive Negative Correlation** (*dopPosNegCorr*): Doppler Positive Negative correlation which show how Doppler Positive and Negative values are related across the frames. The correlation is calculated by the formula as shown below
- **Doppler Positive Azimuth Correlation** (*dopPosAzimCorr*): Doppler Positive Azimuth correlation which show how Doppler Positive and Azimuth values are related across the frames. The correlation is calculated by the formula as shown below
- **Doppler Positive Eelvation Correlation** (*dopPosElevCorr*): Doppler Positive Elevation correlation which show how Doppler Positive and Elevation values are related across the frames. The correlation is calculated by the formula as shown below
- **Doppler Negative Azimuth Correlation** (*dopNegAzimCorr*): Doppler Negative Azimuth correlation which show how Doppler Negative and Azimuth values are related across the frames. The correlation is calculated by the formula as shown below
- **Doppler Negative Eelvation Correlation** (*dopNegElevCorr*): Doppler Positive Elevation correlation which show how Doppler Positive and Elevation values are related across the frames. The correlation is calculated by the formula as shown below

9.6.1 Compensation of doppler and range bins

The few of the bins near the zero doppler (in the demo its fixed to 5 – *NUM_ZERO_DOPPLER_BINS*) are not taken into consideration (for computation of n-points with highest magnitude in detection matrix and in feature extraction part) to remove static objects and leakage of the values from zero doppler. Similarly, for range bins this is done for those which are chosen by the user

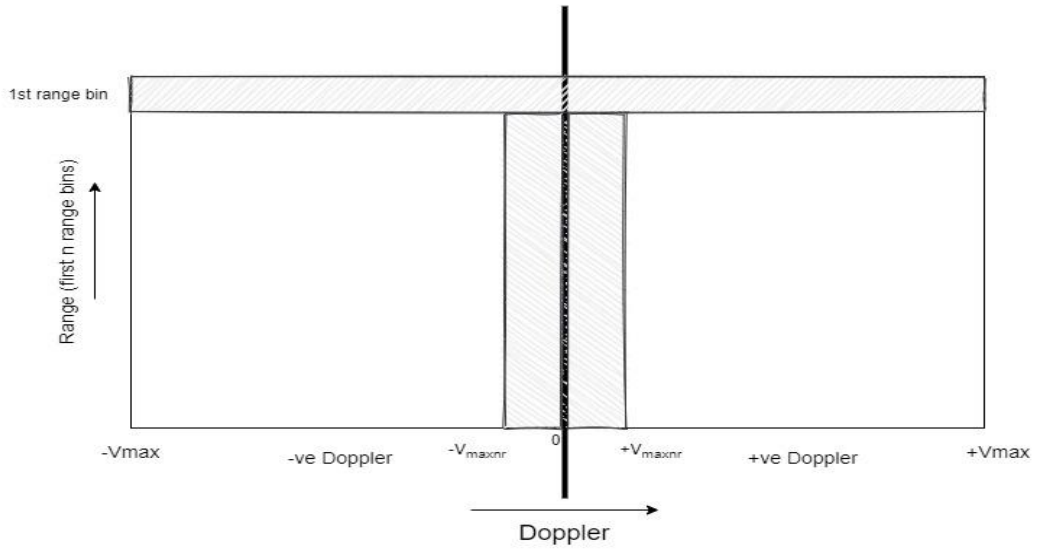


Figure 19 – Figure showing how few range bins/Doppler bins at the edges are dropped to prevent leakage

As shown in the figure above, the Doppler bins near to zero Doppler from $-V_{maxnr}$ to $+V_{maxnr}$ along with the 1st range bin is zeroed out and not used to find the n-points with highest magnitude.

9.6.2 Feature Extraction API

9.6.2.1 featExtract_malloc()

Has functionality to allocate and free heap for the resources used in feature extraction library.

9.6.2.2 featExtract_create()

This is the API that configures the feature extraction library with parameters needed by the library to perform feature extraction. This includes:

- setting number of range bins, Doppler bins, minimum and maximum of range and Doppler bins, max of n-points from RD (range-Doppler) map
- initializing the feature computation strange for hybrid parameters calculation which include correlation
- checking if the config input by the user exceeds the limits set

9.6.2.3 featExtract_compute()

This is the API that is called from the application for feature extraction computation. This API internally calls three functions:

- *computeFeatures_rangeDoppler() :* This is used to compute Range/Doppler related features.
- *computeFeatures_azimElev() :* This is used to compute Azimuth and elevation related features
- *computeFeatures_hybrid():* This is used to create features that depend on both angle(Azimuth/Elev) and Range/Doppler.

Thus, using this API, we extract the 16 features mentioned in the previous section

9.6.2.4 featExtract_delete()

Deletes and frees all the resources used for the feature extraction library.

9.7 Gesture Classifier

The gesture classifier (*gestureClassifier*) library is part of the alg folder (mmwave_lp_sdk\source\alg\classifier\gestureClassifier) in SDK. The gesture classifier uses an ANN (Artificial Neural Network) for gesture classification. In addition to the ANN, the library implements some pre-processing steps to normalize the data (currently Standard Normalization). There is also some post processing done on the output of the ANN in order to improve robustness. Since radar processing happens across frames, features are accumulated across a programmable number (n) of consecutive frames before presenting them to the ANN input. Thus, ANN's first output is given out after a couple frames from which sliding window approach is adopted

The overall flow after feature extraction is shown in the below Figure 20(for a 2-gesture use case with num-features=3 and numFrames = 15 with two hidden layers):

