MMWAVE MCUPLUS SDK User Guide

Product Release 4.2

Release Date: March 28, 2022

Document Version: 2.1

DOCUMENT LICENSE

COPYRIGHT

DISCLAIMER

This mmWave SDK User guide is generic and contains details about all the mmWave devices that are supported by TI in general. However, note that not all mmWave devices may be supported in a given mmWave SDK release. Please refer to the mmWave SDK Release notes to understand the list of devices/platforms supported in a given mmWave SDK release.

⚠ **NOTICE:** This software product is used to configure TI’s mmWave devices, including RF emissions parameters for such devices. Note that many countries or regions impose regulations governing RF emissions. Users are responsible for understanding local RF emission regulations and operating the product within those regulations.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. Out-of-box mmWave Experience

To experience the mmWave technology offered by TI, you will need to procure the following

- Hardware
  1. mmWave TI EVM
  2. Power supply cable as recommended in TI EVM user guide
  3. PC
- Software
  1. Pre-flashed mmWave Demo running on TI EVM (See instructions in this user guide on how to update the flashed demo)
  2. Chrome browser running on PC

Next, to visualize the data flowing out of TI mmWave devices, follow these steps

1. Connect the EVM to a power outlet via the power cable and to the PC via the included USB cable. EVM should be powered up and connected to PC now.
2. On your PC, browse to https://dev.ti.com/gallery/view/mmwave/mmWave_Demo_Visualizer/ver/4.2.0/ in Chrome browser and follow the prompts to install one-time software. [No other software installation is needed at this time]
3. The Visualizer app should detect and connect to your device via COM ports automatically (except for the very first time where users will need to confirm the selection via OptionsSerial Port). Select the right Platform and SDK version and start your evaluation!
   a. **Hint** : Use HelpAbout to know your Platform and SDK version

For details on how to evaluate, any troubleshooting needs and/or to understand the know-how behind these steps, continue reading this SDK User Guide...

If the flashed demo on the EVM is an old version and you would like to upgrade to latest demo, continue reading this SDK User Guide...

2. System Overview

The mmWave SDK is split in two broad components: mmWave Suite and mmWave Demos.

2.1. mmWave Suite

mmWave Suite is the foundational software part of the mmWave SDK and would encapsulate these smaller components:

- Drivers (Part of the MCU PLUS SDK)
- DPL (Part of the MCU PLUS SDK)
- mmWaveLink and Firmware (Part of the Device Firmware Package - DFP)
- mmWave API
- Data processing layer (manager, processing units)
- Board Setup and Flash Utilities

2.2. mmWave Demos

SDK provides demos that depict the various control and data processing aspects of a mmWave application. Data visualization of the demo's output on a PC is provided as part of these demos. These demos are example code that are provided to customers to understand the inner workings of the mmWave devices and the SDK and to help them get started on developing their own application.

- mmWave Processing Demo with TI Gallery App - "mmWave Demo Visualizer"

2.3. External Dependencies

All tools/components needed for building mmWave sdk are included in the mmwave sdk installer. But the following external components (for debugging) are not included in the mmWave SDK.

- CCS (for debugging)

Please refer to the mmWave SDK Release Notes for detailed information on these external dependencies and the list of platforms that are supported.

2.4. Terms used in this document

Terms used	Comment
xWR	This is used throughout the document where that section/component/module applies to both AWR and IWR variants
BSS	This is used in the source code and sparingly in this document to signify the RADARSS. It is also interchangeably referred to as the mmWave Front End. Note that this term will only be used in the context of AWR294X SoC.
MSS	Master Sub-system. It is also interchangeably referred to as Cortex R5F.
DSS	DSP Sub-system. It is also interchangeably referred to as DSS or C66x core.

2.5. Related documentation/links

Other than the documents included in the mmwave_mcuplus_sdk package the following documents/links are important references.

- SoC links:
  - Automotive mmWave Sensors
  - Industrial mmWave Sensors
- Evaluation Modules (Gen1) (EVM) links:
  - Automotive Evaluation modules (Booster Pack, DEVPACK)
  - Industrial Evaluation modules (Booster Pack, ISK)

3. Getting started

The best way to get started with the mmWave SDK is to start running one of the various demos that are provided as part of the package. TI mmWave EVM comes pre-flashed with the mmWave demo. However, the version of the pre-flashed demo maybe older than the SDK version mentioned in this document. Users can follow this section and upgrade/run the flashed demo version. The demos (source and pre-built binaries) are placed at mmwave_mcuplus_sdk_<ver>/ti/demo/<platform>/mmw/ folder.

mmWave Demo

This demo is located at mmwave_mcuplus_sdk_<ver>/ti/demo/<platform>/mmw folder. The millimeter wave demo shows some of the radar sensing and object detection capabilities of the 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. A detailed explanation of this demo is available in the demo's docs folder and can be browsed via mmwave_mcuplus_sdk_<ver>/ docs/mmwave_sdk_module_documentation.html. This demo ships out detected objects and other real-time information that can be visualized using the TI Gallery App - 'mmWave Demo Visualizer' hosted at -https://dev.ti.com/gallery/view/mmwave/mmWave_Demo_Visualizer/ver/4.2.0/. The version of the mmWave Demo running on TI mmWave EVM can be obtained from the Visualizer app using the HelpAbout menu.

Device Support	AM273X+AWR2243	AWR294X
Demo Directory	TDMA, DDMA and TDMA ENET: ti\demo\am273x\mmw\	TDMA, DDMA and TDMA ENET: ti\demo\awr294x\mmw\
Binary prefix	am273x_mmw_demo	awr294x_mmw_demo
EVM	EVM + AWR2243BOOST	AWR294X EVM
Platform selection in Visualizer	AM273X-2243	AWR294X

Following sections describe the general procedure for booting up the device with the demos and then executing it.

3.1. Programming mmWave devices

Here is a little insight into the mmWave devices and the programmable cores they offer. For more detailed information, please refer to the Technical reference manual for the respective mmWave device. These details are needed when loading the binaries using CCS and/or to understand the various terminologies that exist in the "Getting started" section.

AM273X

This device has one cortex R5F core and one DSP C66x core available for user programming and are referred to as MSS/R5F and DSS/C66X respectively. The demos have 2 executables - one for MSS and one for DSS which should be loaded concurrently for the demos to work. See Running the Demos section for more details. The unit tests may have executables for either MSS or DSS or both. These executables are meant to be run in standalone operation. This means MSS unit test executable can be loaded and run on MSS R5F without downloading any code on DSS. Similarly, DSS unit test executable can be loaded and run on DSS C66x without downloading any code on DSS. The exceptions to this are the mmWave unit tests under full and datapath manager (DPM) unit tests.

AW294X

This device has one cortex R5F (MSS) core, one DSP C66x (DSS) core and one cortex R4 (BSS) core. The demos have 2 executables - one for MSS and one for DSS which should be loaded concurrently for the demos to work. The BSS binary comes as a part of the AWR294X Device Firmware Package (DFP). See Running the Demos section for more details. The unit tests may have executables for either MSS or DSS or both. These executables are meant to be run in standalone operation. This means MSS unit test executable can be loaded and run on MSS R5F without downloading any code on DSS in the CCS Development Mode. Similarly, DSS unit test executable can be loaded and run on DSS C66x without downloading any code on DSS. The exceptions to this are the mmWave unit tests under full and datapath manager (DPM) unit tests.

3.2. Loading images onto mmWave EVM

User can choose either one of Demonstration or CCS development modes for loading images onto the EVM.

⚠ **Flash part**

Please note that in the case of AM273X EVM, the flashing procedure works with the EVM with the flash part "GD25Q64CW2G". Ensure that you are using the same if you want to use the demonstration mode.

3.2.1. Demonstration Mode

This mode should be used when either upgrading the factory flashed binaries on the EVM to latest SDK version using the pre-built binaries provided in the SDK release or for field deployment of mmWave sensors.

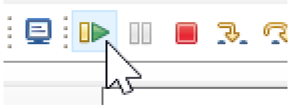
1. Follow the procedure mentioned in the section (How to flash an image onto mmWave EVM). Use the mmwave_mcuplus_sdk_<ver>/ti/demo/<platform>/mmw/<demo_binary>.appimage as the metaimage.
2. Reboot the device to run the demo image every time on power up. No other image loading step is required on subsequent boot to run the demo.
3. Follow the steps mentioned in Running the Demos section.

3.2.2. CCS development mode

This mode should be used when debugging with CCS is involved and/or developing an mmWave application where the .appimage files keep changing constantly and frequent flashing of image onto the board is not desirable. This mode allows you to flash once and then use CCS to download a different image to the device's RAM on every boot.

This mode is the recommended way to run the unit tests for the drivers and components which can be found in the respective test directory for that component. See section mmWave SDK - TI components for location of each component's test code

1. EVM and CCS setup: Follow the steps in How to connect mmWave EVM to CCS using JTAG to setup the environment for CCS connectivity.
2. Load the corresponding .xe66 and/or .xer5f file(s) (for example, for running the demos, load <mmwave_sdk_device>_mmw_demo_mss.xer5f and <mmwave_sdk_device>_mmw_demo_dss.xe66 prebuilt executables provided in the SDK release package at mmwave_mcuplus_sdk_<ver>/ti/demo/<platform>/<demo>/mmw).
3. Click on "Run" button to run the file.



4. To reload, disconnect the connected cores, power cycle the EVM and connect to the cores on CCS again.

3.3. Running the Demos

Follow this subsection to experience the mmWave functionality using the out-of-box mmWave demo. Before you proceed further, make sure that:

1. You have loaded the right demo binary using the section above, set the EVM to functional mode (QSPI Boot Mode) and powered up the device. If you have followed the steps correctly,
    a. In case of Demonstration Mode, mmwave_mcuplus_sdk_<ver>/ti/demo/<platform>/mmw/<demo_binary>.appimage should have been flashed onto the EVM.
    b. In case of CCS development Mode, mmwave_mcuplus_sdk_<ver>/ti/utils/ccsdebug/<platform>_ccsdebug.appimage should have been flashed onto the EVM and <mmwave_sdk_device>_mmw_demo_mss.xer5f and <mmwave_sdk_device>_mmw_demo_dss.xe66 prebuilt executables should be running on the MSS and DSS cores respectively.
2. You have connected the EVM to the PC using its XDS110 micro-USB port/cable.

3.3.1. mmWave Demo

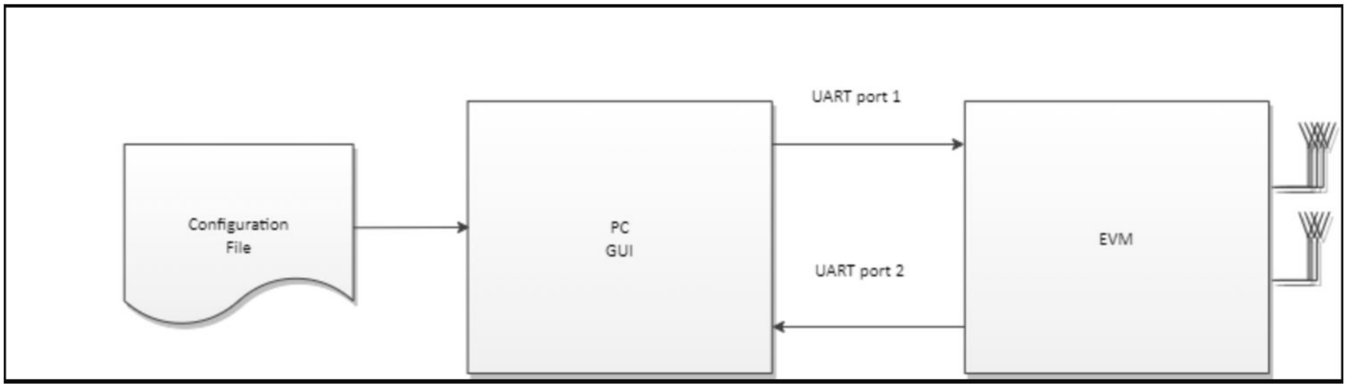


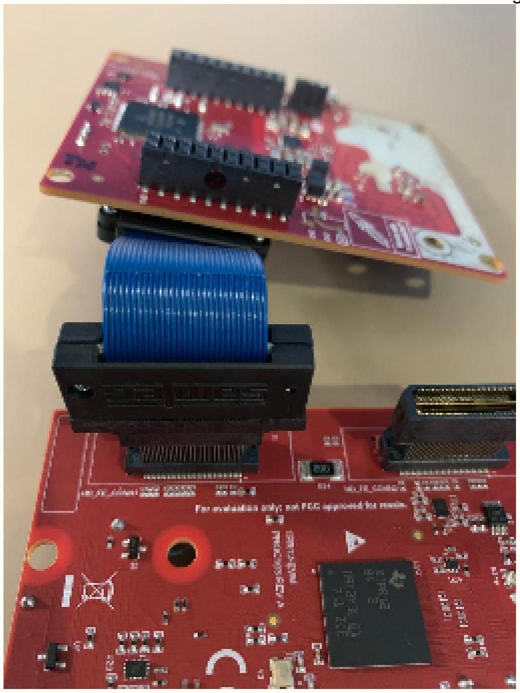
Figure 1: mmWave Demo Visualizer- mmWave Device Connectivity

Note: The "EVM" in the above figure refers to the AM273X+AWR2243BOOST or the AWR294X System.

1. Power on the EVM in functional/QSPI-boot mode with right binary loaded (see section above) and connect it to the PC as shown above with the USB cable.

> **AM273X EVM + AWR2243BOOST Configuration**
>
> In the case of AM273X EVM + AWR2243BOOST Configuration, ensure the following:
>
> - The AWR2243BOOST is setup in "Functional Mode" i.e. only SOP0 is jumper/switch is 1 .

- AWR2243BOOST is connected to the AM273X EVM using the HD_FE_CONN1 port, as shown below:



2. Browse to the TI gallery app "mmWave Demo Visualizer" at https://dev.ti.com/gallery/view/mmwave/mmWave_Demo_Visualizer/ver/4.2.0/. Use HelpREADME.md from inside this app for more information on how to run/configure this app.

⚠ **Baud rate**

In the visualizer, the baud rate to be selected for the Auxillary data port is 892857 for AM273X and 852272 for AWR294X.

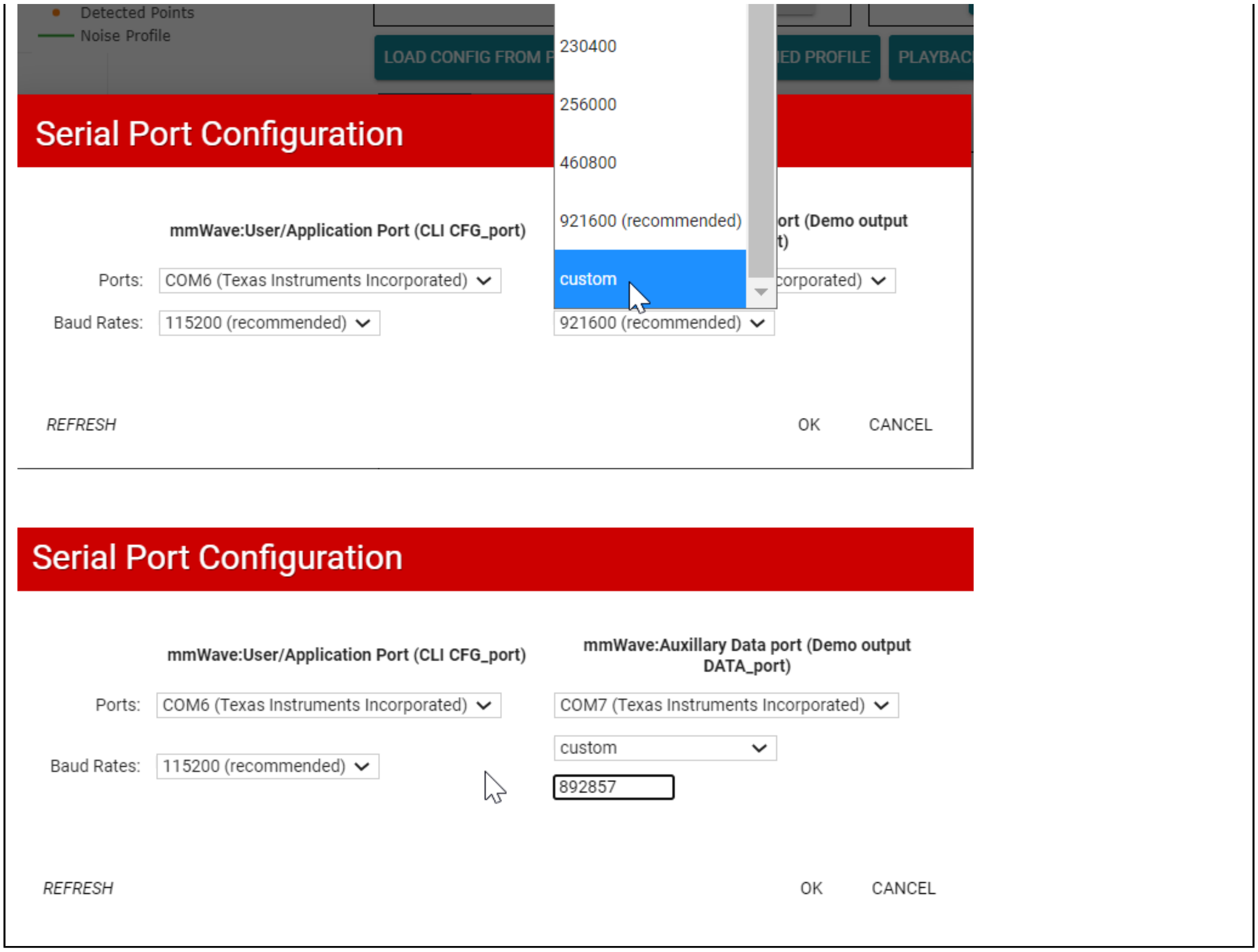
First Time Setup

a. If this is the first time you are using this App, you may be requested to install a plug-in and the TI Cloud Agent Application. This step will also install the right **XDS110 drivers** needed for UART port detection.
b. Once the demo is running on the mmWave sensors and the USB is connected from the board to the PC, the app will try to automatically detect the COM ports for your device. If auto-detection doesn't work, then you will need to configure the serial ports in this App. Run the device manager on the PC and locate the following COM ports as shown in the section "How to identify the COM ports for mmWave EVM" below. In the Visualizer App, go to the Menu->Options->Serial Port and perform the settings as shown below and click on OK.
   i.
      - **CFG_port**: Use COM port number for "**XDS110 Class Application/User UART**": Baud: 115200. This is the port where **CLI (command line interface)** runs for all the demos.
      - **Data_port**: Use COM port "**XDS110 Class Auxiliary Data port**": Baud: 892857/852272. This is the port on which binary data generated by the processing chain in the mmWave demo will be received by the PC. This is the detected object list and its properties (range, doppler, angle, etc). Note that the default Baud for this port is 921600, and you will separately have to set it to 892857 for AM273X and 852272 for AWR294X using Custom Baud Rate selection.

      ⚠ **COM Port**

      Please note that the COM port numbers on your setup maybe different from the one shown in this panel. Please use the correct COM port number from your setup for following steps.

      - **Custom Baud Rate selection**: For selecting the custom baud rate for the Data port, perform the following steps-
         - Click on the Baud Rates drop down menu for the Data port, where "921600 (recommended)" is shown by default.
         - Scroll down the menu and click on "custom".
         - A text box will appear. Enter "892857" in the text box for AM273X and 852272 for AWR294x.



Serial Port Configuration

mmWave:User/Application Port (CLI CFG_port)

Ports: COM6 (Texas Instruments Incorporated)

Baud Rates: 115200 (recommended)

...ort (Demo output ...t)

...corporated)

921600 (recommended)

Serial Port Configuration

mmWave:User/Application Port (CLI CFG_port)

Ports: COM6 (Texas Instruments Incorporated)

Baud Rates: 115200 (recommended)

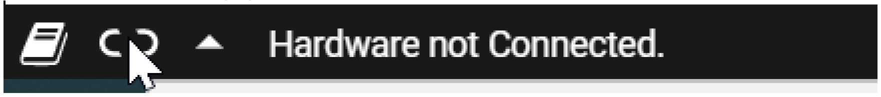
mmWave:Auxillary Data port (Demo output DATA_port)

COM7 (Texas Instruments Incorporated)

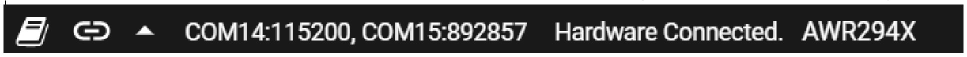
custom

892857

a. At this point, this app will automatically try to connect to the target (mmWave Sensor). DATA_port will be marked connected only after device is configured and sending out detected point cloud. If CFG_port does not connect or if the connection fails, you should try to connect to the target by clicking in the bottom left corner of this App. If that fails as well, redo the serial port configuration as shown in "First time Setup" panel above.


Hardware not Connected.

b. After the App is connected to the target, you can select the configuration parameters (Frequency Band, Platform, etc) in the "Setup details" and "Scene Selection" area of the **CONFIGURE** tab. Note that upon successful connection, you can see the following status:


COM14:115200, COM15:892857 Hardware Connected. AWR294X

Troubleshooting tip

In case you're unable to perform this step correctly, reconfirm the following:

- The demo image has been loaded correctly. In case of CCS development mode, you should see "Debug: CLI is operational" printed on the CCS Console.
- The Serial Ports and corresponding baud rates been set correctly.
- The two COM Ports are not being used by any other application.

Try reconnecting or refreshing the page.

c. Besides selecting the configuration parameters, you should select which plots you want to see. This can be done using the "check boxes" in the "Plot Selection" area. Adjust the frame rate depending on number of plots you want to see. For selecting heatmap plots, set frame rate to less than or equal to 4 fps. When selecting frame rate to be 25-30fps, for better GUI performance, select only the scatter plot and statistics plot.

d. Once the configuration is selected, you can send the configuration to the device (use "SEND CONFIG TO MMWAVE DEVICE" button).

e. After the configuration is sent to the device, you can switch to the **PLOTS** view/tab and the plots that you selected will be shown.

f. You can switch back from "Plots" tab to "Configure" tab, reconfigure your "Scene Selection", "Object Detection" and/or "Plot Selection" values and re-send the configuration to the device to try a different profile. After a new configuration has been selected, just press the "SEND CONFIG TO MMWAVE DEVICE" button again and the device will be reconfigured. This can be done without rebooting the device. If you change the parameters in the "Setup Details", then you will need to take further action before trying the new configurations

    i. If SDK version is changed: make sure the mmW demo running on the connected TI EVM matches the selected SDK version in the GUI
    ii. **If Antenna Config is changed: make sure the TI EVM is rebooted before sending the new configuration.**

g. Alternatively, you can load one of the example configurations (.cfg) present in the mmwave_mcuplus_sdk_<ver>\ti\demo\<platform>\mmw\profiles folder and load it through the "LOAD CONFIG FROM PC AND SEND" button on the "Plots" tab.

Troubleshooting tip

In case you're loading the configuration but unable to to see any plots, check the following in that order:

- Check if the hardware is connected to the visualizer
- Try sending the configuration/file again
- Look at the console log for errors
- Disconnect and reconnect to the visualizer
- Try with one of the example profile configurations
- Load the demo in CCS Development mode and check the console for errors

3. If board is rebooted, follow the steps starting from 1 above.

COM port after reboot

Whenever TI EVM is power-cycled (rebooted), you will need to use the bottom left serial port connection icon inside TI gallery app "mmWave Demo Visualizer" for disconnecting and reconnecting the COM ports. Note that if you used the CLI COM port directly to send the commands (instead of TI gallery app) you will have to close the CLI teraterm window and open a new one on every reboot.

Inner workings of the GUI

In the background, GUI performs the following steps:

- Creates or reads the configuration file and sends to the mmWave device using the COM port called **CFG_port**. It saves the information locally to be able to make sense of the incoming data that it will display. Refer to the CFG Section for details on the configuration file contents.
- Receives the data generated by the demo on the visualization/Data COM port and processes it to create various displays based on the GUI configuration in the cfg file.
    - The format of the data streamed out of the demo is documented in mmw demo's doxygen mmwave_mcuplus_sdk_<ver>\ti\demo\<platform>\mmw\docs\doxygen\html\index.html under section: "Output information sent to host".
- On every reconfiguration, it sends a 'sensorStop' command to the device first to stop the active run of the mmWave device. Next, it sends the command 'flushCfg' to flush the old configuration before sending the new configuration. It is mandatory to flush the old configuration before sending a new configuration. Additionally, it is mandatory to send all the commands for that demo/platform even if the user desires the functionality to be disabled i.e. no commands are optional.

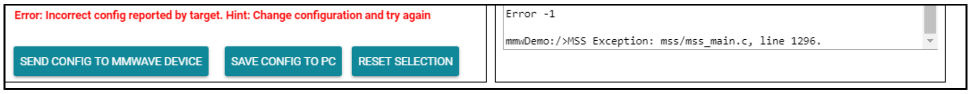
Advanced GUI options

- User can configure the device from their own configuration file or the saved app-generated configuration file by using the "LOAD CONFIG FROM PC AND SEND" button on the **PLOTS** tab. Make sure the first two commands in this config file are "sensorStop" followed by "flushCfg".
- User can temporarily pause the mmWave sensor by using the "STOP" button on the plots tab. The sensor can be restarted by using the "START" button. In this case, sensor starts again with the already loaded configuration and no new configuration is sent from the App.
- User can simultaneously plot and record the processed/detected objects data coming out of the DATA_port using the "RECORD START" button in the plots tab. Set the max limits for file size or record time as per your requirements to prevent infinite capturing of data. The saving of data can be manually stopped using the "Record Stop" button (if the max limits are not reached).
- User can use the "PLAYBACK START" button to playback the data and config file recorded via "RECORD START" button in the plots tab. User should make sure the data file and the config file used in this playback are the matching set. This feature can only be used when sensor device is either not connected or stopped.
- Once the demo has started and plots are active, user can tune the demo using the "Real Time tuning tab" or "Advanced commands" tab and then save the tuned profile using "EXPORT TUNED PROFILE" button on the **PLOTS** tab.

Console Messages window in Visualizer

Console message window echoes the following debug information for the users

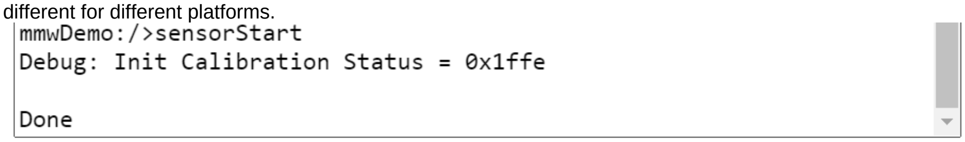
- Every command that is sent to the TI mmWave EVM and the response back from the EVM
- Any runtime assert conditions detected by the demo running on TI mmWave EVM after the sensor is started. This is helpful when mmW demo is flashed onto the EVM and CCS connectivity is not available. It spits out file name and line number to allow users to browse to the source code and understand the error.



Error: Incorrect config reported by target. Hint: Change configuration and try again

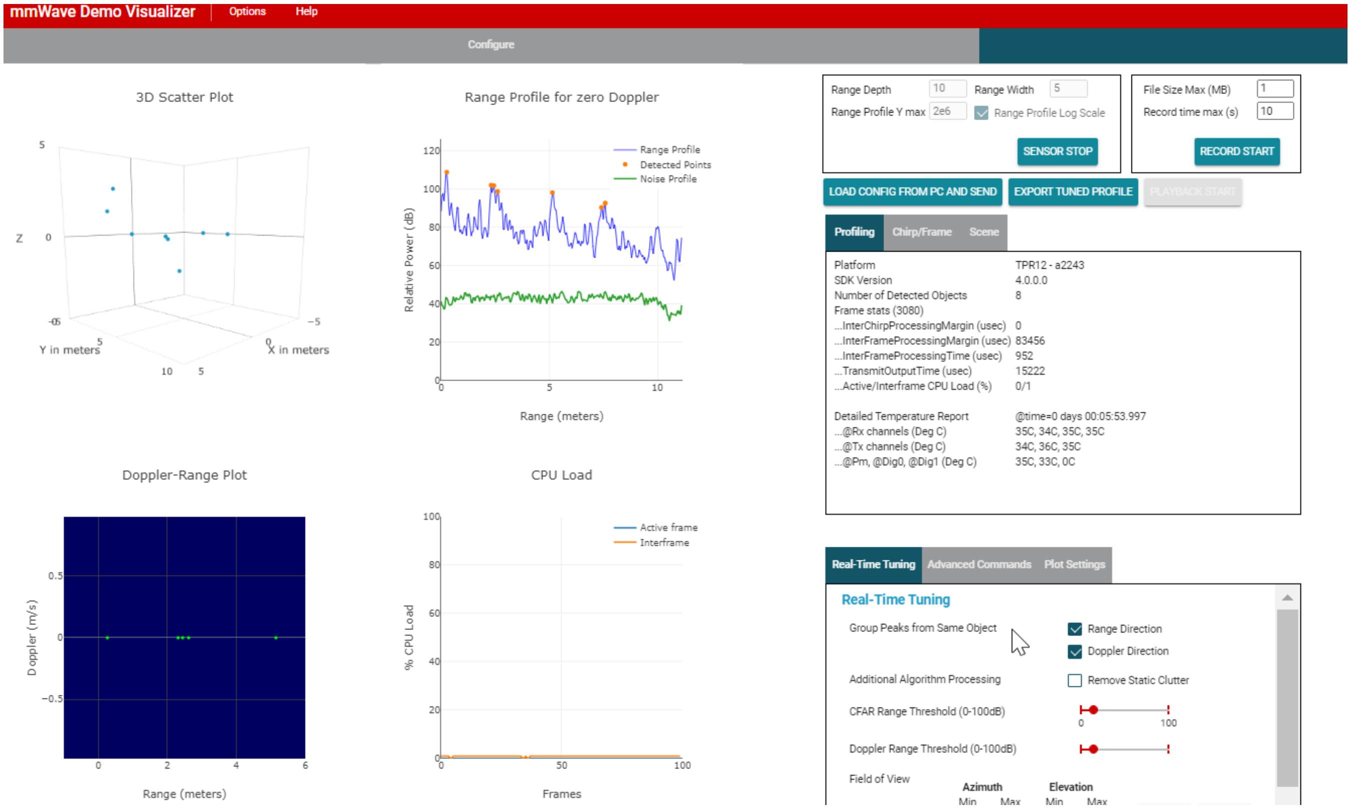
```
Error -1

mmwDemo:/>MSS Exception: mss/mss_main.c, line 1296.
```

- At times, a negative error code is spit out in the error message (either in Visualizer console or in the CCS console window). To understand or decode that error, please refer to the mmWave demo doxygen ((browse via mmwave_mcuplus_sdk_<ver>\docs\mmwave_sdk_module_documentation.html).
- Init time calibration status after the first sensorStart is issued post reboot for debugging boot time or start failures. This status might be different for different platforms.



```
mmwDemo:/>sensorStart
Debug: Init Calibration Status = 0x1ffe

Done
```

Here is an example of plots that mmWave Demo Visualizer produces based on the config that is passed to the demo application running on mmWave sensor.



Range Depth 10 Range Width 5
Range Profile Y max 2e6 ☑ Range Profile Log Scale

File Size Max (MB) 1
Record time max (s) 10

```
Platform                                  TPR12 - a2243
SDK Version                               4.0.0.0
Number of Detected Objects                8
Frame stats (3080)
...InterChirpProcessingMargin (usec)      0
...InterFrameProcessingMargin (usec)      83456
...InterFrameProcessingTime (usec)        952
...TransmitOutputTime (usec)              15222
...Active/Interframe CPU Load (%)         0/1

Detailed Temperature Report               @time=0 days 00:05:53.997
...@Rx channels (Deg C)                   35C, 34C, 35C, 35C
...@Tx channels (Deg C)                   34C, 36C, 35C
...@Pm, @Dig0, @Dig1 (Deg C)              35C, 33C, 0C
```

Real-Time Tuning

Group Peaks from Same Object ☑ Range Direction ☑ Doppler Direction

Additional Algorithm Processing ☐ Remove Static Clutter

CFAR Range Threshold (0-100dB)

Doppler Range Threshold (0-100dB)

Field of View Azimuth Min Max Elevation Min Max

> ⚠ **COM port after reboot**
>
> Refer to the Visualizer Readme and the User Guide for a deep dive into the demo visualizer (present in the "Help" menu).

3.4. mmWave SDK OOB Demo with LVDS Based Instrumentation

In this use case, high bandwidth data (raw ADC capture) is shipped from the device to a PC over the LVDS interface (captured by the DCA1000EVM) and saved onto the filesystem. For instructions on how to run the mmWave demo with this enabled, refer to the section How to run mmWave demo with LVDS-based instrumentation. For implementation details, refer to the mmwave demo documentation (mmwave_mcuplus_sdk_<ver>/docs/mmwave_sdk_module_documentation.html).

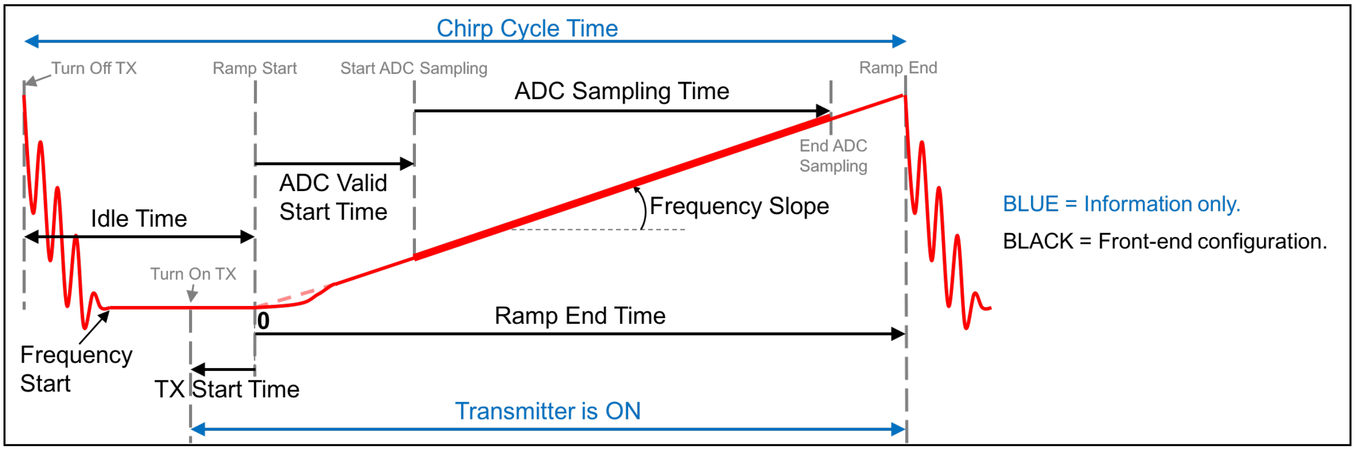
3.5. mmWave SDK OOB Demo with Ethernet Streaming Enabled

mmWave SDK out-of-box demo offers the capability of streaming the coordinates and velocity of the detected objects over Ethernet using TCP protocol and LwIP stack. It is based on the TCPECHO example which comes as a part of the default LwIP contrib apps. For instructions on how to run the mmWave demo with Ethernet streaming enabled, refer to the section How to Run MMWAVE SDK OOB Demo with Ethernet Streaming Enabled. For implementation details, refer to the mmwave demo documentation (mmwave_mcuplus_sdk_<ver>/docs/mmwave_sdk_module_documentation.html).