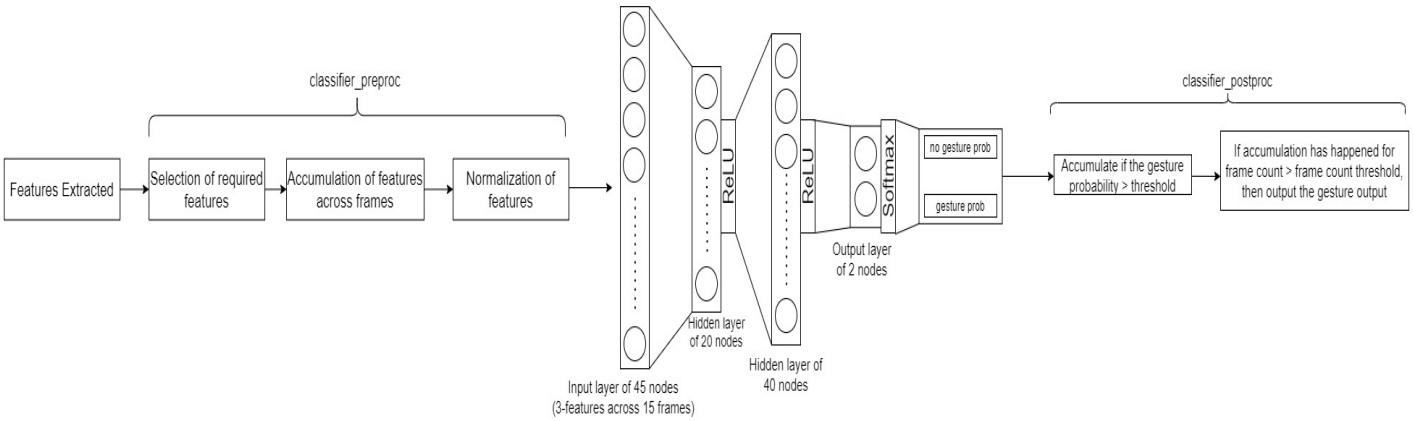


Figure 20 – Classification process with classifier, pre-proc and post-proc

9.7.1 Gesture Classifier API used in gesture recognition application

9.7.1.1 Classifier_prep_proc()

This is a helper function before giving input to the ANN classifier. It does the following operations:

- Selects the features to be used in the demo
- Arranges the data as per requirement for ANN
- Normalizes the data as needed (standard normalization - similar to what is done for training)
- Implements the sliding window approach for presenting data to the ANN.

9.7.1.2 Classifier_predict()

This function takes the input set in a particular format by the classifier_prep_proc and gives the output of probabilities computed by ANN. This API is used by the *gestureClassifier* library for ANN probability computation. This API invokes function from *classifier_layers* for prediction purpose. The library currently supports 2 or 3- hidden layers only.

9.7.1.3 Classifier_post_proc()

This function does post-processing of probabilities output by the classifier to improve robustness and performance. The following operations are performed by this function:

- It keeps track of *classifierMatrix* (ANN output for gestures across frames) which is used in post-processing.
- There is a count that is incremented if the gesture probability from ANN is greater than threshold probability
- Sets the classifier output if the count is accumulated in *classifierMatrix* over a period of frames (programmable by user).

9.7.2 Gesture Classifier API for library

9.7.2.1 Classifier_malloc()

This function allocates heap to be used by *gestureClassifier*

9.7.2.2 *Classifier_create()*

This function configures the ANN library where it sets the following:

- *Num_frames* set by user after which 1st prediction probability comes out, *num_features* to be used, *num_classes* at ANN output.
- Allocates the internal buffer to store intermediate outputs of the ANN

9.7.2.3 *Classifier_layers()*

This function has all the list of layers, activation functions used by the ANN for output probability prediction. This function has implementation of:

- Dense layer
- ReLU activation function
- Softmax function used at the output
- Approximate exponential function (reduces compute of CNN with minimum tradeoff in accuracy)

9.7.2.4 *Classifier_delete()*

This function deletes and frees up the resource used by the *gestureClassifier.*

10 Presence Mode to Gesture Mode Configuration switch

The presence to gesture mode chirp parameter switching and vice-versa is handled in the *powerManagmentTask*. In the low-power mode enabled case, the FECSS (Front-End Configuration Sub-System) shuts down every frame on completion of chirping. Subsequently after inter-frame processing is completed, the device goes into a Low Power Deep Sleep (LPDS) mode. Thus, the FECSS needs to be re-configured after the device wakes up from deep sleep for the chirping to happen for the next frame. The function *CLI_MMWStart()* is the function that reconfigures the FECSS and creates the task for the next frame.

For mode switching, we make use of the reconfiguration of FECSS available in the *powerManagementTask*. This is achieved by passing the relevant parameters to the struct that sets the FECSS for chirping as the device wakes up from the deep sleep. This is portrayed in the Figure 21 shown below:

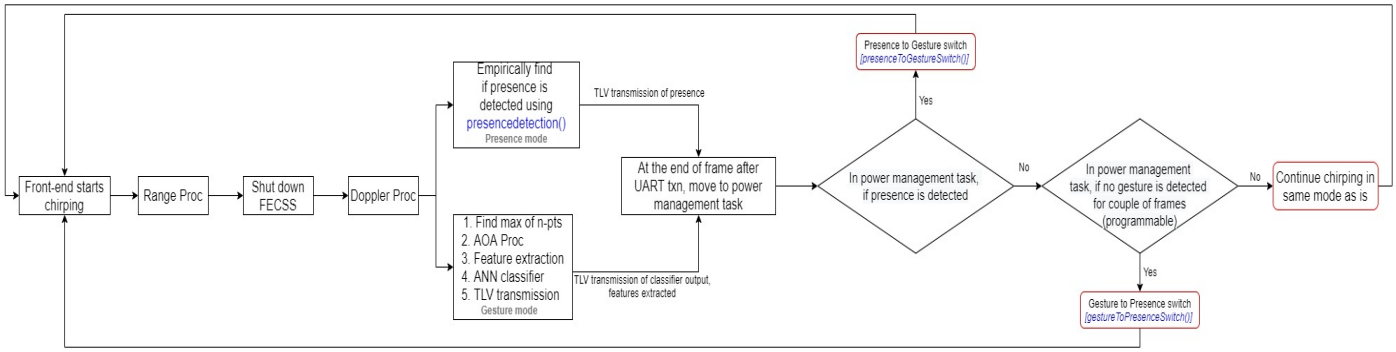


Figure 21 – Chirp config switch from presence to gesture and vice-versa

The config switch as portrayed in the figure is done for the below conditions:

- Switching from presence to gesture mode only if presence is detected using the function *presenceToGestureSwitch().*
- Switching from gesture back to presence only if no gesture is detected for couple of frames (programmable based on the #def variable *NUM_FRAMES_GESTURE_TO_PRESENCE1*) using the function *gestureToPresenceSwitch()*. Based on the value set this switch can be increased/decreased as per user convenience and wish.
- If none of the above conditions are satisfied, the conditions for switching are not met and the mode in which the application is currently continues.

11 System execution flow

11.1 Initialization and Configuration

The system execution flow at the initialization and configuration time is shown in Figure 22. At the startup the system initialization is performed, the CLI task is created, waiting for the input CLI commands. After receiving the CLI **sensorStart** command, the last command in the configuration sequence, it configures the RF and then creates the demo DPC task, (`mmwDemo_dpcTask`). As the DPC task has higher priority than the CLI task, it preempts the CLI task, and performs the configuration of the signal processing chain components, and calls the range processing API as a first processing DPU in the chain. The range processing function starts pending on the semaphore waiting for the first frame chirping and the chirp processing time to complete. The task control is then switched back to CLI task that creates the UART transmit task, (mmwDemo_TransmitProcessedOutputTask), and issues the sensor start command to the RF. The CLI task then enters the pending state and from this moment on it stays in it.