3.6. Configuration (.cfg) File Format (For DDMA commands, refer to the DDMA Documentation)

Each line in the .cfg file describes a command with parameters. The various commands and their arguments are described in the table below (arguments are in sequence). For mmW demo, users can create their own config files from the Visualizer GUI by using the "Save Config to PC" button or starting from the few sample profiles provided in the mmwave_mcuplus_sdk_<ver>\ti\demo\<platform>\mmw\profiles directory.

Most of the parameters described below are the same as the mmwavelink API specifications (see doxygen mmwave_mcuplus_sdk_<ver>\ti\control\mmwavelink\docs\doxygen\html\index.html.) Additionally, users can refer to the chirp diagram below to understand the chirp and profile related parameters or the appnote http://www.ti.com/litv/pdf/swra553



3.6.1.1.1. Figure 2: Chirp Diagram

Configuration command	Command details	Command Parameters	Usage in mmW demo
dfeDataOutputMode	The values in this command should not change between sensorStop and sensorStart. Reboot the board to try config with different set of values in this command This is a mandatory command.		
		<modeType> 1 - frame based chirps 2 - continuous chirping 3 - advanced frame config	TDM: only option 1 and 3 are supported DDM: only option 1 is supported
channelCfg	Channel config message to RadarSS. See mmwavelink doxgen for details. The values in this command should not change between sensorStop and sensorStart. Reboot the board to try config with different set of values in this command This is a mandatory command.		
		<rxChannelEn> Receive antenna mask e.g for 4 antennas, it is 0x1111b = 15	4 antennas supported
		<txChannelEn> Transmit antenna mask	Refer to the antenna layout on the EVM/board to determine the right Tx antenna mask needed to enable the desired virtual antenna configuration. For example, in AWR2243BOOST the 2 azimuth antennas can be enabled using bitmask 0x5 (i.e. tx1 and tx3). The azimuth and elevation antennas can both be enabled using bitmask 0x7 (i.e. tx1, tx2 and tx3). AWR294X: Supports 4 transmit antennas using bitmask 0xF (i.e. tx1, tx2, tx3 and tx4).
		<cascading> SoC cascading, not applicable, set to 0	n/a
adcCfg	ADC config message to RadarSS. See mmwavelink doxgen for details.		
		<numADCBits> Number of ADC bits (0 for 12-bits, 1 for	only 16-bit is supported

	The values in this command should not change between sensorStop and sensorStart. Reboot the board to try config with different set of values in this command This is a mandatory command.	14-bits and 2 for 16-bits)	
		<adcOutputFmt> Output format : 0 - real 1 - complex 1x (image band filtered output) 2 - complex 2x (image band visible))	AM273X: only complex mode is supported AWR294X: only real mode is supported
adcbufCfg	adcBuf hardware config. The values in this command can be changed between sensorStop and sensorStart. This is a mandatory command.		
		<subFrameIdx> subframe Index	For legacy mode, that field should be set to -1. For advanced frame mode, it should be set to either the intended subframe number or -1 to apply same config to all subframes.
		<adcOutputFmt> ADCBUF out format 0-Complex, 1-Real	AM273X: only complex mode is supported AWR294X: only real mode is supported
		<SampleSwap> ADCBUF IQ swap selection: 0-I in LSB, Q in MSB, 1-Q in LSB, I in MSB	AM273X: only option 1 is supported
		<ChanInterleave> ADCBUF channel interleave configuration: 0 - interleaved, 1 - non-interleaved	TDM: only option 1 is supported DDM: only option 0 is supported
		<ChirpThreshold> Chirp Threshold configuration used for ADCBUF buffer to trigger ping/pong buffer switch. Valid values: 0-8 for demos that use DSP for 1D FFT and LVDS streaming is disabled only 1 for demos that use HWA for 1D FFT	only value of 1 is supported since demos use HWA for 1D FFT
profileCfg	Profile config message to RadarSS and datapath. See mmwavelink doxgen for details. The values in this command can be changed between sensorStop and sensorStart. This is a mandatory command. **txCalibEnCfg Field** This CLI command doesn't expose the txCalibEnCfg field in the mmwavelink structure. User should follow the mmwavelink documentation and update the CLI profileCfg handler function accordingly. The current handler sets the value to 0 for this field (backward compatible mode) ⓘ Combination of numAdcSamples in profileCfg (and numRangeBins), numDopplerChirps = total number of chirps/(num TX in MIMO mode) in frameCfg or subFrameCfg, number of TX and RX antennas in channelCfg and chirpCfg determine the size of Radarcube and other internal buffers/heap in the demo. It is possible that some combinations of these values result in out of memory conditions for these heaps and demo will reject such configuration. Refer to demo and		
		<profileId> profile Identifier	Legacy frame (dfeOutputMode=1): could be any allowed value but only one valid profile per config is supported Advanced frame (dfeOutputMode=3): could be any allowed value but only one profile per subframe is supported. However, different subframes can have different profiles
		<startFreq> "Frequency Start" in GHz (float values allowed) Examples: 77 61.38	any value as per mmwavelink doxgen/device datasheet but represented in GHz. Refer to the chirp diagram shown above to understand the relation between various profile parameters and inter-dependent constraints.
		<idleTime> "Idle Time" in u-sec (float values allowed) Examples: 7 7.15	any value as per mmwavelink doxgen/device datasheet but represented in usec. Refer to the chirp diagram shown above to understand the relation

	DPC doxygen to understand the data buffer layout and use the system printfs on sensorStart in CCS console window to understand the exact heap usage for a given configuration.		between various profile parameters and inter-dependent constraints.
		<adcStartTime> "ADC Valid Start Time" in u-sec (float values allowed) Examples: 7 7.34	any value as per mmwavelink doxgen/device datasheet but represented in usec. Refer to the chirp diagram shown above to understand the relation between various profile parameters and inter-dependent constraints.
		<rampEndTime> "Ramp End Time" in u-sec (float values allowed) Examples: 58 216.15	any value as per mmwavelink doxgen/device datasheet but represented in usec Refer to the chirp diagram shown above to understand the relation between various profile parameters and inter-dependent constraints.
		<txOutPower> Tx output power back-off code for tx antennas	only value of '0' has been tested within context of mmW demo
		<txPhaseShifter> tx phase shifter for tx antennas	only value of '0' has been tested within context of mmW demo
		<freqSlopeConst> "Frequency slope" for the chirp in MHz/usec (float values allowed) Examples: 68 16.83	any value greater than 0 as per mmwavelink doxgen/device datasheet but represented in MHz/usec. Refer to the chirp diagram shown above to understand the relation between various profile parameters and inter-dependent constraints.
		<txStartTime> "TX Start Time" in u-sec (float values allowed) Examples: 1 2.92	any value as per mmwavelink doxgen/device datasheet but represented in usec. Refer to the chirp diagram shown above to understand the relation between various profile parameters and inter-dependent constraints.
		<numAdcSamples> number of ADC samples collected during "ADC Sampling Time" as shown in the chirp diagram above Examples: 256 224	any value as per mmwavelink doxgen/device datasheet. Refer to the chirp diagram shown above to understand the relation between various profile parameters and inter-dependent constraints.
		<digOutSampleRate> ADC sampling frequency in ksps. (<numAdcSamples> / <digOutSampleRate> = "ADC Sampling Time") Examples: 5500	any value as per mmwavelink doxgen/device datasheet. Refer to the chirp diagram shown above to understand the relation between various profile parameters and inter-dependent constraints.
		<hpfCornerFreq1> HPF1 (High Pass Filter 1) corner frequency 0: 175 KHz 1: 235 KHz 2: 350 KHz 3: 700 KHz	any value as per mmwavelink doxgen/device datasheet
		<hpfCornerFreq2> HPF2 (High Pass Filter 2) corner frequency 0: 350 KHz 1: 700 KHz	any value as per mmwavelink doxgen/device datasheet

		2: 1.4 MHz 3: 2.8 MHz	**Note:** for AWR294X, <hpfCornerFreq2> and <hpfCornerFreq1> should have the same value. Different values cannot be used.
		<rxGain> OR'ed value of RX gain in dB and RF gain target (See mmwavelink doxgen for details)	any value as per mmwavelink doxgen/device datasheet
chirpCfg	Chirp config message to RadarSS and datapath. See mmwavelink doxgen for details. The values in this command can be changed between sensorStop and sensorStart. This is a mandatory command.		
		chirp start index	any value as per mmwavelink doxygen
		chirp end index	any value as per mmwavelink doxygen
		profile identifier	should match the profileCfg->profileId
		start frequency variation in Hz (float values allowed)	only value of '0' has been tested within context of mmW demo
		frequency slope variation in kHz/us (float values allowed)	only value of '0' has been tested within context of mmW demo
		idle time variation in u-sec (float values allowed)	only value of '0' has been tested within context of mmW demo
		ADC start time variation in u-sec (float values allowed)	only value of '0' has been tested within context of mmW demo
		tx antenna enable mask (Tx2,Tx1) e.g (10)b = Tx2 enabled, Tx1 disabled.	See note under "Channel Cfg" command above. TDM: Individual chirps should have either only one distinct Tx antenna enabled (MIMO) or same TX antennas should be enabled for all chirps DDM: All transmitters should be active for the same chirp
lowPower	Low Power mode config message to RadarSS. See mmwavelink doxgen for details. The values in this command should not change between sensorStop and sensorStart. Reboot the board to try config with different set of values in this command. This is a mandatory command.		
		<don’t_care>	set to 0
		ADC Mode 0x00 : Regular ADC mode 0x01 : Low power ADC mode	use value of '0' or '1' (depending on profileCfg->digOutSampleRate)
frameCfg	frame config message to RadarSS and datapath. See mmwavelink doxgen for details. dfeOutputMode should be set to 1 to use this command The values in this command can be changed between sensorStop and sensorStart. This is a mandatory command when dfeOutputMode is set to 1.		
		chirp start index (0-511)	any value as per mmwavelink doxgen but corresponding chirpCfg should be defined
		chirp end index (chirp start index-511)	any value as per mmwavelink doxgen but corresponding chirpCfg should be defined
		number of loops (1 to 255)	any value as per mmwavelink doxgen/device datasheet but greater than or equal to 4. Note: If value of 2 is desired for number of Doppler Chirps, one must update the demo/object detection DPC source code to use rectangular window for Doppler DPU instead of Hanning window.
		number of frames (valid range is 0 to 65535, 0 means infinite)	any value as per mmwavelink doxgen
		numAdcSamples	any value as per mmwavelink doxgen/device datasheet.

		number of ADC samples collected during "ADC Sampling Time" as shown in the chirp diagram above Examples: 256 224	Refer to the chirp diagram shown above to understand the relation between various profile parameters and inter-dependent constraints.
		frame periodicity in ms (float values allowed)	any value as per mmwavelink doxgen and represented in msec. However frame should not have more than 50% duty cycle (i.e. active chirp time should be <= 50% of frame period). Also it should allow enough time for selected UART output to be shipped out (selections based on guiMonitor command) else demo will assert if the next frame start trigger is receive from the front end and current frame is still ongoing. User can use the output of stats TLV to tune this parameter.
		trigger select 1: Software trigger 2: Hardware trigger.	only option for Software trigger is supported
		Frame trigger delay in ms (float values allowed)	any value as per mmwavelink doxgen and represented in msec.
advFrameCfg	Advanced config message to RadarSS and datapath. See mmwavelink doxgen for details. The dfeOutputMode should be set to 3 to use this command. The values in this command can be changed between sensorStop and sensorStart. This is a mandatory command when dfeOutputMode is set to 3.		
		<numOfSubFrames> Number of sub frames enabled in this frame	any value as per mmwavelink doxgen
		<forceProfile> Force profile	only value of 0 is supported
		<numFrames> Number of frames to transmit (1 frame = all enabled sub frames)	any value as per mmwavelink doxgen
		<triggerSelect> trigger select 1: Software trigger 2: Hardware trigger.	only option for Software trigger is supported
		<frameTrigDelay> Frame trigger delay in ms (float values allowed)	any value as per mmwavelink doxgen and represented in msec.
		<numOfSubFrames> Number of sub frames for sequence configuration	should be kept the same as the first argument
subFrameCfg	Subframe config message to RadarSS and datapath. See mmwavelink doxgen for details. The dfeOutputMode should be set to 3 to use this command. The values in this command can be changed between sensorStop and sensorStart. This is a mandatory command when dfeOutputMode is set to 3.		
		<subFrameNum> subframe Number for which this command is being given	value of 0 to RL_MAX_SUBFRAMES-1
		<forceProfileIdx> Force profile index	ignored as <forceProfile> in advFrameCfg should be set to 0
		<chirpStartIdx> Start Index of Chirp	any value as per mmwavelink doxgen but corresponding chirpCfg should be defined
		<numOfChirps> Num of unique Chirps per burst including start index	any value as per mmwavelink doxgen but corresponding number of chirpCfg should be defined
		<numLoops> No. of times to loop through the unique chirps	any value as per mmwavelink doxgen but greater than or equal to 4 Note: If value of 2 is desired for number of Doppler Chirps, one must update the demo/object

			detection DPC source code to use rectangular window for Doppler DPU instead of Hanning window.
		<burstPeriodicity> Burst periodicty in msec (float values allowed) and meets the criteria burstPeriodicity >= ((numLoops)* (Sum total of time duration of all unique chirps in that burst)) + InterBurstBlankTime	any value as per mmwavelink doxgen and represented in msec but subframe should not have more than 50% duty cycle and allow enough time for selected UART output to be shipped out (selections based on guiMonitor command)
		<chirpStartIdxOffset> Chirp Start address increament for next burst	set it to 0 since demo supports only one burst per subframe
		<numOfBurst> Num of bursts in the subframe	set it to 1 since demo supports only one burst per subframe
		<numOfBurstLoops> Number of times to loop over the set of above defined bursts, in the sub frame	set it to 1 since demo supports only one burst per subframe
		<subFramePeriodicity> subFrame periodicty in msec (float values allowed) and meets the criteria subFramePeriodicity >= Sum total time of all bursts + InterSubFrameBlankTime	set to same as <burstPeriodicity> since demo supports only one burst per subframe
subDateFrame Cfg	Subframe data configuration parameters. See mmwavelink doxygen for more details. The dfeOutputMode should be set to 3 to use this command. The values in this command can be changed between sensorStop and sensorStart. This is a mandatory command when dfeOutputMode is set to 3.		
		<subFrameNum> subframe Number for which this command is being given	value of 0 to RL_MAX_SUBFRAMES-1
		<numAdcSamples> Number of half words of ADC samples per data packet	should be same as what was set in profileCfg
		<totalChirps> Number of Chirps in Sub-Frame	should be set to numOfChirps * numLoops * numOfBurst * burstLoop
		<numChirpsInDataPacket> Number of Chirps Per Data Packet to process at a time	should be set to 1
guiMonitor	Plot config message to datapath. The values in this command can be changed between sensorStop and sensorStart. This is a mandatory command.		
		All parameters below are flags (1 to enable and 0 to disable)	
		<subFrameIdx> subframe Index	For legacy mode, that field should be set to -1 whereas for advanced frame mode, it should be set to either the intended subframe number or -1 to apply same config to all subframes.
		<detected objects> 1 - enable export of point cloud (x,y,z,doppler) and point cloud sideinfo (SNR, noiseval) 2 - enable export of point cloud (x,y,z,doppler) 0 - disable	TDM: All values supported DDM: Only 2 and 0 supported
		<log magnitude range> 1 - enable export of log magnitude range profile at zero Doppler 0 - disable	all values supported
		<noise profile> 1 - enable export of log magnitude noise profile 0 - disable	TDM: all values supported DDM: must be 0
			TDM: all values supported DDM: must be 0

		<rangeAzimuthHeatMap> or <rangeAzimuthElevationHeatMap> range-azimuth or range-azimuth-elevation heat map related information <rangeAzimuthHeatMap> This output is provided only in demos that use AoA (legacy) DPU for AoA processing 1 - enable export of zero Doppler radar cube matrix, all range bins, all azimuth virtual antennas to calculate and display azimuth heat map. (The GUI computes the FFT of this to show heat map) 0 - disable < rangeAzimuthElevationHeatMap > This output is provided in demos that use AoA 2D DPU for AoA processing 1 - enable export of zero Doppler radar cube matrix, all range bins, all virtual antennas to calculate and display azimuth heat map. (The GUI remaps the antenna symbols and computes the FFT of this stream to show azimuth heat map only). 0 - disable	
		<rangeDopplerHeatMap> range-doppler heat map 1 - enable export of the whole detection matrix. Note that the frame period should be adjusted according to UART transfer time. 0 - disable	all values supported
		<statsInfo> statistics (CPU load, margins, device temperature readings, etc) 1 - enable export of stats data. 0 - disable	all values supported
aoaFovCfg	Command for datapath to filter out detected points outside the specified range in azimuth or elevation plane The values in this command can be changed between sensorStop and sensorStart and even when the sensor is running. This is a mandatory command.		Specific to TDM.
		<subFrameIdx> subframe Index	For legacy mode, that field should be set to -1 whereas for advanced frame mode, it should be set to either the intended subframe number or -1 to apply same config to all subframes.
		<minAzimuthDeg>	minimum azimuth angle (in degrees) that specifies the start of field of view
		<maxAzimuthDeg>	maximum azimuth angle (in degrees) that specifies the end of field of view
		<minElevationDeg>	minimum elevation angle (in degrees) that specifies the start of field of view
		<maxElevationDeg>	maximum elevation angle (in degrees) that specifies the end of field of view
cfarCfg (TDM)	CFAR config message to datapath. The values in this command can be changed between sensorStop and sensorStart and even when the sensor is running. This is a mandatory command.		Specific to TDM. For DDM, check cfarCfg (DDM)
		<subFrameIdx> subframe Index	For legacy mode, that field should be set to -1 whereas for advanced frame mode, it should be set to either the intended

			subframe number or -1 to apply same config to all subframes.
		<procDirection> Processing direction: 0 – CFAR detection in range direction 1 – CFAR detection in Doppler direction	all values supported; 2 separate commands need to be sent; one for Range and other for doppler.
		<mode> CFAR averaging mode: 0 - CFAR_CA (Cell Averaging) 1 - CFAR_CAGO (Cell Averaging Greatest Of) 2 - CFAR_CASO (Cell Averaging Smallest Of)	all values supported
		<noiseWin> noise averaging window length: Length of the one sided noise averaged cells in samples Make sure 2*(noiseWIn+guardLen) <numRangeBins for range direction and 2*(noiseWIn+guardLen) <numDopplerBins for doppler direction.	supported
		<guardLen> one sided guard length in samples Make sure 2*(noiseWIn+guardLen) <numRangeBins for range direction and 2*(noiseWIn+guardLen) <numDopplerBins for doppler direction.	supported
		<divShift> Cumulative noise sum divisor expressed as a shift. Sum of noise samples is divided by 2^<divShift>. Based on <mode> and <noiseWin> , this value should be set as shown in next columns. The value to be used here should match the "CFAR averaging mode" and the "noise averaging window length" that is selected above. The actual value that is used for division (2^x) is a power of 2, even though the "noise averaging window length" samples may not have that restriction.	CFAR_CA: <divShift> = ceil(log$_2$(2 x <noiseWin>)) CFAR_CAGO/_CASO: <divShift> = ceil(log$_2$(<noiseWin>)) In profile_2d.cfg, value of 3 means that the noise sum is divided by 2^3=8 to get the average of noise samples with window length of 8 samples in CFAR -CASO mode.
		cyclic mode or Wrapped around mode. 0- Disabled 1- Enabled	supported
		Threshold scale in dB using float representation. This is used in conjuntion with the noise sum divisor (say x). the CUT comparison for log input is: CUT > (Threshold scale converted from dB to Q8) + (noise sum / 2^x) For example: 15 10.75	Detection threshold is specified in dB scale. Maximum value allowed is 100dB
		peak grouping 0 - disabled 1 - enabled	supported
multiObjBeam Forming (TDM)	Multi Object Beamforming config message to datapath. This feature allows radar to separate reflections from multiple objects originating from the same range/Doppler detection.		Specific to TDM.
		<subFrameIdx> subframe Index	For legacy mode, that field should be set to -1 whereas for advanced frame mode, it should be set to either the intended subframe number or -1 to apply same config to all subframes.

	The procedure searches for the second peak after locating the highest peak in Azimuth FFT. If the second peak is greater than the specified threshold, the second object with the same range/Doppler is appended to the list of detected objects. The threshold is proportional to the height of the highest peak. The values in this command can be changed between sensorStop and sensorStart and even when the sensor is running. This is a mandatory command.	<Feature Enabled> 0 - disabled 1 - enabled	supported
		<threshold> 0 to 1 – threshold scale for the second peak detection in azimuth FFT output. Detection threshold is equal to <thresholdScale> multiplied by the first peak height. Note that FFT output is magnitude squared.	supported
calibDcRangeSig (TDM)	DC range calibration config message to datapath. Antenna coupling signature dominates the range bins close to the radar. These are the bins in the range FFT output located around DC. When this feature is enabled, the signature is estimated during the first N chirps, and then it is subtracted during the subsequent chirps. During the estimation period the specified bins (defined as [negativeBinIdx, positiveBinIdx]) around DC are accumulated and averaged. It is assumed that no objects are present in the vicinity of the radar at that time. This procedure is initiated by the following CLI command, and it can be initiated any time while radar is running. Note that the maximum number of compensated bins is 32. The values in this command can be changed between sensorStop and sensorStart and even when the sensor is running. This is a mandatory command.		Specific to TDM.
		<subFrameIdx> subframe Index	For legacy mode, that field should be set to -1 whereas for advanced frame mode, it should be set to either the intended subframe number or -1 to apply same config to all subframes.
		<enabled> Enable DC removal using first few chirps 0 - disabled 1 - enabled	supported
		<negativeBinIdx> negative Bin Index (to remove DC from farthest range bins) Maximum negative range FFT index to be included for compensation. Negative indices are indices wrapped around from far end of 1D FFT. Ex: Value of -5 means last 5 bins starting from the farthest bin	supported
		<positiveBinIdx> positive Bin Index (to remove DC from closest range bins) Maximum positive range FFT index to be included for compensation Value of 8 means first 9 bins (including bin#0)	supported
		<numAvg> number of chirps to average to collect DC signature (which will then be applied to all chirps beyond this). Value of 256 means first 256 chirps (after command is issued and feature is enabled) will be used for collecting (averaging) DC signature in the bins specified above. From 257th chirp, the collected DC signature will be removed from every chirp.	The value must be power of 2, and must be greater than the number of Doppler bins.
clutterRemoval (TDM)	Static clutter removal config message to datapath. Static clutter removal algorithm implemented by subtracting from the samples the mean value of the input samples to the 2D-FFT The values in this command can be changed between sensorStop and sensorStart and even when the sensor is running. This is a mandatory command.		Specific to TDM.
		<subFrameIdx> subframe Index	For legacy mode, that field should be set to -1 whereas for advanced frame mode, it should be set to either the intended subframe number or -1 to apply sameconfig to all subframes.
		<enabled> Enable static clutter removal technique 0 - disabled 1 - enabled	supported
cfarFovCfg (TDM)	Command for datapath to filter out detected points outside the specified limits in the range direction or doppler direction The values in this command can be changed between sensorStop and sensorStart and even when the sensor is running.		Specific to TDM.
		<subFrameIdx> subframe Index	For legacy mode, that field should be set to -1 whereas for advanced frame mode, it should

	This is a mandatory command.		be set to either the intended subframe number or -1 to apply same config to all subframes.
		<procDirection> Processing direction: 0 – point filtering in range direction 1 – point filtering in Doppler direction	both values supported but this command should be given twice - one for range direction and other for doppler direction
		<min (meters or m/s)> the units depends on the value for <procDirection> field above. meters for Range direction and meters/sec for Doppler direction	minimum limits for the range or doppler below which the detected points are filtered out
		<max (meters or m/s)> the units depends on the value for <procDirection> field above. meters for Range direction and meters/sec for Doppler direction	maximum limits for the range or doppler above which the detected points are filtered out
compRangeBiasAndRxChanPhase (TDM)	Command for datapath to compensate for bias in the range estimation and receive channel gain and phase imperfections. Refer to the procedure mentioned here The values in this command can be changed between sensorStop and sensorStart and even when the sensor is running. This is a mandatory command.		Specific to TDM.
		<rangeBias> Compensation for range estimation bias in meters	supported
		<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)> Set of Complex value representing compensation for virtual Rx channel phase bias in Q15 format. Pairs of I and Q should be provided for all Tx and Rx antennas in the device	For AM273X demo: 12 pairs of values should be provided here since the device has 4 Rx and 3 Tx (total of 12 virtual antennas). For AWR2944 demo: 16 pairs of values should be provided here since the device has 4 Rx and 4 Tx (total of 16 virtual antennas).
measureRangeBiasAndRxChanPhase (TDM)	Command for datapath to enable the measurement of the range bias and receive channel gain and phase imperfections. Refer to the procedure mentioned here The values in this command can be changed between sensorStop and sensorStart and even when the sensor is running. This is a mandatory command.		Specific to TDM.
		<enabled> 1 - enable measurement. This parameter should be enabled only using profile calibration. 0 - disable measurement. This should be the value to use for all other profiles.	supported
		<targetDistance> distance in meters where strong reflector is located to be used as test object for measurement. This field is only used when measurement mode is enabled.	supported
		<searchWin> distance in meters of the search window around <targetDistance> where the peak will be searched	supported
extendedMaxVelocity (TDM)	Velocity disambiguation config message to datapath. A simple technique for velocity disambiguation is implemented. It corrects target velocities up to (2*vmax). The output of this feature may not be reliable when two or more objects are present in the same range bin and are too close in azimuth plane. The values in this command can be changed between sensorStop and sensorStart and even when the sensor is running. This is a mandatory command.		Specific to TDM.
		<subFrameIdx> subframe Index	For legacy mode, that field should be set to -1 whereas for advanced frame mode, it should be set to either the intended subframe number or -1 to apply same config to all subframes.
		<enabled> Enable velocity disambiguation technique 0 - disabled 1 - enabled	supported.
CQRxSatMonitor	Rx Saturation Monitoring config message for Chirp quality to RadarSS and datapath. See	<profile> Valid profile Id for this monitoring	not presently supported

	mmwavelink doxgen for details on rlRxSatMonConf_t. The enable/disable for this command is controlled via the "analogMonitor" CLI command. The values in this command can be changed between sensorStop and sensorStart.	configuration. This profile ID should have a matching profileCfg. <satMonSel> RX Saturation monitoring mode <priSliceDuration> Duration of each slice, 1LSB=0.16us, range: 4 -number of ADC samples <numSlices> primary + secondary slices ,range 1-127. Maximum primary slice is 64. <rxChanMask> RX channel mask, 1 - Mask, 0 - unmask	
CQSigImgMonitor	Signal and image band energy Monitoring config message for Chirp quality to RadarSS and datapath. See mmwavelink doxgen for details on rlSigImgMonConf_t. The enable/disable for this command is controlled via the "analogMonitor" CLI command. The values in this command can be changed between sensorStop and sensorStart.	<profile> Valid profile Id for this monitoring configuraiton. This profile ID should have a matching profileCfg <numSlices> primary + secondary slices, range 1-127. Maximum primary slice is 64. <numSamplePerSlice> Possible range is 4 to "number of ADC samples" in the corresponding profileCfg. It must be an even number.	not presently supported
analogMonitor	Controls the enable/disable of the various monitoring features supported in the demos. The values in this command can be changed between sensorStop and sensorStart.	<rxSaturation> CQRxSatMonitor enable/disable 1:enable 0: disable <sigImgBand> CQSigImgMonitor enable/disable 1:enable 0: disable	not presently supported
lvdsStreamCfg	Enables the streaming of various data streams over LVDS lanes. When this feature is enabled, make sure chirpThreshold in adcbufCfg is set to 1. The values in this command can be changed between sensorStop and sensorStart. **This command is valid for only AWR294X device.**	<subFrameIdx> subframe Index	For legacy mode, that field should be set to -1 whereas for advanced frame mode, it should be set to either the intended subframe number or -1 to apply same config to all subframes.
		<enableHeader> 0 - Disable HSI header for all active streams 1 - Enable HSI header for all active streams	Only 0 is supported.
		<dataFmt> Controls HW streaming. Specifies the HW streaming data format. 0-HW STREAMING DISABLED 1-ADC 4-CP_ADC_CQ	When choosing CP_ADC_CQ, please ensure that CQRxSatMonitor and CQSigImgMonitor commands are provided with appropriate values and these monitors are enabled using analogMonitor command.
		<enableSW> 0 - Disable user data (SW session) 1 - Enable user data (SW session) <enableHeader> should be set to 1 when this field is enabled.	Only ADC format is currently supported.
numLvdsLanesCfg	Configure number if LVDS lanes. **This command is valid for only AM273X device.**	<numLvdsLanesCfg> 3 - Enables 2-Lanes for Streaming Raw ADC Data 15 - Enable 4-Lanes for Streaming Raw ADC Data	2-lane/4-lane is Supported.
calibdata	Boot time RF calibration save/restore command.	<save enable>	supported

	Provides user to either save the boot time RF calibration performed by the RadarSS onto the FLASH or to restore the previously saved RF calibration data from the FLASH and instruct RadarSS to not re-perform the boot-time calibration. User can either save or restore or perform neither operations. User is not allowed to simultaneous save and restore in a given boot sequence. Boot time phase shift calibration data is also saved along with all other calibration data. The values in this command should not change between sensorStop and sensorStart. Reboot the board to try config with different set of values in this command This is a mandatory command.	0 - Save enabled. Application will boot -up normally and configure the RadarSS to perform all applicable boot calibrations during mmWave_open. Once the calibrations are performed, application will retrieve the calibration data from RadarSS and save it to FLASH. User need to specify valid <flash offset> value. <restore enable> option should be set to 0. 1 - Save disabled.	
		<restore enable> 0 - Restore enabled. Application will check the FLASH for a valid calibration data section. If present, it will restore the data from FLASH and provide it to RadarSS while configuring it to skip any real-time boot calibrations and use provided calibration data. User need to specify valid <flash offset> value which was used during saving of calibration data. <save enable> option should be set to 0. 1 - Restore disabled.	supported
		<Flash offset> Address offset in the flash to be used while saving or restoring calibration data. Make sure the address doesn't overlap the location in FLASH where application images are stored and has enough space for saving rlCalibrationData_t and rlPhShiftCalibrationData_t This field is don't care if both save and restore are disabled	supported
enetStreamCfg (TDM)	Ethernet streaming configuration command This is a mandatory command if mmwDemoEnet is used.		Specific to TDM.
		<isEnable> 0 - Ethernet streaming of detected object data is enabled 1 - Ethernet streaming of detected object data is disabled	supported
		<remoteIpD> <remoteIpC> <remoteIpB> <remoteIpA> If the IP Address obtained by the PC is D.B.C.A, the arguments to send would be D B C A.	supported
sensorStart	sensor Start command to RadarSS and datapath. Starts the sensor. This function triggers the transmission of the frames as per the frame and chirp configuration. By default, this function also sends the configuration to the mmWave Front End and the processing chain. This is a mandatory command.		
		Optionally, user can provide an argument 'doReconfig' 0 - Skip reconfiguration and just start the sensor using already provided configuration. <any other value> - not supported	supported
sensorStop	sensor Stop command to RadarSS and datapath. Stops the sensor. If the sensor is running, it will stop the mmWave Front End and the processing chain. After the command is acknowledged, a new		supported

	config can be provided and sensor can be restarted or sensor can be restarted without a new config (i.e. using old config). See 'sensorStart' command. This is mandatory before any reconfiguration is performed post sensorStart.		
flushCfg	This command should be issued after 'sensorStop' command to flush the old configuration and provide a new one. This is mandatory before any reconfiguration is performed post sensorStart.		
configDataPort	This is an optional command to change the baud rate of the DATA_port. By default, the baud rate is 892857. This command will be accepted only when sensor is in init state or stopped state i.e. between sensorStop and sensorStart. It is recommended to use this command outside of the CFG file so that PC tools can also be configured to accept data at the desired baud rate.	<baudrate> The new baud rate for the DATA_port. Recommended value: 892857.	supported
		<ackPing> 0 - Do not send any bytes on data port 1- Send 16 bytes of value '0xFF' to ack/sync over the DATA_port (binary) after change to baud rate is applied.	supported
queryDemoStatus	This is an optional command that can be issued anytime to get the sensor state (0-init,1-opened, 2-started,3-stopped) of the device and the current baud rate of the DATA_port. The response of this command is provided on the CLI port.		
queryLocalIp (TDM)	This is an optional command that can be issued anytime via the "Status" window of the visualizer to print out the IP Address obtained by the EVM.		Specific to TDM.
dataFormatConfig	dataFormat config for the datapath. The values in this command can be changed between sensorStop and sensorStart. This is a mandatory command. **This command is valid for only AM273X device.**	<SampleSwap> IQ swap selection: 0-I in LSB, Q in MSB 1-Q in LSB, I in MSB	only option 1 is supported (note: this command does not impact the actual swap in the current demo. Please use this as it is in the files in the profiles/ directory)
		<ChanInterleave> Channel interleave configuration: 0 - non-interleaved 1 - interleaved	only option 0 is supported (note: this command does not impact the actual channel interleave in the current demo. Please use this as it is in the files in the profiles/ directory)
dataPathConfig	mmwave radar data path config. The values in this command can be changed between sensorStop and sensorStart. Refer to mmwavelink documentation for details. See profile_2d_am273x.cfg file for usage This is a mandatory command. **This command is valid for only AM273X device.**		
		<intfSel> Data Path Interface, 0 - CSI2 interface selected 1 - LVDS interface selected	only option CSI2 is supported
		<transferFmtPkt0> Data out Format, b5:0 Packet 1 content selection 000000 - Suppress Packet 1 001110 - CP_CQ_DATA 001011 - CQ_CP_DATA b7:6 Packet 1 virtual channel number (valid only for CSI2) 00 Virtual channel number 0 (Default) 01 Virtual channel number 1 10 Virtual channel number 2 11 Virtual channel number 3	as per mmwavelink documentation
		<transferFmtPkt1> Data out Format,	as per mmwavelink documentation