The power management task, that is also created at the start time is engaged only in the power saving mode.

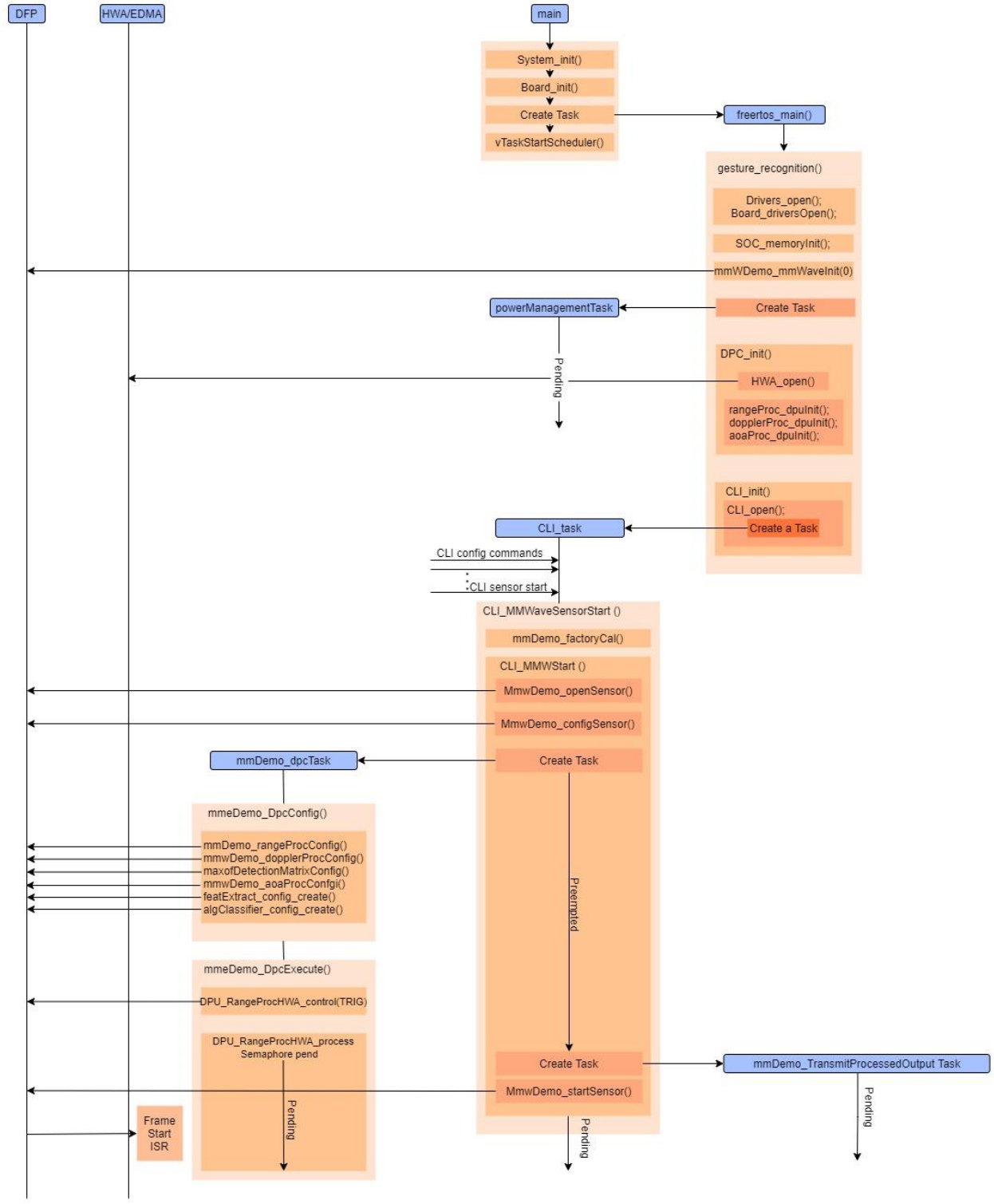


Figure 22 – Initialization and configuration

11.2 Steady state – low power mode disabled

The steady state system execution flow is periodic, executed per frame. Figure 23 shows the system execution flow over one frame period. The frame ISR function is triggered upon the frame start event, and currently it is only used for diagnostic purpose. During the chirping time, the HWA is performing the chirp processing, while the range DPU process function is waiting for the HWA processing to complete. On completion of chirping, the Doppler DPU followed by AoA DPU, Feature Extraction and Gesture Classifier is run creating the feature vectors and classifier output. The classifier output along with extracted features are ready to be sent out to the host. The HWA is then configured for the next frame, a semaphore is posted to the UART task indicating that the current frame data is ready, and the range processing API is called for the next frame. The DPC task goes to pending state giving control to the UART task to send data to the HOST.