		b5:0 Packet 1 content selection 000000 - Suppress Packet 0 001110 - CP_CQ_DATA 001011 - CQ_CP_DATA b7:6 Packet 1 virtual channel number (valid only for CSI2) 00 Virtual channel number 0 (Default) 01 Virtual channel number 1 10 Virtual channel number 2 11 Virtual channel number 3	
		<cqConfig> Field specifies the data size of CQ samples on the lanes. b1:0 Data size 00 12 bits 01 14 bits 1 0 16 bits b7:2 Reserved	as per mmwavelink documentation
		<cq0TransSize> Number of samples (in 16 bit halfwords) of CQ0 data to be transferred. Valid range [32 halfwords to 128 halfwords] 0 - Disabled	Ensure that the number of halfwords specified are a multiple of the number of lanes selected. Value used in default demo config-64
		<cq1TransSize> Number of samples (in 16 bit halfwords) of CQ1 data to be transferred. Valid range [32 halfwords to 128 halfwords] 0 - Disabled	Ensure that the number of halfwords specified are a multiple of the number of lanes selected. Value used in default demo config-64
		<cq2TransSize> Number of samples (in 16 bit halfwords) of CQ2 data to be transferred. Valid range [32 halfwords to 128 halfwords] 0 - Disabled	Ensure that the number of halfwords specified are a multiple of the number of lanes selected. Value used in default demo config-64
hsiClockConfig	High Speed Interface Clock configurations. The values in this command can be changed between sensorStop and sensorStart. Refer to mmwavelink documentation for details. See profile_2d_am273x.cfg file for usage This is a mandatory command. **This command is valid for only AM273X device.**		
		<hsiClk> High Speed Interface Clock configurations. HSICLKRATECODE (corresponding datarate in Mbps): SDR - 0x5(900 mbps), 0xA(600 mbps), 0x6(450 mbps), 0x2(400 mbps), 0xB(300 mbps), 0x7(225 mbps) DDR - 0xD(900 mbps), 0x9(600 mbps), 0x5(450 mbps), 0x1(400 mbps), 0xA(300 mbps), 0x6(225 mbps), 0xB(150 mbps)	should be 9 (DDR - 600 Mbps supported)
		<reserved>	should be 0
hsiLaneConfig	CSI2 configuration. The values in this command can be changed between sensorStop and sensorStart. Refer to mmwavelink documentation for details. See profile_2d_am273x.cfg file for usage This is a mandatory command. **This command is valid for only AM273X device.**		
		<laneEn> CSI Lane Enable	should be 15 refer to mmwavelink documentation for more details
		<DATA_LANE0_POS> <DATA_LANE0_POL> <DATA_LANE1_POS> <DATA_LANE1_POL> <DATA_LANE2_POS> <DATA_LANE2_POL> <DATA_LANE3_POS> <DATA_LANE3_POL> CSI Lane Position and Polarity	should be 1 0 2 0 4 0 5 0 refer to mmwavelink documentation for more details
		<CLOCK_POS> <CLOCK_POL> CSI Clock Position and Polarity	should be 3 0 refer to mmwavelink documentation for more details

		<csi2_lineStartEndDis> CSI2 Line Start and End 0 - Disable 1 - Enable	0 recommended
dataPathClkCfg	DataPath clock configuration. The values in this command can be changed between sensorStop and sensorStart. Refer to mmwavelink documentation for details. See profile_2d_am273x.cfg file for usage This is a mandatory command. **This command is valid for only AM273X device.**	<laneClkCfg> Clock Configuration 0 - SDR Clock 1 - DDR Clock (Only valid value for CSI2)	Only DDR Clock is supported
		dataRate Data rate selection 0001b - 600 Mbps (DDR only) 0010b - 450 Mbps (SDR, DDR) 0011b - 400 Mbps (DDR only) 0100b - 300 Mbps (SDR, DDR) 0101b - 225 Mbps (DDR only) 0110b - 150 Mbps (DDR only) Others - Reserved	600 Mbps supported
compressionCfg (DDM)	Compression / Decompression configuration This is a mandatory command.		Specific to DDM.
		<enabled>: 1 if enabled	must be enabled, since disabling compression is not currently supported
		<compressionRatio>: compression ratio	floating point number between 0 and 1
		<rangeBinsPerBlock>: number of range bins to compress per block	must be a power of 2
		<compressionMethod>	only 0 is supported
intfMitigationCfg (DDM)	Interference mitigation configuration. Enabled by default in the current release. Mandatory command.		Specific to DDM.
		<subFrameNum>: subframe number	must be -1
		<magSNRdB>: mag SNR for interference mitigation	integer val in dB
		<magDiffSNRdB>: magDiff SNR for interference mitigation	integer val in dB
localMaxCfg (DDM)	Local max configuration. Enabled by default in the current release. Mandatory command.		Specific to DDM.
		<azimThreshdB>: azimuth threshold for local max paramset	integer val in dB
		<dopplerThreshdB>: doppler threshold for local max paramset	integer val in dB
antennaCalibParams (DDM)	Antenna calibration parameters Mandatory command.		Specific to DDM.
		<Q0> <I0> …. <Q15> <I15>: antenna calibration parameters for the 16 virtual antennas of AWR294X in Im(Q)-Re(I) format For AWR2943, the last 8 values will be ignored.	Im(Q)-Re(I) format
cfarCfg (different from cfarCfg used in TDM chain) (DDM)	cfar configuration Mandatory command.		Specific to DDM.
		< procDirection> Processing direction: 0 – CFAR detection in range direction 1 – CFAR detection in Doppler direction	Both values supported
		<averageMode> CFAR OS is supported	must be 3 (CFAR-OS)
		<winLen> noise averaging window length: Length of the one sided noise averaged cells in samples	supported

		Make sure 2*(noiseWIn+guardLen) <numRangeBins for range direction and 2*(noiseWIn+guardLen) <numDopplerBins for doppler direction.	
		<guardLen> guard length	must be 0 (for CFAR-OS)
		<noiseDivShift> Cumulative noise sum divisor expressed as a shift.	noise div shift
		<cyclicMode> cyclic mode 0- Disabled 1- Enabled	supported
		<threshold> CFAR detection threshold in dB	Detection threshold is specified in dB scale.
		<peakGroupingEn> Peak grouping 0: Enabled 1: Disabled	supported
		<osKvalue> CFAR-OS Edge K Scale Value	See model profile configuration file in demo/awr294x/mmw/profiles for example configuration.
		<osEdgeKscaleEn> CFAR-OS Edge K Scale Value	See model profile configuration file in demo/awr294x/mmw/profiles for example configuration.
%		Any line starting with '%' character is considered as comment line and is skipped by the CLI parsing utility.	

3.6.1.1.2. Table 1: mmWave SDK Demos - CLI commands and parameters

3.7. Running the prebuilt unit test binaries (.xer5f and .xe66)

Unit tests for the drivers and components can be found in the respective test directory for that component. See section "mmWave SDK - TI components" for location of each component's test code. For example, the Range Processing DPU test codes (that can run on RF5 and C66x separately) can be found in <mmwave_mcuplus_sdk_mcuplus_ver>/ti/datapath/dpu/rangeproc/test. In this test directory, you will find .xer5f and .xe66 files (either prebuilt or build as a part of instructions mentioned in "Building datapath/control components"). Follow the instructions in section "CCS development mode" to download and execute these unit tests via CCS.

4. How-To Articles

4.1. How to identify the COM ports for mmWave EVM

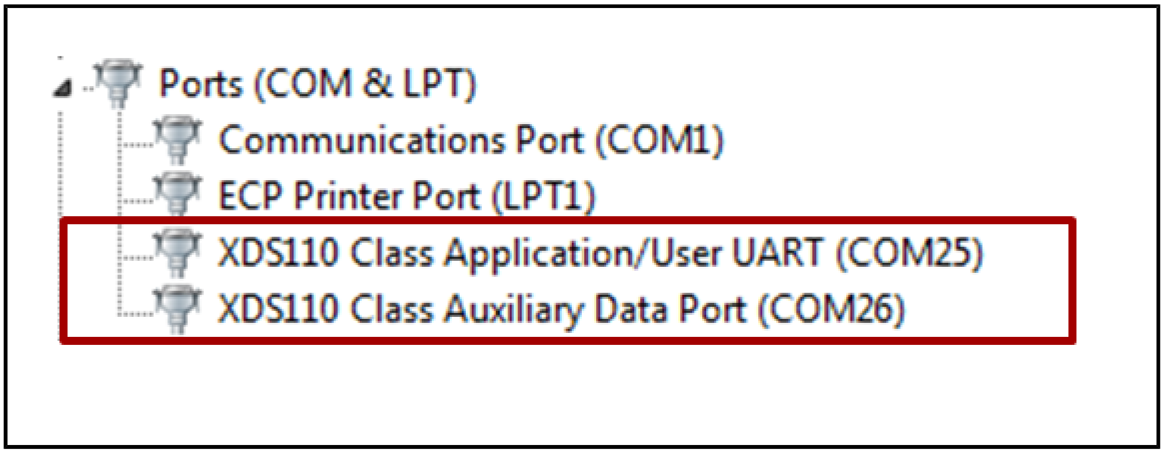
When the EVM is powered on and connected to Windows PC via the supplied USB cable, there should be two additional COM Ports in Device Manager. See your mmWave devices' TI EVM User Guide for details on the COM port.

Troubleshooting Tip

If the COM ports don't show up in the Device Manager or are not working (i.e. no demo output seen on the data port), then one of these steps would apply depending on your setup:

1. If you want to run the Out-of-box demo, simple browse to the Visualizer (https://dev.ti.com/gallery/view/mmwave/mmWave_Demo_Visualizer/ver/4.2.0/) and follow the one-time setup instructions.
2. If you are trying to flash the board, using Uniflash command line tool and desktop version installation instructions would also install the right drivers for the COM ports.
3. If above methods didn't work and if TI code composer studio is not installed on that PC, then download and install the standalone XDS110 drivers.
4. If TI code composer studio is installed, then version of CCS and emulation package need to be checked and updated as per the mmWave SDK release notes. See section Emulation Pack Update for more details.

After following the above steps, disconnect and re-connect the EVM and you should see the COM ports now. See the highlighted COM ports in the Figure below



Ports (COM & LPT)
Communications Port (COM1)
ECP Printer Port (LPT1)
XDS110 Class Application/User UART (COM25)
XDS110 Class Auxiliary Data Port (COM26)

4.1.1.1.1. Figure 3: mmWave EVM PC Connectivity - Device Manager - COM Ports

COM Port

Please note that the COM port numbers on your setup maybe different from the one shown above. Please use the correct COM port number from your setup for following steps.

Go back to: How to flash an image onto mmWave EVM

4.2. How to flash an image onto mmWave EVM

You will need the mmWave Device TI EVM, USB cable and a Windows/Linux PC to perform these steps.

1. **Setup the EVM for Flashing**

   Refer to the EVM User Guide to understand the bootup modes of the EVM and the SOP jumper/switch locations (See "Sense-on-Power (SOP) Jumpers" section in mmWave device's EVM user guide). The table detailing the boot modes and the figure with the SOP jumper /switch locations are provided below for quick reference.

SOP0 **jumper /switch**	**SOP1** **jumper /switch**	**SOP2** **jumper /switch**	**Bootloader mode & operation**
1	1	0	**No-Boot Mode** ROM does not execute in this boot mode.
1	0	1	**UART Boot Mode** ROM receives the secondary boot loader via UART interface MSS SCI A and loads it in SoC memory and switches to SBL execution. SBL receives multicore application image via UART interface MSS SCI A and loads it in SoC memory and switches to application image execution.
1	0	0	**QSPI Boot Mode** ROM loads the SBL from serial flash at flash offset of 0x0, and SBL loads the multicore appimage from serial flash at flash offset of 0xA0000.

   SOP Switch Order

   Be cautious when setting the SOP jumpers as AM273X and AWR294x EVM have slight difference in order of SOP jumpers.

2. **Procure the Images**
   For flashing AM273X /AWR294X devices, flashing tool at mcu_plus_sdk_<platform>_<version>\tools\boot\uart_uniflash.py should be used. For details related to tools required for flashing procedure, please refer "Flashing Tools" section of mcu_plus_sdk api_guide available at mcu_plus_sdk_<platform>_<version>\docs\api_guide_<platform>. The flashing procedure requires the flashing of three binaries, the first two (flash programmer and SBL) of which are present in the mmwave_mcuplus_sdk_<version>\tools\<platform> folder. The third step is to flash the application image (with an extentsion *.appimage*) of your choice (for example, CCS Debug binary or demo binary). For the SDK packaged demos, there is a *.appimage* file provided in their respective folder: mmwave_mcuplus_sdk_<version>\ti\demo\<platform>\mmw\<platform>_<demo>.appimage which is the metaImage to be used for flashing. The metaImage already has the MSS, DSS and BSS (as applicable) application combined into one file. Users can use the flashing procedure given below to flash the metaImage of their custom demo as well.
3. **Setup the Build Environment**
   a. Run the setenv script to set up the build environment, as shown in the following code block. Note that for UNIX, the shell script which performs this step is located in mmwave_mcuplus_sdk_<version>\scripts\unix (use source ./setenv.sh as mentioned in this section).

   Setting up Build Environment variables

```
@REM Change the path/version
cd C:\ti\mmwave_mcuplus_sdk_<version>\scripts\windows
@REM Remember to change the variable "MMWAVE_SDK_DEVICE" in this file!
setenv.bat
```

4. **Flashing procedure**

   a. Power up the EVM and check the Device Manager in your windows PC. Note the number for the serial port marked as " **XDS110 Class Application/User UART** " for the EVM. Refer to How to identify the COM ports for mmWave EVM section to identify the COM port. Lets say for this example, it showed up as COM25. The steps for flashing the binaries to the EVM are as follows:
   b. Switch to **UART Boot Mode (SOP0: 1, SOP1: 0, SOP2: 1)** and power cycle the EVM.
   c. The flashing procedure requires the flashing of three binaries, the first two of which are present in the mmwave_mcuplus_sdk_<version>\tools\<platform> folder. The third step is to flash the image (with an extentsion *.appimage*) of your choice (for example, CCS Debug binary or demo binary). The following code blocks shows the steps, which must be executed in the UART Boot Mode.

   Flashing the appimage onto the EVM

```
@REM Step(i): updated the default.cfg file available at %MMWAVE_SDK_INSTALL_PATH%
\tools\<platform> with user's .appimage path
@REM Example ../../ti/utils/ccsdebug/%Platform%_ccsdebug.appimage
@REM Example ../../ti/utils/ccsdebug/%Platform%_ccsdebug.appimage

@REM Step(ii): cd to path
cd %MMWAVE_SDK_INSTALL_PATH%/tools/<platform>

@REM Step(iii): Command to flash the files
@REM The <COM Port> can be, for example, COM25
@REM For AM273X path would be %MCU_PLUS_AM273X_INSTALL_PATH%/tools/boot/uart_uniflash.py
@REM For AWR294x path would be %MCU_PLUS_AWR294X_INSTALL_PATH%/tools/boot/uart_uniflash.py
python %MCU_PLUS_AM273X_INSTALL_PATH%/tools/boot/uart_uniflash.py -p <COM Port> --cfg=default.
cfg
```

5. **Switch to the QSPI Boot Mode and power-cycle the EVM**

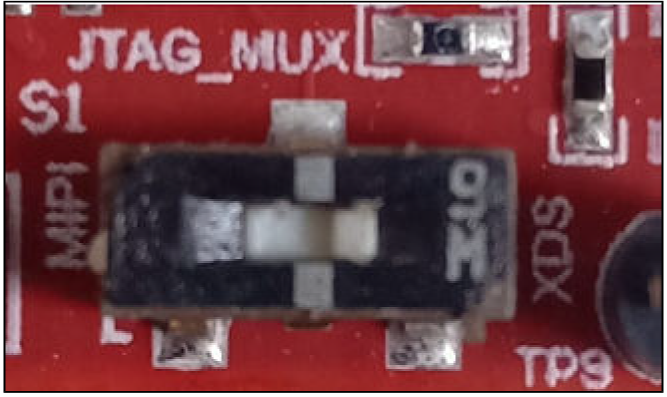
Go back to: How to connect mmWave EVM to CCS using JTAG, Loading images onto mmWave EVM -->Demonstration Mode

4.3. How to connect mmWave EVM to CCS using JTAG

Debug/JTAG capability is available via the same XDS110 micro-USB port/cable on the EVM. TI Code composer studio would be required for accessing the debug capability of the device.

> ⚠ **FTDI**
>
> As SOP setting is also driven by FTDI cable, do not connect FTDI cable while executing any of the SDK applications.

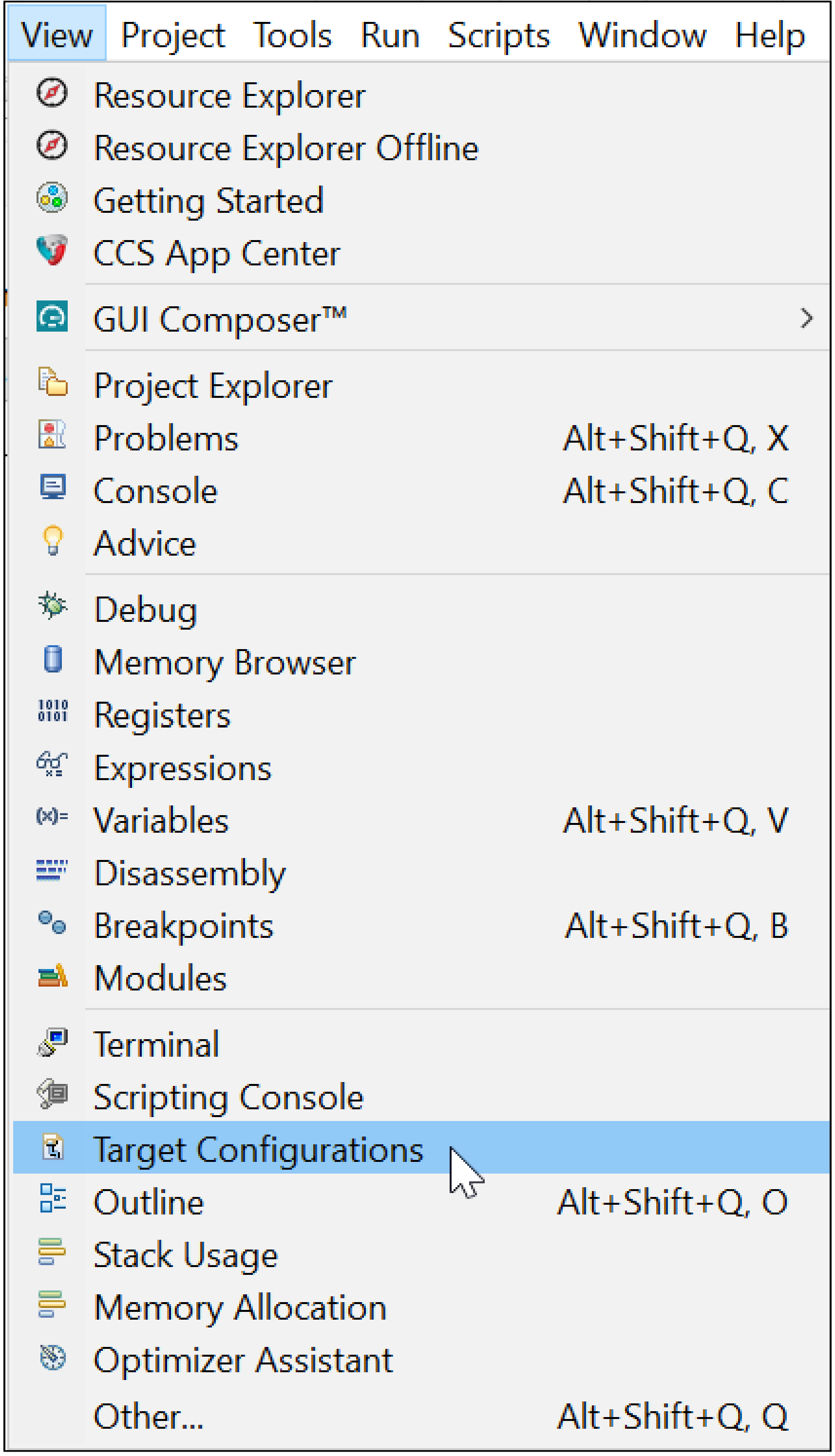
1. Use the instructions present in the How to flash an image onto mmWave EVM to flash the CCS Debug binary (user will need to give the path to the CCS Debug binary: %MMWAVE_SDK_INSTALL_PATH%\ti\utils\ccsdebug\%MMWAVE_SDK_DEVICE%_ccsdebug.appimage in Step 4.c.(iv) of the flashing instructions).
2. Ensure JTAG mux is set to onboard XDS110, as in the figure below and device is in QSPI boot mode.



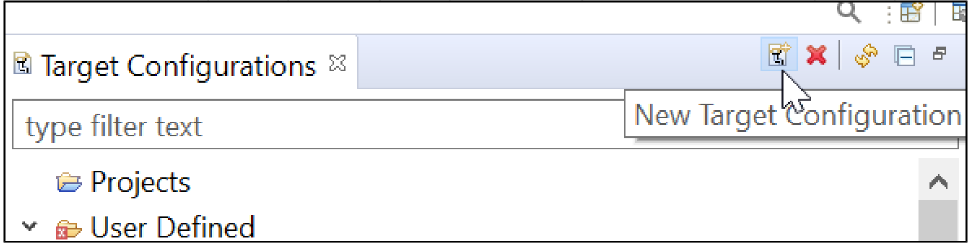
3. Connect both micro-USB cables from EVM to PC. One is for UART/JTAG and the other is for FTDI.
4. Target Configuration file for CCS (CCXML)

   Create a target configuration for the EVM (for example, *<device>evm_xds110.ccxml*). Follow the steps mentioned below for the same.

a. If your CCS window does not already show "Target Configurations" pane, go to *View->Target Configurations*

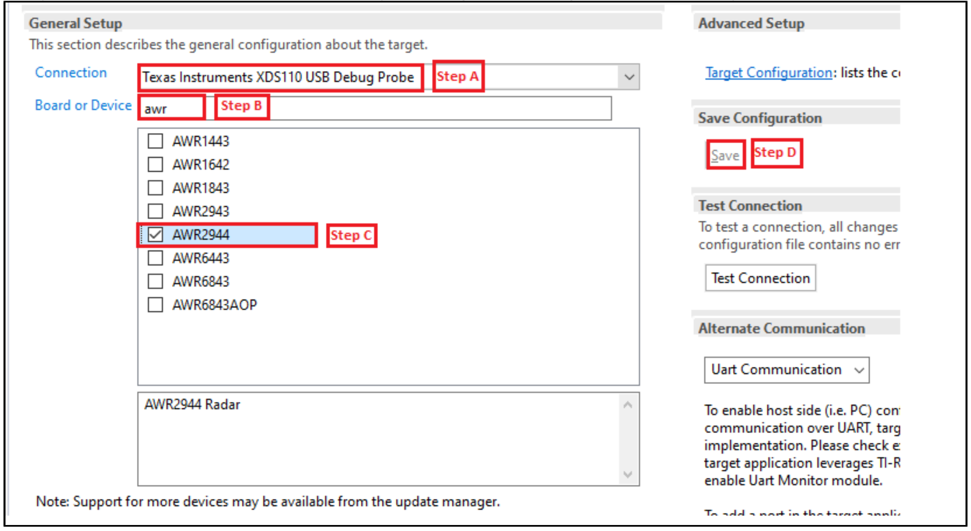


b. This will show the "Target Configurations" pane. right click in this pane and select "New Target Configuration".



c. Give an appropriate name to the .ccxml file you want to create for the EVM

d. As shown in the figure below, scroll the "Connection" list and select "Texas Instruments XDS110 USB Debug Probe" (Step A). When this is done, the "Board or Device" list will be filtered to show the possible candidates, find (Step B) and choose the mmWave device of interest (AWR294X/AM273X) and check the box (Step C). Click save (Step D) to store this ccxml file. Note that if you don't find your desired device in the list, it is likely that step 4 was not performed correctly.



General Setup

This section describes the general configuration about the target.

Connection: Texas Instruments XDS110 USB Debug Probe

Board or Device: awr

- □ AWR1443
- □ AWR1642
- □ AWR1843
- □ AWR2943
- ☑ AWR2944
- □ AWR6443
- □ AWR6843
- □ AWR6843AOP

AWR2944 Radar

Note: Support for more devices may be available from the update manager.

Advanced Setup

Target Configuration: lists the c

Save Configuration

Test Connection

To test a connection, all changes configuration file contains no err

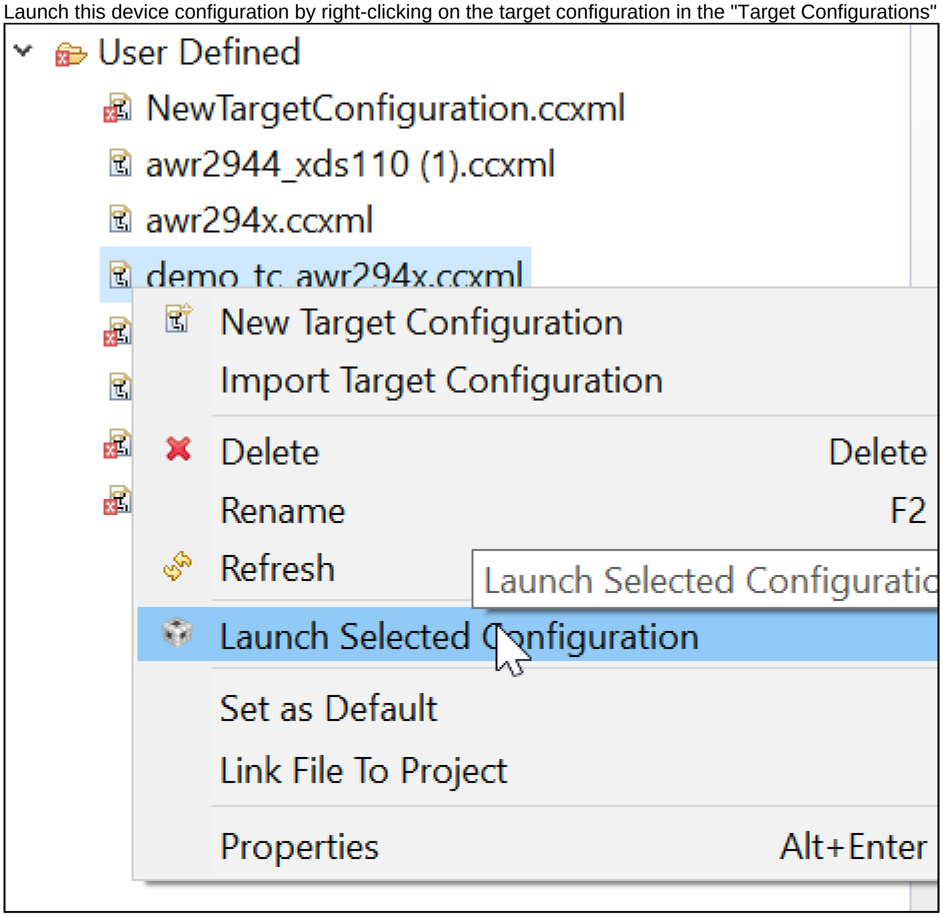
Alternate Communication

Uart Communication

To enable host side (i.e. PC) con communication over UART, targ implementation. Please check e target application leverages TI-R enable Uart Monitor module.

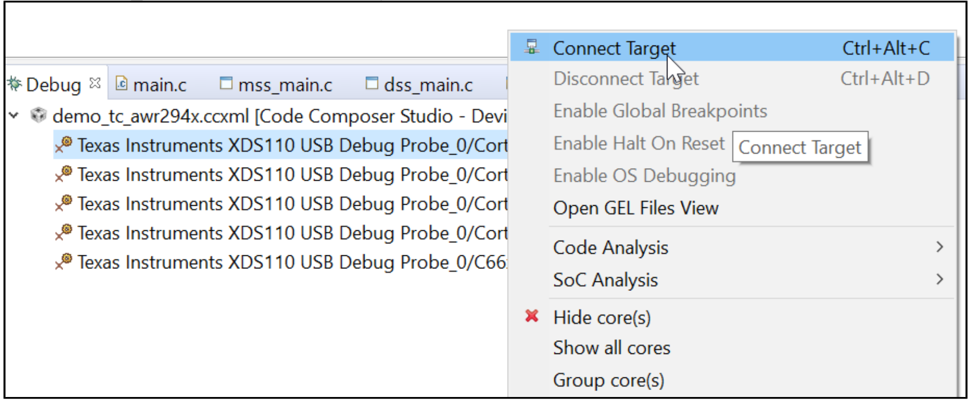
> **CCS Configuration**
>
> Steps 4 and 5 above are a one-time configuration. Once a target configuration is saved, the user can simply load it again as it will be listed as an existing configuration.

5. Launch this device configuration by right-clicking on the target configuration in the "Target Configurations" pane.

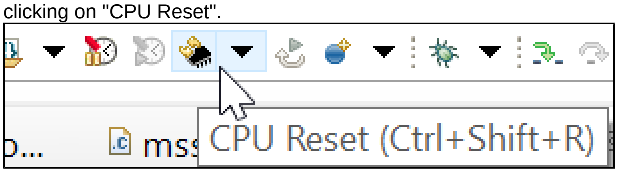


6. This will list three cores of AWR294X R5_0, R5_1 and C66x in the Debug pane. If you don't see this pane, go to ViewDebug.

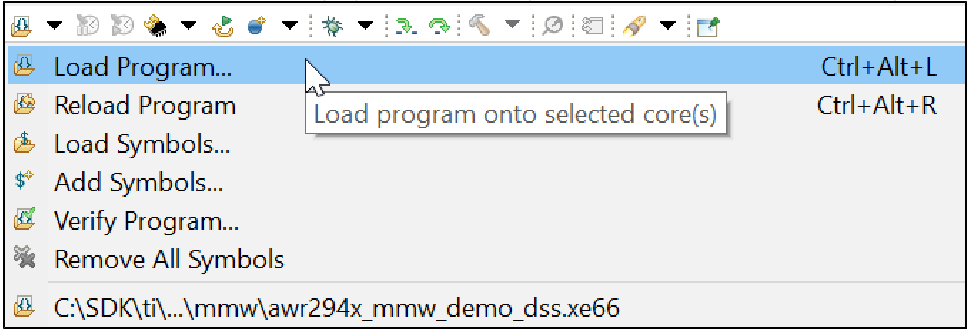
7. Connect to R5_0 and/or C66x in the Debug window.


Debug
main.c
mss_main.c
dss_main.c
demo_tc_awr294x.ccxml [Code Composer Studio - Devi
Texas Instruments XDS110 USB Debug Probe_0/Cort
Texas Instruments XDS110 USB Debug Probe_0/Cort
Texas Instruments XDS110 USB Debug Probe_0/Cort
Texas Instruments XDS110 USB Debug Probe_0/Cort
Texas Instruments XDS110 USB Debug Probe_0/C66
Connect Target Ctrl+Alt+C
Disconnect Target Ctrl+Alt+D
Enable Global Breakpoints
Enable Halt On Reset
Connect Target
Enable OS Debugging
Open GEL Files View
Code Analysis
SoC Analysis
Hide core(s)
Show all cores
Group core(s)

8. After connecting to either Cortex_R5_0 or C66xx_DSP core, **perform a CPU Reset** on each of the cores by selecting them one by one and clicking on "CPU Reset".



9. The EVM is ready to use. To load a program onto a core, right click on the core and select 'Load Program'.


Load Program... Ctrl+Alt+L
Load program onto selected core(s)
Reload Program Ctrl+Alt+R
Load Symbols...
Add Symbols...
Verify Program...
Remove All Symbols
C:\SDK\ti\...\mmw\awr294x_mmw_demo_dss.xe66

Go back to: CCS development mode

4.3.1. Emulation Pack Update

Refer to the mmWave SDK release notes for the emulation pack version that would be needed within CCS to connect to the EVM. Check if that particular or its later version of "TI Emulators" is available within your CCS installation. If you have an older version on your system, refer to CCS help on how to update software packages within CCS.

4.4. How to run mmWave demo with LVDS-based instrumentation

4.4.1. AWR294X

Mmwave Studio CLI tool available in TI Resource Explorer should be used to capture raw ADC data and post process captured data. mmwave Studio cli has inbuilt mss application to configure, capture and post process raw ADC data.

AM273X

Mmwave Studio CLI tool currently doesn't support raw ADC data capture for AM273X + AWR2243. mmwave mcuplus SDK has separate demo application to capture Raw ADC data. Demo binaries are available at mmwave_sdkmmwave_mcuplus_sdk_<ver>/ti/demo/am273x/mmw/am273x_mmw_demo<Proc_Chain>LVDS.appimage. LVDS Demo application configures L3 memory for capturing CSIRX data instead of HWA memory. Unlike the standard Demo application we do not directly pump Chirp data into HWA Memory in this case. We pump the chirp data through CSIRX into an L3 buffer from where EDMA transfers the data to the HWA Memory for range processing. This is done because if the data were pumped directly into the HWA memory, the CBUFF and HWA will try to access the HWA memory at the same time (HWA for range processing and CBUFF for LVDS-based data capture), and such simultaneous is not allowed for HWA memories.