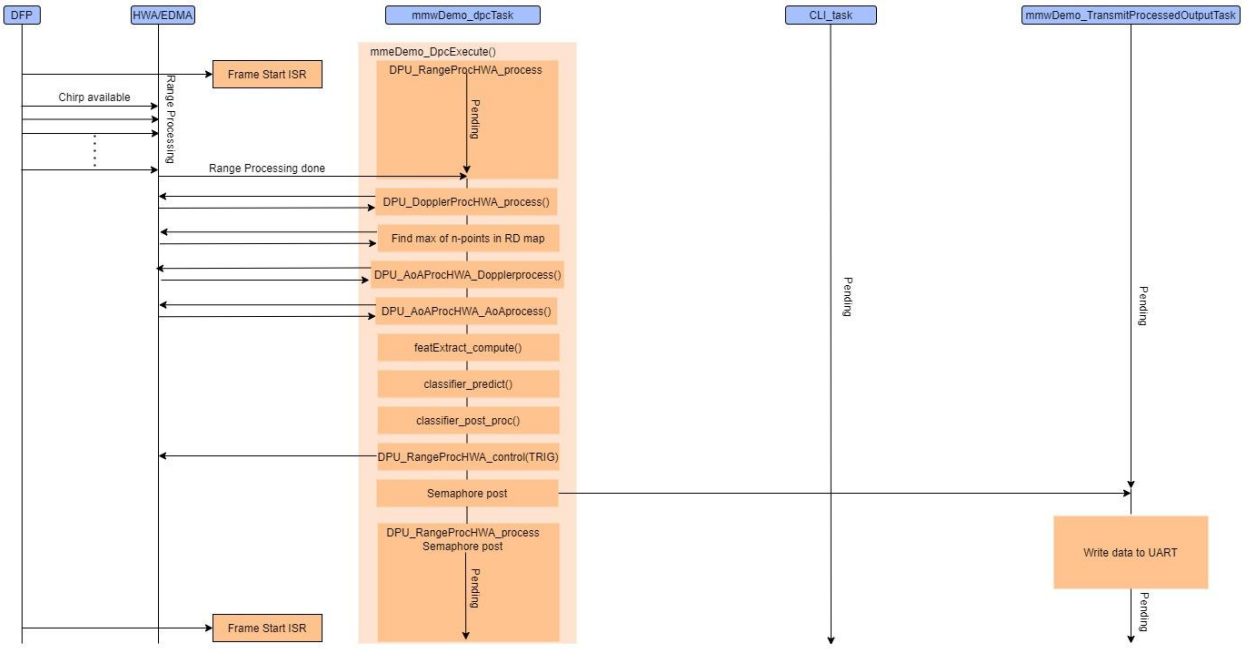


Figure 23 – System execution per frame when low power mode is disabled

11.3 Steady state – low power mode enabled

The steady state system flow in low power mode is shown in Figure 24. In the low power mode the powerManagementTask() controls the power savings. When the UART task completes the data transfer to the Host of the frame n, it calculates the available time for the low power deep sleep phase as

$$T_{sleep}(n) = T_{Frame} - T_{Demo}(n) - T_{lpdsLatency}$$

and if $T_{sleep}(n) > 0$ it posts the semaphore to the power management task. The power management task places the part of the SOC into the low power deep sleep (LPDS) mode for the period of $T_{sleep}(n)$.

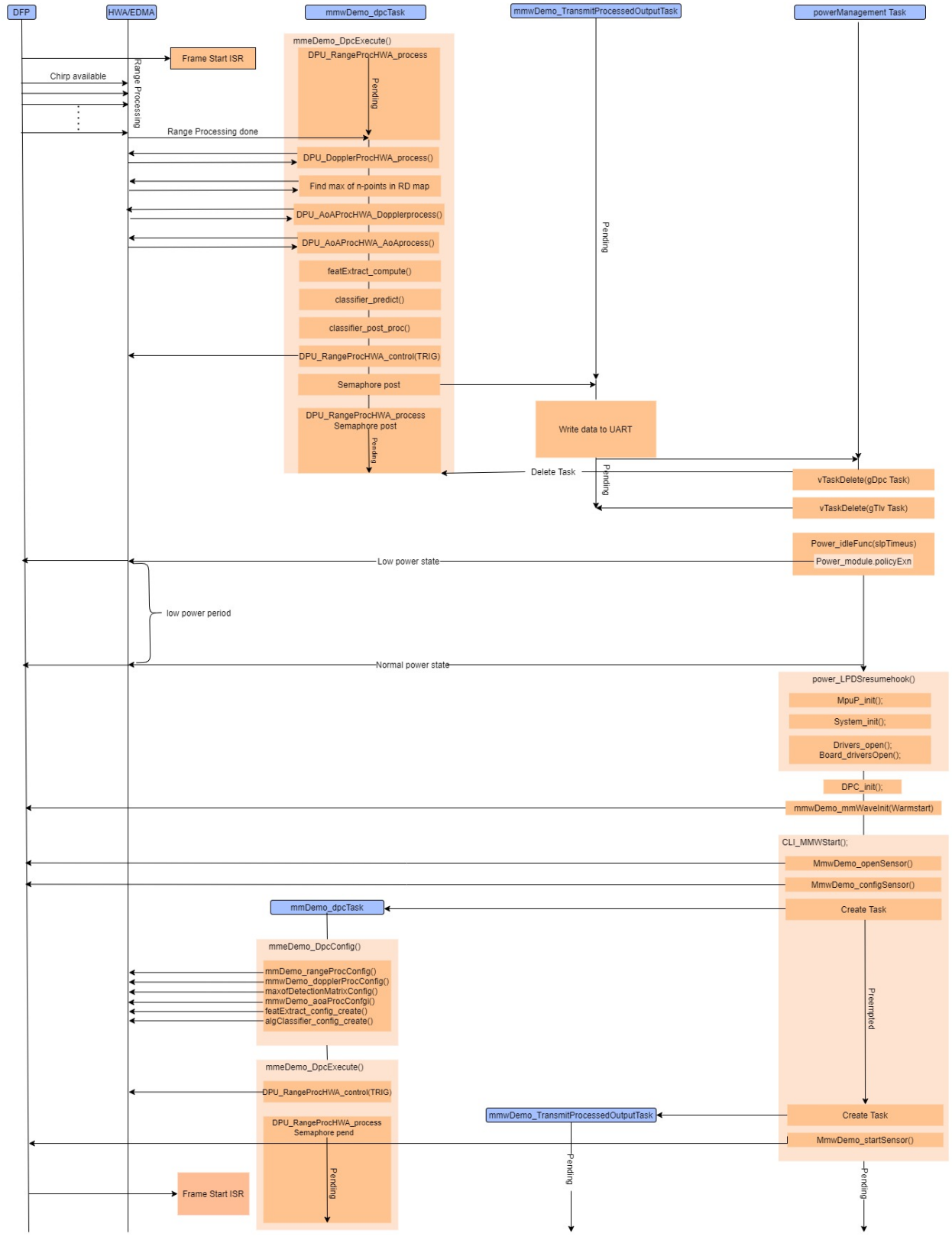


Figure 24 – System execution per frame when low power mode is enabled

11.4 Steady State – presence detection mode (Low power is always enabled)

This section describes the task model of presence detection mode. In this mode, on completion of Range and Doppler FFT, a metric (sum of the magnitudes across range bins in the non-zero Doppler bin) is calculated to check for presence. The config switch to Gesture mode happens only if presence is detected. The Figure 25 below shows the tasks and functions invoked to achieve the same:

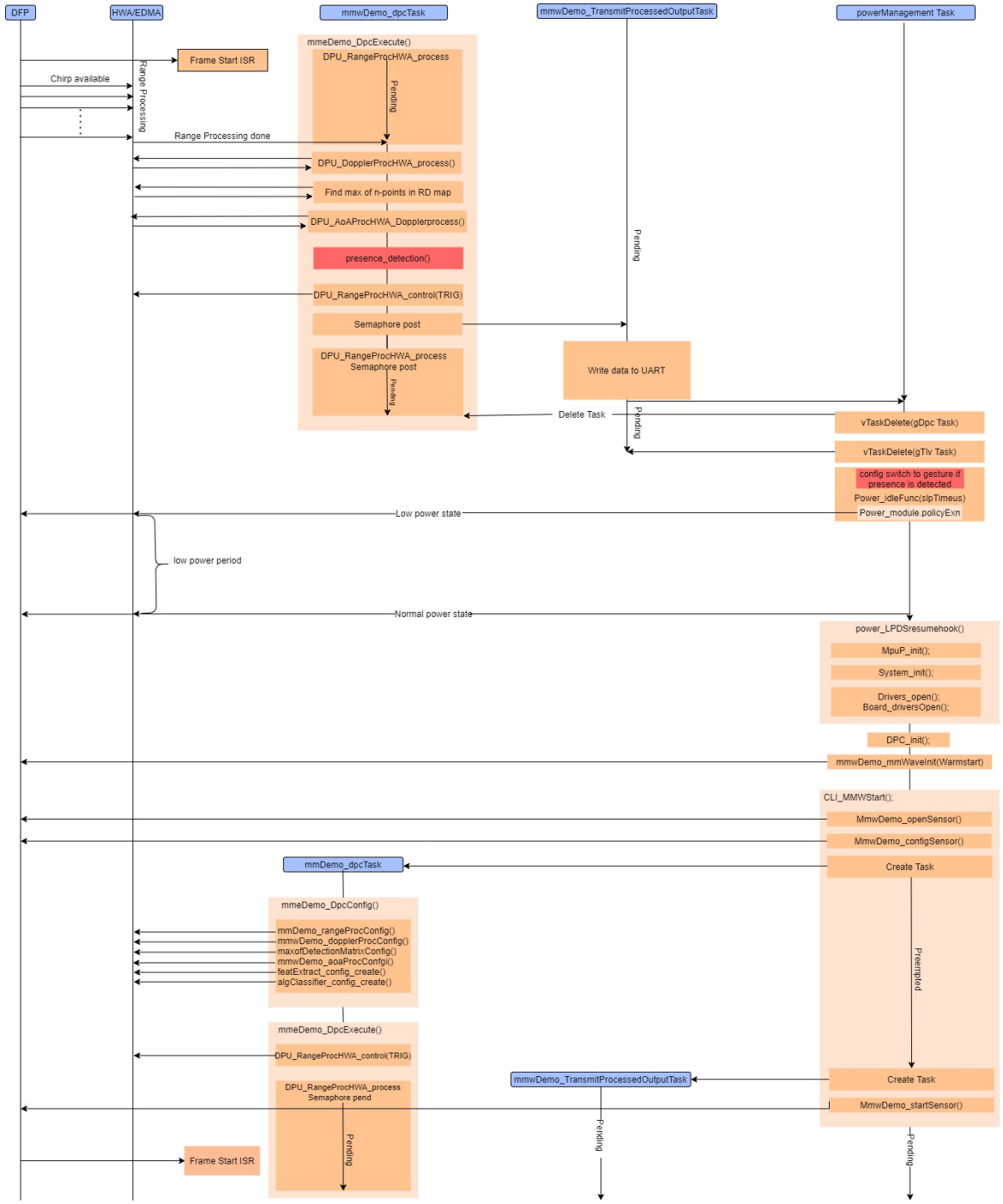


Figure 25 – System execution per frame when presence detection mode is enabled

12 Task Model

Gesture Recognition demo is implemented on xWRLx432 using multiple tasks running in the system. Table 1 list all tasks used in the system.

Task	Name in the code	Priority
Main freertos task	freertos_main()	15
Power management task	powerManagementTask()	2
Command line interface task	CLI_task()	1
Detection processing chain (DPC) task	mmwDemo_dpcTask()	5
Transmit data to host (UART) task	mmwDemo_TransmitProcessedOutputTask()	3
Task only for testing purpose	mmwDemo_adcFileReadTask()	4

Table 1 – Gesture recognition demo tasks

12.1 Main Task

This is free RTOS main task initially called by the main function. It is active only during the start time, and after the creation of the CLI task it rests in pending state.

12.2 CLI Task

The CLI task provides the execution context for the command line interface. It includes simple command parser.

12.3 DPC Task

The DPC task provides the execution context for the processing chain (Presence Detection and Gesture Modes).

12.4 UART Task

This task controls the transfer of radar detection data to the host. The task transmits one packet of data per radar frame using a TLV format structure. Depending on the CLI configuration the task sends point cloud, range profile, detection matrix, presence detection information, feature vectors, classifier output, etc.

12.5 ADC read data Task

This task is used for running the unit tests for motion detection demo. The task reads reference ADC data samples from the file and feeds the signal processing chain instead of real ADC data coming from the RF (see details in Section 16).

12.6 Power Management Task

This task is engaged when the motion detection demo is running in the power saving deep sleep mode.

12.7 Timing Diagram

The timing diagram during the startup and the first two frames is illustrated in Figure 26.