Follow these steps for capturing Raw ADC data:

1. Load demo applications.
2. Connect to visualizer (https://dev.ti.com/gallery/view/mmwave/mmWave_Demo_Visualizer/ver/4.2.0/
3. Copy AM273X_Capture.json file from mmwave_sdkmmwave_mcuplus_sdk_<ver>/tools/data_capture to C:/ti/mmwave_studio_03_00_00_14/mmWaveStudio/PostProc
4. Before loading profile, open a command prompt at C:/ti/mmwave_studio_03_00_00_14/mmWaveStudio/PostProc. and issue below commands.
   a. DCA1000EVM_CLI_Control.exe fpga AM273X_Capture.json
   b. DCA1000EVM_CLI_Control.exe start_record AM273X_Capture.json
   c. AM273X_Capture.json file configures to save the captured LVDS data at "C:\ti\data_capture\am273x__Raw_0"
5. Load profile config from visualizer. Captured Raw ADC data should be available "C:\ti\data_capture\am273x__Raw_0"

To enable mmWave demo in the instrumentation usecase where high bandwidth data is shipped from the device to a PC over LVDS interface and saved onto the filesystem, please follow the following steps:

1. Power on the EVM in functional mode with right demo binary loaded or in the CCS Development mode with the demo programs loaded. Connect mmWave EVM to DCA1000 EVM by following DCA1000EVM Data Capture Card User's Guide. Place SW2.5 in CONFIG_VIA_SW mode to be able to use the remainder of the instructions specified here. In case of CCS Development mode, run the loaded programs through CCS.
2. Open a command prompt and navigate to the mmwave_mcuplus_sdk_<version>/tools/studio_cli/mmw_cli_tool folder.
3. Run the following command: *mmwave_studio_cli.exe*

4.5. How to Run MMWAVE SDK OOB Demo with Ethernet Streaming Enabled

4.6. Setting up

4.6.1.1. Connections

Make the connections as follows:

1. Ethernet connection between EVM and multiport router.
2. Ethernet connection between multiport router and PC.

4.6.1.2. Software

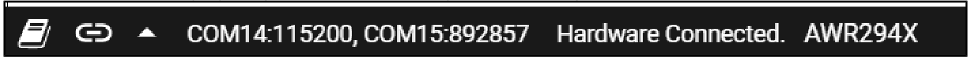
Ensure the following additional software are installed:

1. Python >=v3.7.4 (https://www.python.org/downloads/release/python-374/) with the following libraries:
   a. *socket*: for socket programming in Python
   b. *struct*: aids processing of captured data
   c. *ctypes*: aids processing of captured data

4.6.1.3. Capturing Detected Object Data over Ethernet using OOB Demo

The pre-built binaries of the OOB Demo of MMWAVE SDK have ethernet streaming enabled TDM processing chain.

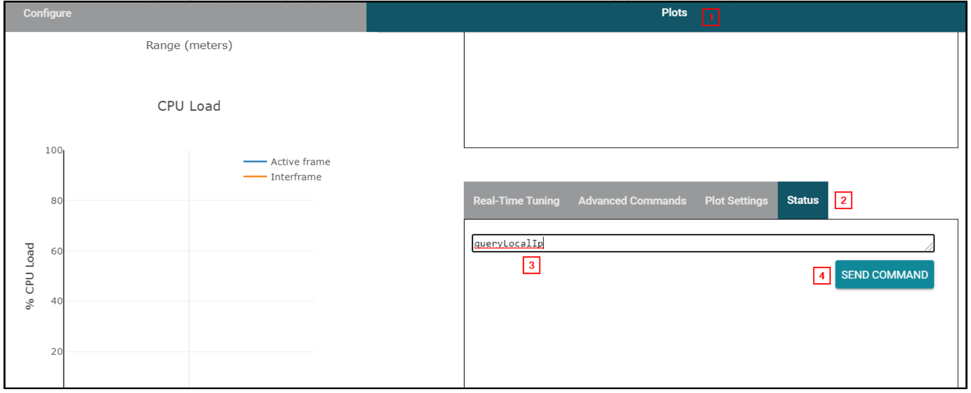
1. Load the demo and run it.
2. Ensure that the demo is running and connect to the MMWAVE Demo Visualizer. The visualizer should show "Hardware Connected." like shown in the following figure (note: COM Ports might be different),



COM14:115200, COM15:892857 Hardware Connected. AWR294X

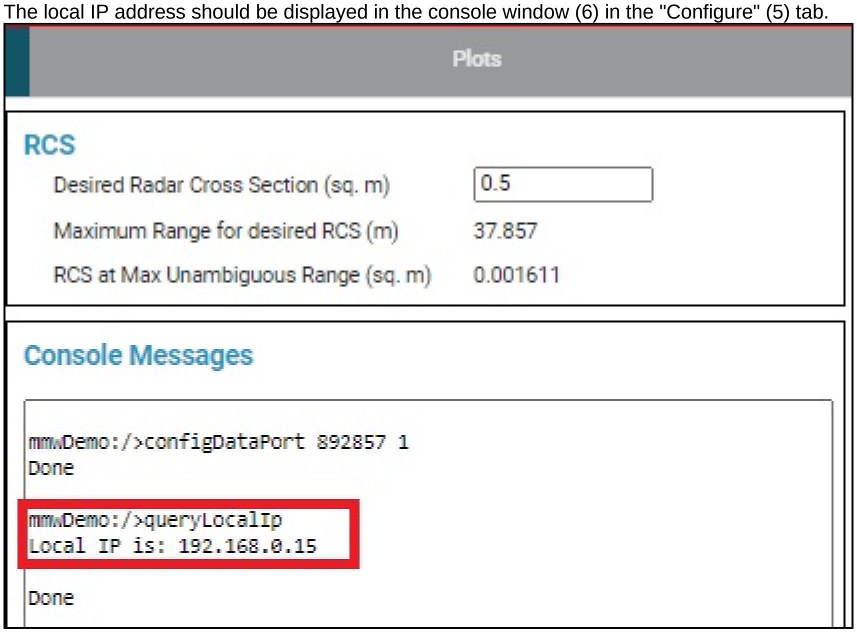
3. Perform the following steps in the demo visualizer to obtain the IP address of the EVM. The figure also shows the steps:
   a. Open the "Status" pane (2) in the "Plots" tab in the visualizer (1).

b. In the text box, type *queryLocalIp* (3) and click on "Send Command" (4). This will query the IP address obtained by the EVM.



```
queryLocalIp
```

c. The local IP address should be displayed in the console window (6) in the "Configure" (5) tab.



RCS

Desired Radar Cross Section (sq. m) 0.5

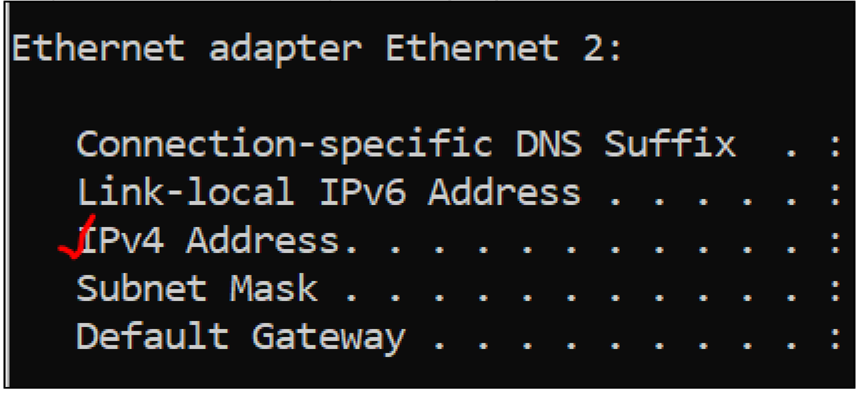
Maximum Range for desired RCS (m) 37.857

RCS at Max Unambiguous Range (sq. m) 0.001611

Console Messages

```
mmwDemo:/>configDataPort 892857 1
Done

mmwDemo:/>queryLocalIp
Local IP is: 192.168.0.15

Done
```

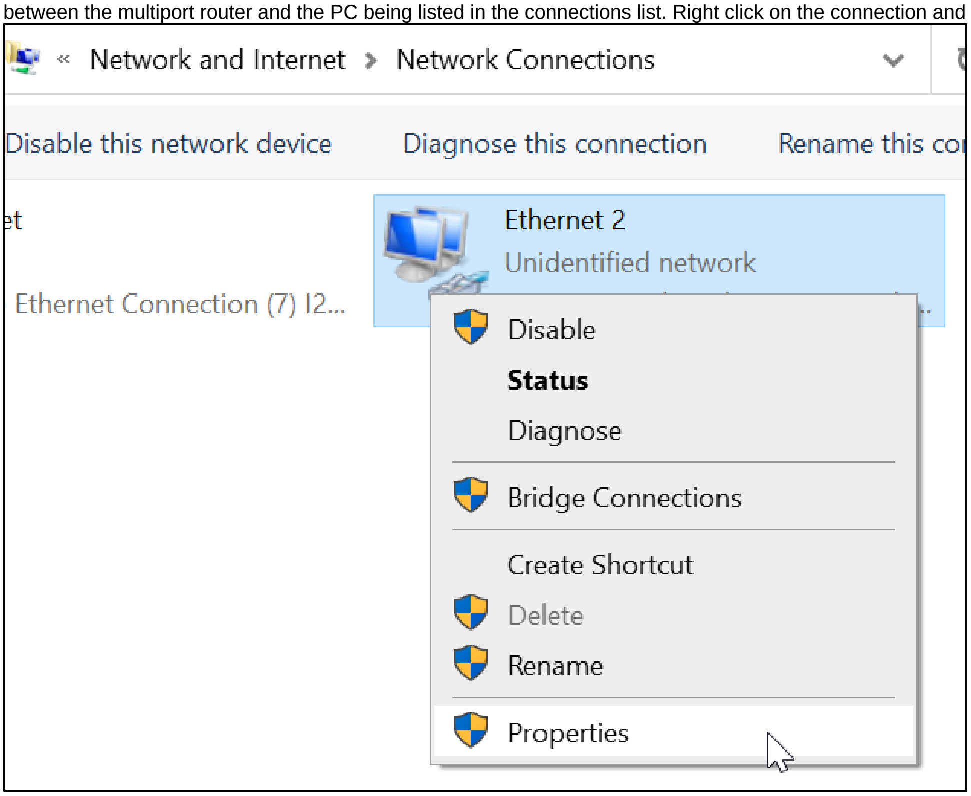
4. Open a command prompt and type the command *ipconfig* to get the IP address of the newly configured network. We call this the "RemoteIP". Note the IPv4 address of the network as shown in the following figure. Ensure that the IP address is in the same subnet as the one obtained in step 2. If that is not true, follow step 4. **Else, skip step 4**.



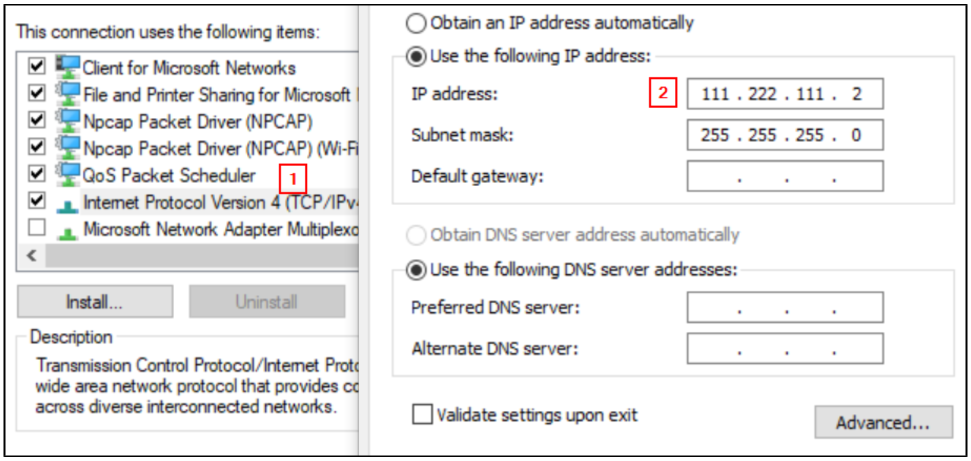
```
Ethernet adapter Ethernet 2:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . :
   IPv4 Address. . . . . . . . . . . :
   Subnet Mask . . . . . . . . . . . :
   Default Gateway . . . . . . . . . :
```

5. Configure a static IP address for the PC. Ensure that the configured IP address is in the same subnet as the EVM IP address obtained in step 2. To do this, perform the following (these steps are given for Windows 10, the user may check how to configure a static IP address for their own system):

a. Go to Control Panel\Network and Internet\Network Connections. You should see the Ethernet connection that you have made between the multiport router and the PC being listed in the connections list. Right click on the connection and click on "Properties".



b. After the network properties pane opens, click on "Internet Protocol Version 4 (TCP/IPv4)" (1) and give the desired IP address (2). Ensure that this address and the EVM IP are in the same subnet.


This connection uses the following items:

☑ Client for Microsoft Networks
☑ File and Printer Sharing for Microsoft
☑ Npcap Packet Driver (NPCAP)
☑ Npcap Packet Driver (NPCAP) (Wi-Fi
☑ QoS Packet Scheduler
☑ Internet Protocol Version 4 (TCP/IPv
☐ Microsoft Network Adapter Multiplexo

Description
Transmission Control Protocol/Internet Prot wide area network protocol that provides c across diverse interconnected networks.

○ Obtain an IP address automatically
● Use the following IP address:
IP address: 111 . 222 . 111 . 2
Subnet mask: 255 . 255 . 255 . 0
Default gateway: . . .

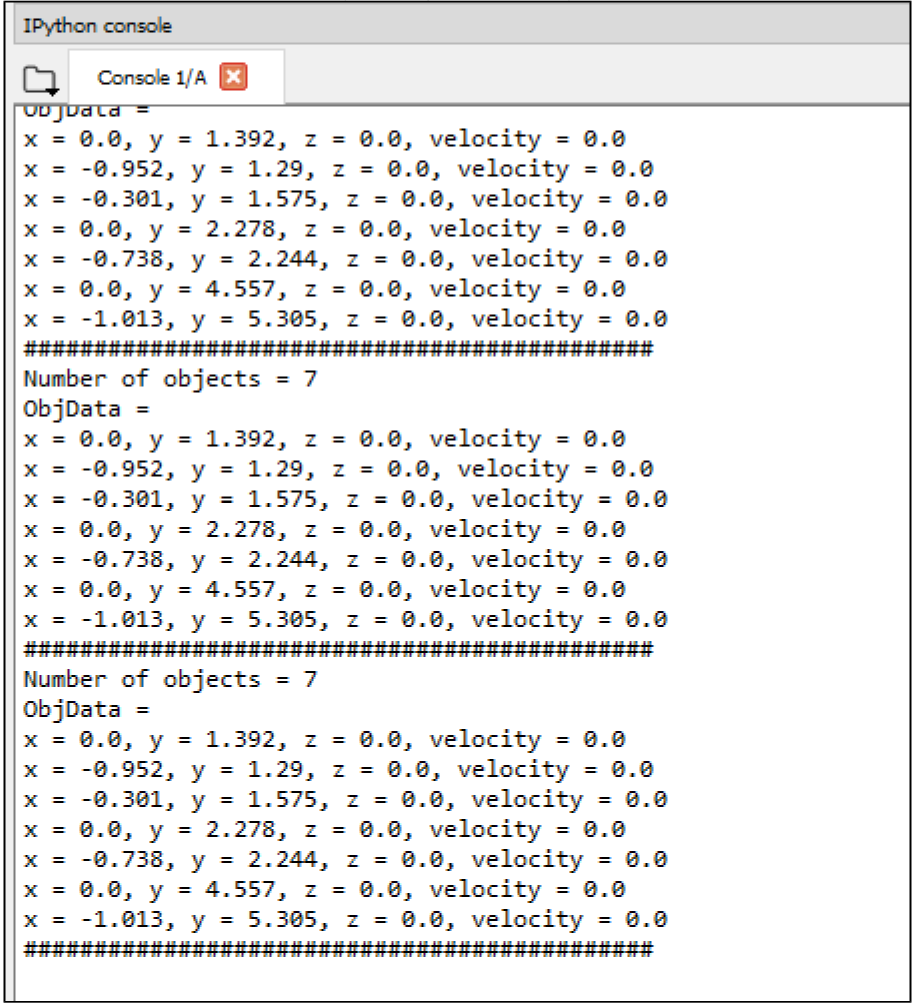
○ Obtain DNS server address automatically
● Use the following DNS server addresses:
Preferred DNS server: . . .
Alternate DNS server: . . .

☐ Validate settings upon exit

c. Click on OK.

6. Open the python script "tcpserver.py" present in mmwave_mcuplus_sdk_<ver>\ti\demo\utils. Change the HOST IP address obtained in step 3 or 4 and configure it in the line "HOST =" in the Python file. For example, if the remote IP obtained in step 3 or 4 is 111.222.111.2, modify the tcpserver.py file to say "*HOST = '111.222.111.2' # Local IP Address*".
7. Execute tcpserver.py.
8. In the profile configuration (.cfg) file *profile_enet.cfg* present in mmwave_mcuplus_sdk_<ver>\ti\demo\<platform>\mmw\profiles folder, modify the enetStreamCfg command before *sensorStart* as follows ***enetStreamCfg <isEnabled> <remoteIpD> <remoteIpC> <remoteIpB> <remoteIpA>***, where the IP address of the PC is *D.C.B.A*. For example, *enetStreamCfg 1 111 222 111 2*, and send this configuration file out through the visualizer.