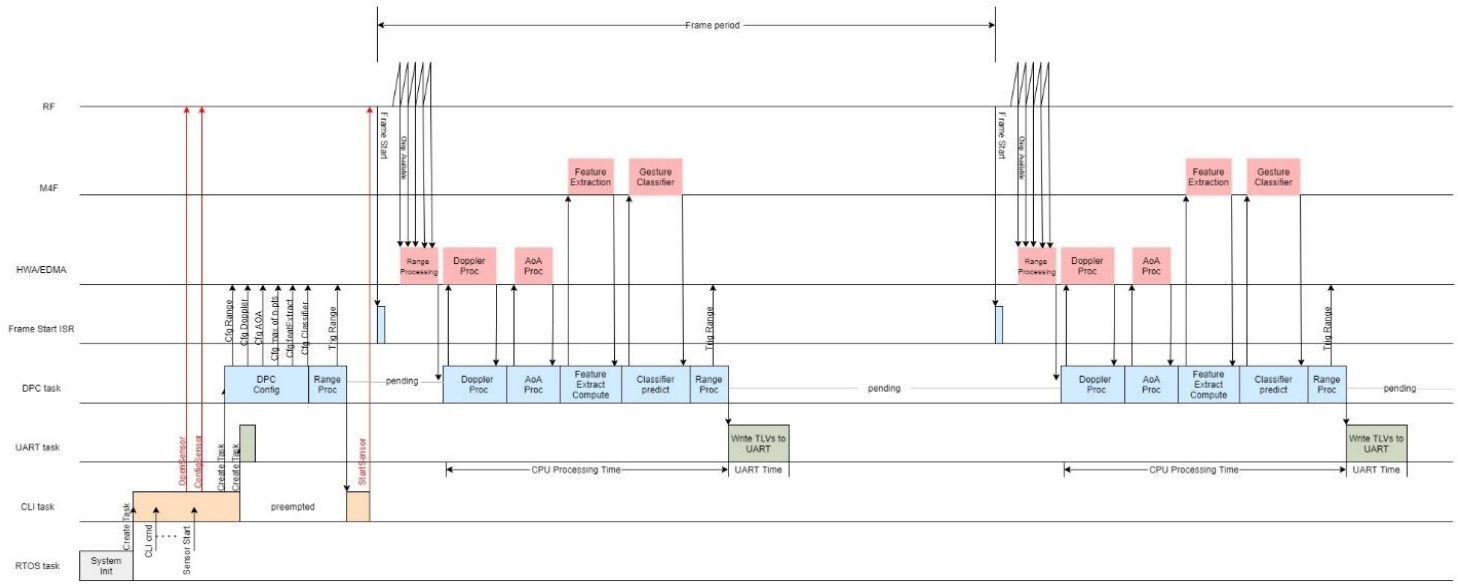


Figure 26 – Timing diagram – low power mode disabled

The timing diagram in the low power mode is shown in Figure 27.

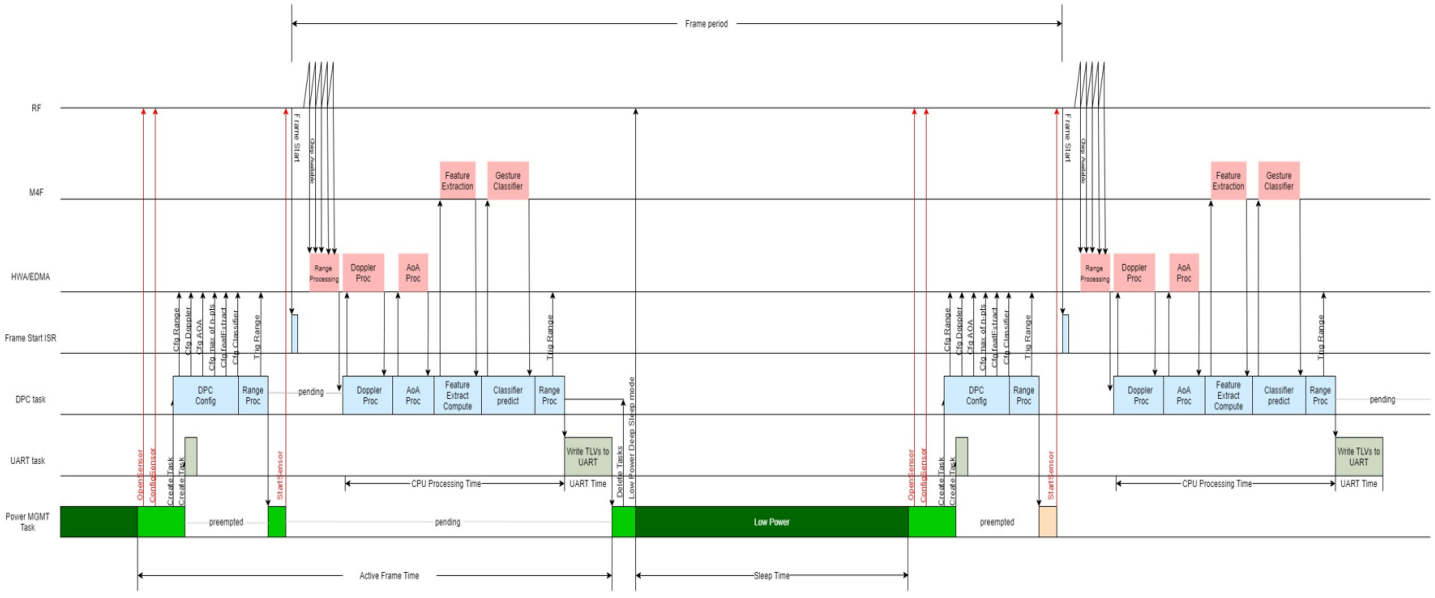


Figure 27 – Timing diagram – low power mode enabled

The **timing diagram in the presence detection mode** will be similar to the above diagram except for the fact that after Doppler Proc, there will be presence detection function called which will check if the presence is detected and then go into the low power deep sleep mode before switching to gesture mode (if presence is detected) or wake up back in the presence mode itself.

13 Memory Usage

The OOB gesture recognition demo memory usage can always be found in the MAP file. The current usage is shown in Table 2.

Section	Size(Bytes)
.vectors	320
.bss	42496
.text	123762
.rodata	14413
.stack	8192
.sysmem	8192
.data	53305

Table 2 – Gesture recognition OOB demo memory usage

Some of the data arrays are dynamically handled using a set of simple memory allocation utility functions, provided in Table 3.

Function	Description
DPC_ObjDet_MemPoolAlloc()	allocates memory from the given memory pool
DPC_ObjDet_MemPoolGetMaxUsage()	reports the memory usage for the given memory pool
DPC_ObjDet_MemPoolSet()	sets the free memory pointer to a given value. This is useful for shared scratch memory between DPUs
DPC_ObjDet_MemPoolReset()	Resets the free memory pointer to the beginning of the given pool.

Table 3 – Memory allocation functions

Two memory heaps are created, one in the local core memory, gMmwCoreLocMem[40*1024] , and the other, gMmwL3[256K], occupying the whole 256KB of the APSS shared memory. Currently less than one 1kB of memory is allocated in the local core RAM. The usage of the shared memory depends on the configuration. At the start time, after the CLI commands has been received and configuration completed, memory usage of these two heaps is printed out on the console.

Thu current usage of memory for the gesture recognition demo configurations, provided in the SDK, is shown in Table 4 and Table 5 for the gesture and presence mode respectively.

Gesture Mode	
Input	**Value**
Number of ADC samples (real)	128
Range FFT size	128
Number of range bins (real)	64
Number of chirps	128
Number of Doppler bins	128
Number of TX antennas	2
Number of RX antennas	3

Number of virtual antennas	6
Number of azimuth bins	32
Number of azimuth rows in the antenna pattern	2
Number of elevation bins	32
Max number of points from detection matrix	25

Gesture Mode	
APPSS Shared Memory	**Size (Bytes)**
Radar Cube	196608
Angle Matrix data	1600
Total	198208

Local Core Memory	**Size (Bytes)**
Range FFT Window	256
Doppler FFT window	256
Detection Matrix	32768
Vector RAM coefficients	1024
Max Val of elevation-azimuth heatmap	100
Range gates count (Doppler recalculation)	2
Range_idx (intermediate while finding max)	50
Range_idx_arr(final value for Doppler recalculation)	50
Angle_idx_Doppler_count (intermediate while finding max)	50
Angle_idx_Doppler_count_arr(final value for Doppler recalculation)	50
Doppler_idx_arr	1500
Antenna_array	64
Elev_idx (used for feature extraction)	50
Azim_idx (used for feature extraction)	50
detMatrixMaxIndex (used for feature extraction)	100
Feature output	64
Classifier Input	180
Classifier Probability	8
Classifier Matrix	20
Classifier Output	1
Total	36643

Table 4 – Memory usage – dynamic allocation in gesture mode

Presence Mode	
Input	**Value**
Number of ADC samples (real)	128
Range FFT size	128
Number of range bins (real)	64
Number of chirps	2
Number of Doppler bins	2
Number of TX antennas	2
Number of RX antennas	3
Number of virtual antennas	6

Presence Mode	
APPS Shared Memory	**Size(Bytes)**
Radar Cube	512
Detection Matrix	512
Total	1024

Local Core Memory	**Size(Bytes)**
Range FFT Window	128
Total	128

Table 5 – Memory usage - dynamic allocation in presence mode

14 Benchmarks

Currently the CPU time and the UART transmit time is reported to Host within the stats TLV (*gMmwMSSMCB.outStats*). For the default chirp config chosen for the gesture mode as mentioned in section 5, the signal processing chain time with classifier takes around 5.8ms.

15 UART and Output to the Host

15.1 Output TLV Description

The packet structure consists of fixed sized frame header, followed by variable number of TLVs (see Figure 28). Each TLV has fixed header followed by variable size payload. The Byte order is Little Endian.

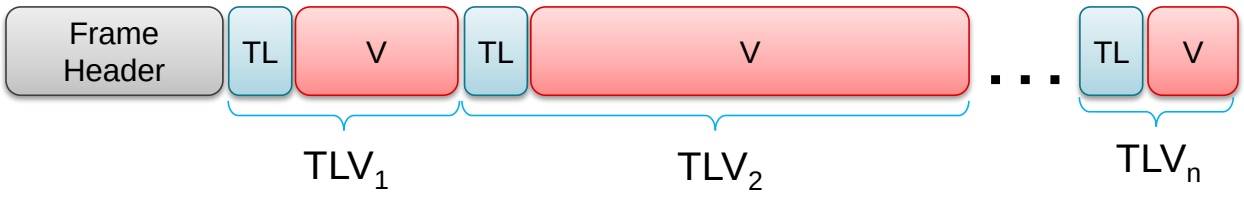


Figure 28 – Data packet structure sent to Host.

15.1.1 Frame Header Structure

The frame header is of fixed size (52bytes). It is defined by the structure as

```
typedef struct MmwDemo_output_message_header_t
{

    uint16_t    magicWord[4];       /* Sync word: {0x0102,0x0304,0x0506,0x0708} */
    uint32_t    version;            /* MajorNum*2^24+MinorNum*2^16+BugfixNum*2^8+BuildNum */
    uint32_t    totalPacketLen;     /* Total packet length including header in Bytes */
    uint32_t    platform;           /* platform type */
    uint32_t    frameNumber;        /* Frame number */
    uint32_t    timeCpuCycles;      /* Time in CPU cycles when the message was created */
    uint32_t    numDetectedObj;     /* Number of detected objects */
    uint32_t    numTLVs;            /* Number of TLVs */
    uint32_t    subFrameNumber;     /* Subframe number */
} MmwDemo_output_message_header;
```

15.1.2 TLV Structure

The TLV structure consists of a fixed header, TL (8bytes) followed by TLV specific payload. The TLV header structure is shown below.

```
typedef struct MmwDemo_output_message_tl_t
{
    uint32_t    type;           /* TLV type */
    uint32_t    length;         /* Length in bytes */
} MmwDemo_output_message_tl;
```

15.1.3 Range Doppler Features TLV

Gesture Recognition demo outputs the features extracted using the azimuth-elevation indices in conjugation with range-Doppler heatmap. The range Doppler features are streamed out in the floating-point format with order as shown in the structure below:

```
typedef struct
{
    float rangeAvg;
    float DopplerAvg;
    float DopplerAvgPos;
    float DopplerAvgNeg;
    float numPoints;
    float azimWtMean;
    float elevWtMean;
    float azimWtDisp;
    float elevWtDisp;
    float dopAzimCorr;
    float dopElevCorr;
    float dopPosNegCorr
    float dopPosElevCorr
    float dopPosAzimCorr
    float dopNegElevCorr
    float dopNegAzimCorr
} FeatExtract_featOutput;
```

Length = `sizeof(FeatExtract_featOutput);`

15.1.4 Classifier Output TLV

This TLV contains output the gesture classified by the demo. In gesture recognition demo it sends out the gesture number encoded by the user in the demo.

Type = MMWDEMO_OUTPUT_EXT_CLASSIFIER_OUTPUT

Length = `sizeof(uint8_t)`

15.1.5 Presence Output TLV

This TLV contains the presence output sent out in the presence + gesture mode else this TLV is not sent out in normal gesture mode.

Type = MMWDEMO_OUTPUT_EXT_PRESENCE_OUTPUT
Length = `sizeof(uint8_t)`

15.1.6 Presence Detection Threshold TLV

This TLV contains the sum of magnitude across 2nd Doppler bin from range bin 'x' to range bin 'y' in the presence mode.

Type = MMWDEMO_OUTPUT_EXT_PRESENCE_DETECT_THRESHOLD
Length = `sizeof(uint32_t)`

15.1.7 Stats TLV

This TLV contains statistics described in the structure below.

Type = MMWDEMO_OUTPUT_EXT_MSG_STATS
Length = `sizeof(MmwDemo_output_message_stats)`

```
typedef struct MmwDemo_output_message_stats_t
{
   uint32_t interFrameProcessingTime; /* Interframe processing time in usec */
   uint32_t transmitOutputTime; /* Transmission data transmit time in usec */
   uint16_t powerMeasured[4]; /* Power at 1.8V, 3.3V, 1.2V and 1.2V RF rails
                                 (1LSB = 100 uW) */
   int16_t tempReading[4]; /* Temperature: Rx, Tx, PM, DIG. (ºC), 1LSB = 1ºC */
} MmwDemo_output_message_stats;
```

16 Running in test mode

For testing the signal processing chain, the source for the ADC samples can be configured to be the input file, instead of RF input stream, as shown in Figure 29. This mode is enabled using a CLI command adcDataSource which specifies the input binary ADC data file (see [1]). Note that this mode can run when the code is loaded through CCS. In this mode the RF part of the SOC is not configured.

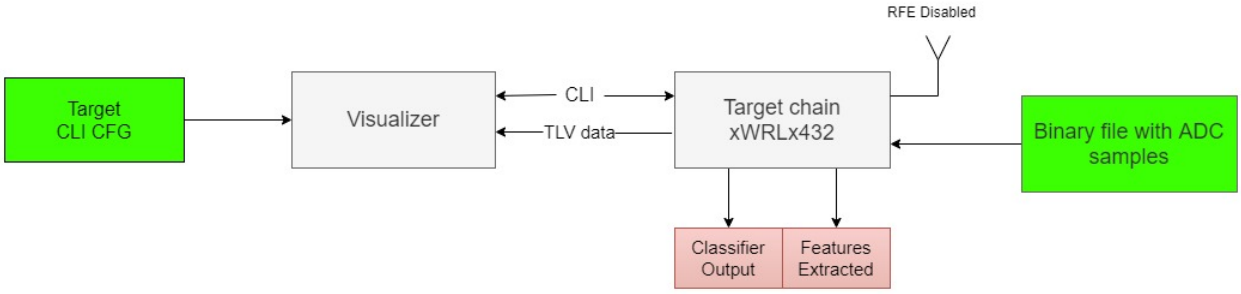


Figure 29 – Reading ADC data from file

In this mode demo is controlled form the ADC read data task mmwDemo_adcFileReadTask(), that reads data from the file, and triggers the input EDMA of the range DPU. This task also saves the classifier output, probabilities of the ANN classifier and the features extracted from range-Doppler heatmap to binary files.

The timing diagram is illustrated in Figure 30.

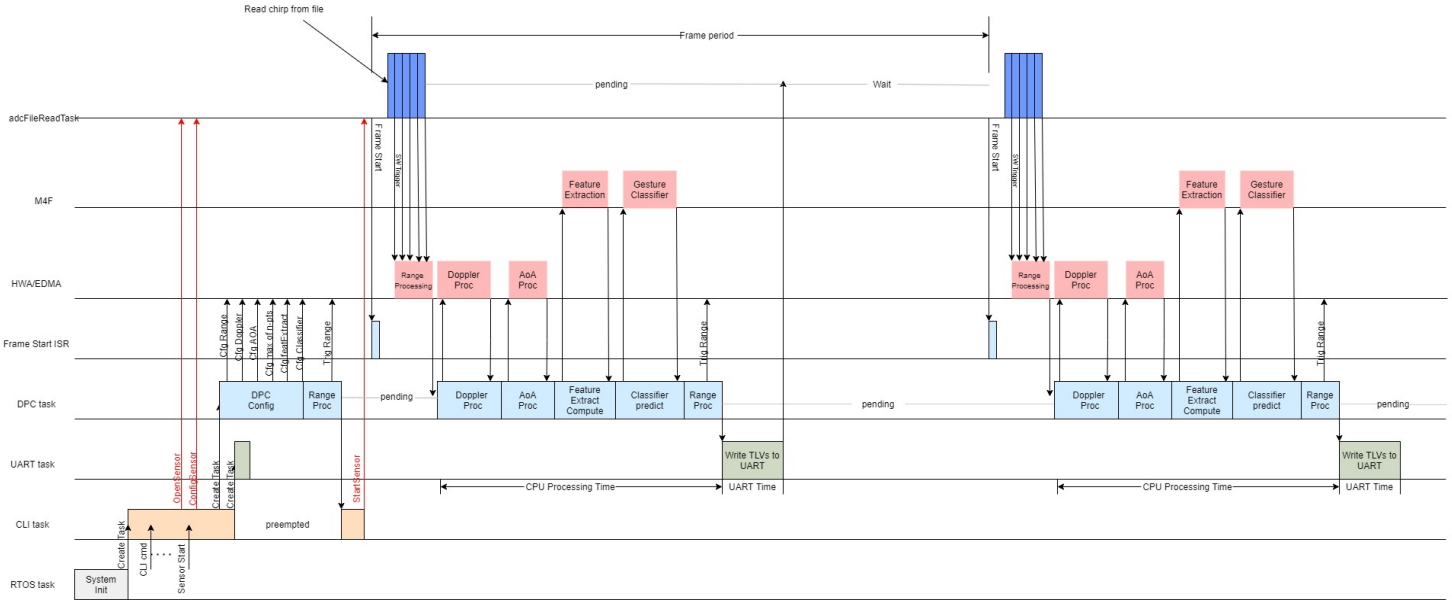


Figure 30 – Timing diagram in demo test mode - ADC samples read from file.

17 Kick-2-Open demo specific details

- The Kick-2-Open demo uses 2Tx-2Rx hence the demo radar cube size is reduced by one-third and it becomes 128KB, similarly reduction in angle matrix cube is also there.
- The features used for Kick-2-Open demo for training/testing will be *rangeAvg*, *dopplerAvg* and *elevWtMean* with 15frames taken at a time in sliding window fashion.
- The Kick-2-Open demo uses 2-layer ANN with 20, 40 nodes in hidden layer 1, hidden layer 2 respectively and has 2-nodes in output layer (refers to kick and no-kick).
- The presence detection region is set from 0.25m to 2.25m.

- The kick detection region is set from 0.25 to 1.6m. If kicks need to be detected outside this zone, data collection and re-training for the same is required.
- For running the Kick-2-Open demo on xWRL1432, the start frequency must be set to 76GHz while on xWRL6432, this has to be set to 59GHz.
- The maxNumPoints in the CLI is set to 25 for the kick-2-open demo.
- The K2O demo has been trained for height kept at 40-45cm (from ground) to the center of chip in TI's EVM.

18 References

[1] Motion/Presence Detection demo documentation, Texas Instruments, April 2022.
[2] Motion/Presence Detection Tuning Guide, Texas Instruments, April 2022.
[3] Radar hardware accelerator user guide, SWRU526b, October 2018.
[4] TMS320C6472/TMS320TCI648x DSP Enhanced DMA (EDMA3) Controller, User's Guide, SPRU727E, July 2011.
[5] E. Jacobsen and P. Kootsookos, "Fast, Accurate Frequency Estimators [DSP Tips & Tricks]," in IEEE Signal Processing Magazine, vol. 24, no. 3, pp. 123-125, May 2007