9. You should be able to see detected object data printed on the Python console.



```
ObjData =
x = 0.0, y = 1.392, z = 0.0, velocity = 0.0
x = -0.952, y = 1.29, z = 0.0, velocity = 0.0
x = -0.301, y = 1.575, z = 0.0, velocity = 0.0
x = 0.0, y = 2.278, z = 0.0, velocity = 0.0
x = -0.738, y = 2.244, z = 0.0, velocity = 0.0
x = 0.0, y = 4.557, z = 0.0, velocity = 0.0
x = -1.013, y = 5.305, z = 0.0, velocity = 0.0
##############################################
Number of objects = 7
ObjData =
x = 0.0, y = 1.392, z = 0.0, velocity = 0.0
x = -0.952, y = 1.29, z = 0.0, velocity = 0.0
x = -0.301, y = 1.575, z = 0.0, velocity = 0.0
x = 0.0, y = 2.278, z = 0.0, velocity = 0.0
x = -0.738, y = 2.244, z = 0.0, velocity = 0.0
x = 0.0, y = 4.557, z = 0.0, velocity = 0.0
x = -1.013, y = 5.305, z = 0.0, velocity = 0.0
##############################################
Number of objects = 7
ObjData =
x = 0.0, y = 1.392, z = 0.0, velocity = 0.0
x = -0.952, y = 1.29, z = 0.0, velocity = 0.0
x = -0.301, y = 1.575, z = 0.0, velocity = 0.0
x = 0.0, y = 2.278, z = 0.0, velocity = 0.0
x = -0.738, y = 2.244, z = 0.0, velocity = 0.0
x = 0.0, y = 4.557, z = 0.0, velocity = 0.0
x = -1.013, y = 5.305, z = 0.0, velocity = 0.0
##############################################
```

tcpserver.py

Note that the application tcpserver.py stops whenever 0 objects are detected. Users can modify the script to serve their own needs.

4.7. How to run 2-Chip cascade application (Applicable for only AM273X + 2xAWR2243 EVM)

Pre-built binaries (R5F only) are available in mmwave_sdkmmwave_mcuplus_sdk_<ver>/ti/utils/test/cascade/am273x. 2-chip cascade application supports capturing raw ADC data using DCA1000 EVM.

1. Power on the cascade EVM.
2. Connected to R5F Core and load "am273x_cascade_mss.xer5f" application and do not start the execution.
3. Copy AM273X_Capture.json file from mmwave_sdkmmwave_mcuplus_sdk_<ver>/tools/data_capture
4. Before starting execution of application, open a command prompt at C:/ti/mmwave_studio_03_00_00_14/mmWaveStudio/PostProc. and issue below commands.
    a. DCA1000EVM_CLI_Control.exe fpga AM273X_Capture.json
    b. DCA1000EVM_CLI_Control.exe start_record AM273X_Capture.json
    c. AM273X_Capture.json file configures to save the captured LVDS data at "C:\ti\data_capture\am273x__Raw_0"
5. Start execution of application.

4.8. Developing using SDK

4.8.1. Build Instructions

Follow the mmwave_mcuplus_sdk_release_notes instructions to install the mmwave_sdk in your development environment (windows or linux). All the tools needed for mmwave sdk build are installed as part of mmwave sdk installer.

4.8.2. Setting up build environment

4.8.2.1. Windows

1. Create command prompt at mmwave_mcuplus_sdk_<ver> install path>/scripts/windows folder. Under this folder you should see a setenv.bat file that has all the tools environment variables set automatically based on the installation folder. Review this file and change the few build variables shown below (if needed) and save the file. Please note that the rest of the environment variables should not be modified if the standard installation process was followed.

Build variables that can be modified (if needed) in setenv.bat

```
@REM ################################################################################
@REM # Build variables (to be modified based on build need)
@REM ################################################################################
@REM Select your device. Options (case sensitive) are: am273x, awr2944, awr2943
@REM Change this accordingly!!
set MMWAVE_SDK_DEVICE=am273x

@REM If download via CCS is needed, set below define to yes else no
@REM yes: Out file created can be loaded using CCS.
@REM Binary file created can be used to flash
@REM no: Out file created cannot be loaded using CCS.
@REM Binary file created can be used to flash
@REM (additional features: write-protect of TCMA, etc)
set DOWNLOAD_FROM_CCS=yes
```

> ⚠ If you see the following line in the setenv.bat file then most probably the wrong installer was used (Linux installation being compiled under Windows)
>
> *set MMWAVE_SDK_TOOLS_INSTALL_PATH=__MMWAVE_SDK_TOOLS_INSTALL_PATH__*
>
> In a proper installation the __MMWAVE_SDK_TOOLS_INSTALL_PATH__ would have been replaced with the actual installation folder path

2. Run **setenv.bat** as shown below.
3. **Run setenv.bat**

```
setenv.bat
```

This should not give errors and should print the message **"mmWave Build Environment Configured"**. The build environment is now setup.

4.8.2.2. Linux

1. Open a terminal and cd to mmwave_mcuplus_sdk_<ver> install path>/scripts/unix. Under this folder you should see a setenv.sh file that has all the tools environment variables set automatically based on the installation folder. Review this file and change the few build variables shown below (if needed) and save the file. Please note that the rest of the environment variables should not be modified if the standard installation process was followed.

   Build variables that can be modified (if needed) in setenv.sh

   ```
   ###############################################################################
   # Build variables (to be modified based on build need)
   ###############################################################################
   @REM Select your device. Options (case sensitive) are: am273x, awr2944, awr2943
   set MMWAVE_SDK_DEVICE=awr2944

   # If download via CCS is needed, set below define to yes else no
   # yes: Out file created can be loaded using CCS.
   # Binary file created can be used to flash
   # no: Out file created cannot be loaded using CCS.
   # Binary file created can be used to flash
   # (additional features: write-protect of TCMA, etc)
   export DOWNLOAD_FROM_CCS=yes
   ```

   > If you see the following line in the setenv.sh file then most probably the wrong installer was used (Windows installation being compiled under Linux)
   >
   > *export MMWAVE_SDK_TOOLS_INSTALL_PATH=__MMWAVE_SDK_TOOLS_INSTALL_PATH__*
   >
   > In a proper installation the __MMWAVE_SDK_TOOLS_INSTALL_PATH__ would have been replaced with the actual installation folder path

2. Assuming build is on a Linux 64bit machine, install modules that allows Linux 32bit binaries to run. This is needed for Image Creator binaries

   ```
   sudo dpkg --add-architecture i386
   ```

3. Install build-essential package for 'make'. Install mono-complete package as one of the Image Creator binaries (out2rprc.exe) is a windows executable that needs mono to run in Linux environment

   ```
   sudo apt-get install build-essential
   sudo apt-get --assume-yes install mono-complete
   ```

4. Run **setenv.sh** as shown below.

   Run setenv.sh

   ```
   source ./setenv.sh
   ```

   This should not give errors and should print the message **"mmWave Build Environment Configured"**. The build environment is now setup.

4.8.3. Building demo

To clean build a demo, first make sure that the environment is setup as detailed in earlier section. Then run the following commands. On successful execution of the commands, the output is <demo>.xe* which can be used to load the image via CCS and <demo>.bin which can be used as the binary in the steps mentioned in section "How to flash an image onto mmWave EVM".

4.8.3.1. Building demo in Windows

Building demo in windows

```
REM Fill <device type> with appropriate device that supports demo in a particular release
cd %MMWAVE_SDK_INSTALL_PATH%/ti/demo/<device type>/mmw

REM Clean and build
gmake clean
gmake all
```

```
REM Incremental build
gmake all

REM For example to build the mmw demo
cd %MMWAVE_SDK_INSTALL_PATH%/ti/demo/%MMWAVE_SDK_DEVICE%/mmw
gmake clean
gmake all
REM This will create <mmwave_sdk_device>_mmw_demo_mss.xer5f, <mmwave_sdk_device>_mmw_demo_dss.xe66 &
<mmwave_sdk_device>_mmw_demo.appimage binaries
REM under %MMWAVE_SDK_INSTALL_PATH%/ti/demo/<mmwave_sdk_device>/mmw folder
```

4.8.3.2. Building demo in Linux

Building demo in linux

```
# Fill <device type> with appropriate device that supports demo in a particular release
cd ${MMWAVE_SDK_INSTALL_PATH}/ti/demo/<device type>/mmw

# Clean and build
make clean
make all

# Incremental build
make all

# For example to build the mmw demo
cd ${MMWAVE_SDK_INSTALL_PATH}/ti/demo/${MMWAVE_SDK_DEVICE}/mmw
make clean
make all
# This will create <mmwave_sdk_device>_mmw_demo_mss<Proc_Chain>.xer5f,
<mmwave_sdk_device>_mmw_demo_dss<Proc_Chain>.xe66 & <mmwave_sdk_device>_mmw_demo<Proc_Chain>.appimage
binaries
# under ${MMWAVE_SDK_INSTALL_PATH}/ti/demo/${MMWAVE_SDK_DEVICE}/mmw folder
```

⚠ Each demo has dependency on various drivers and control components. The libraries for those components need to be available in their respective lib folders for the demo to build successfully.

4.8.4. Advanced build

The mmwave sdk package includes all the necessary libraries and hence there should be no need to rebuild the driver, algorithms or control component libraries. In case a modification has been made to any of these modules then the following section details how to build these components.

4.8.4.1. Building datapath/control/utils components

To clean build control, datapath or utils components and unit tests, first make sure that the environment is setup as detailed in earlier section. Then run the following commands

Building component in windows

```
cd %MMWAVE_SDK_INSTALL_PATH%/ti/<component_path_under_ti>
gmake clean
gmake all


REM The command will create the following file
REM     lib<component>_<device_type>.aer5f library under ti/<component_path_under_ti>/lib folder
REM If the module has unit test, it will also create
REM     <device_type>_<component>_mss.xer5f unit test binary under ti/<component_path_under_ti>/test
/<device_type> folder
REM If the device has a DSP and the driver supports DSP then the command will also create
REM     lib<component>_<device_type>.ae66 library for DSS under ti/<component_path_under_ti>/lib folder
REM If the module has unit test, it will also create
REM     <device_type>_<component>_dss.xe66 unit test binary for DSS under ti/<component_path_under_ti>/test
/<device_type> folder
```

```
REM Above paths are relative to %MMWAVE_SDK_INSTALL_PATH%/

REM For example to build the dpm lib and unit test
cd %MMWAVE_SDK_INSTALL_PATH%/ti/control/dpm
gmake clean
gmake all

REM For example to build the mmwavelink lib (it does not have a unit test)
cd %MMWAVE_SDK_INSTALL_PATH%/ti/control/mmwavelink
gmake clean
gmake all

REM For example to build the aoaproc dpu lib and unit test
cd %MMWAVE_SDK_INSTALL_PATH%/ti/datapath/dpc/dpu/aoaproc
gmake clean
gmake all

REM Additional build options for each component can be found by invoking make help
gmake help
```

Building component in linux

```
cd ${MMWAVE_SDK_INSTALL_PATH}/ti/<component_path_under_ti>
make clean
make all

# The command will create the following file
#     lib<component>_<device_type>.aer5f library under ti/<component_path_under_ti>/lib folder
# If the module has unit test, it will also create
#     <device_type>_<component>_mss.xer5f unit test binary under ti/<component_path_under_ti>/test
/<device_type> folder
# If the device has a DSP and the driver supports DSP then the command will also create
#     lib<component>_<device_type>.ae66 library for DSS under ti/<component_path_under_ti>/lib folder
# If the module has unit test, it will also create
#     <device_type>_<component>_dss.xe66 unit test binary for DSS under ti/<component_path_under_ti>/test
/<device_type> folder
# Above paths are relative to ${MMWAVE_SDK_INSTALL_PATH}/

# For example to build the dpm lib and unit tests
cd ${MMWAVE_SDK_INSTALL_PATH}/ti/control/dpm
make clean
make all

# For example to build the mmwavelink lib
cd ${MMWAVE_SDK_INSTALL_PATH}/ti/control/mmwavelink
make clean
make all

# For example to build the aoaproc dpu lib and unit test
cd ${MMWAVE_SDK_INSTALL_PATH}/ti/datapath/dpc/dpu/aoaproc
make clean
make all

# Additional build options for each component can be found by invoking make help
gmake help
```

example output of make help for dpm

```
****************************************************************************************
* Makefile Targets for the DPM
clean                -> Clean out all the objects
lib                  -> Build the Driver only
libClean             -> Clean the Driver Library only
test                 -> Build the applicable unit tests
testClean            -> Clean the unit tests
****************************************************************************************
```

example output of make help for demo

```
****************************************************************************************
* Makefile Targets for the DEMO
all             -> Builds the mmw Demo (both MSS and DSS binaries) and appimage
clean           -> Cleans the mmw Demo (both MSS and DSS binaries) and appimage
mssDemo         -> Builds the MSS binary for the mmw Demo
mssDemoClean    -> Cleans the MSS binary for mmw Demo
dssDemo         -> Builds the DSS binary for the mmw Demo
dssDemoClean    -> Cleans the DSS binary for mmw Demo
bin             -> Builds the appimage for the demo
binClean        -> Cleans the appimage for the demo
mmwDemo         -> Builds the mmw Demo (both MSS and DSS binaries) but not the appimage
mmwDemoClean    -> Builds the mmw Demo (both MSS and DSS binaries)
****************************************************************************************
```

⚠ Please note that not all components are supported for all devices and not all components have unit tests. List of supported components for each device is listed in the Release Notes.

4.8.4.2. "Error on warning" compiler and linker setting

By default, the SDK build uses "–emit_warnings_as_errors" option to help users identify certain common mistakes in code that are flagged as warning but could lead to unexpected results. If user desires to disable this feature, then please set the flag MMWAVE_DISABLE_WARNINGS_AS_ERRORS to 1 in the above mentioned setenv.bat or setenv.sh and invoke that file again to update the build environment. User can also compile with the "MMWAVE_DISABLE_WARNINGS_AS_ERRORS=1" flag (for example, *gmake all MMWAVE_DISABLE_WARNINGS_AS_ERRORS=1*).

5. MMWAVE SDK deep dive

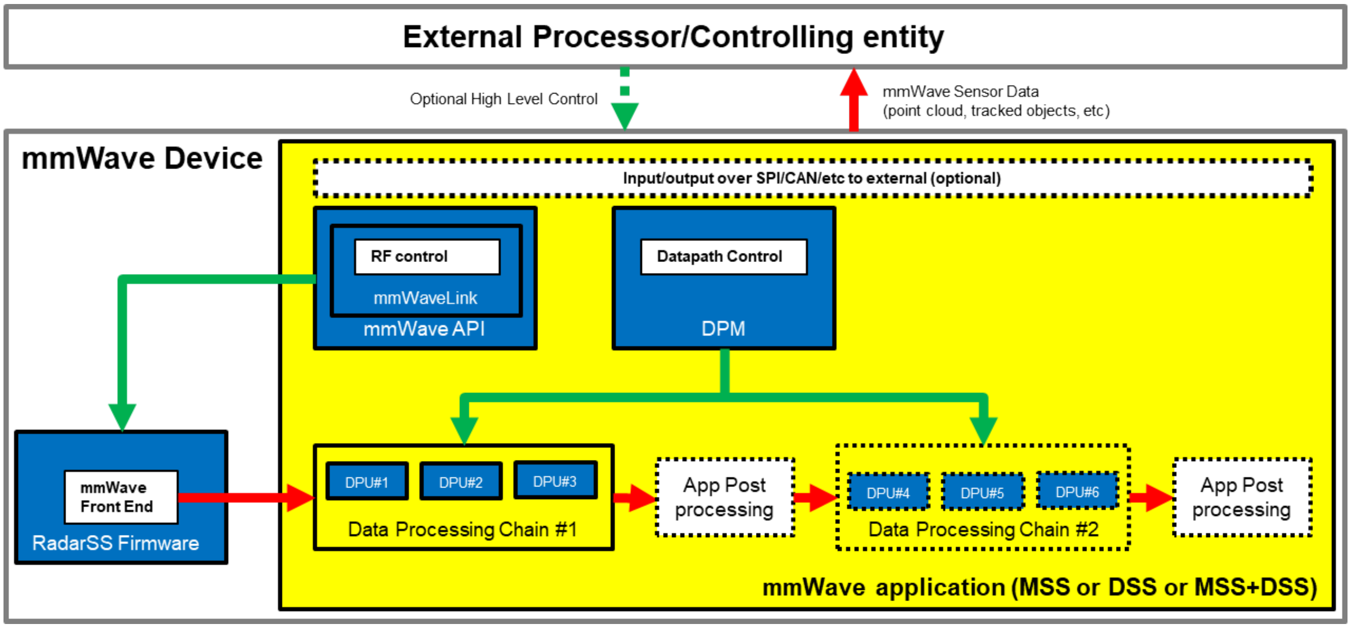
5.1. System Deployment

A typical mmWave application would perform these operations:

- Control and monitoring of RF front-end through mmaveLink
- Transport of external communications through standard peripherals
- Some radar data processing using DSP

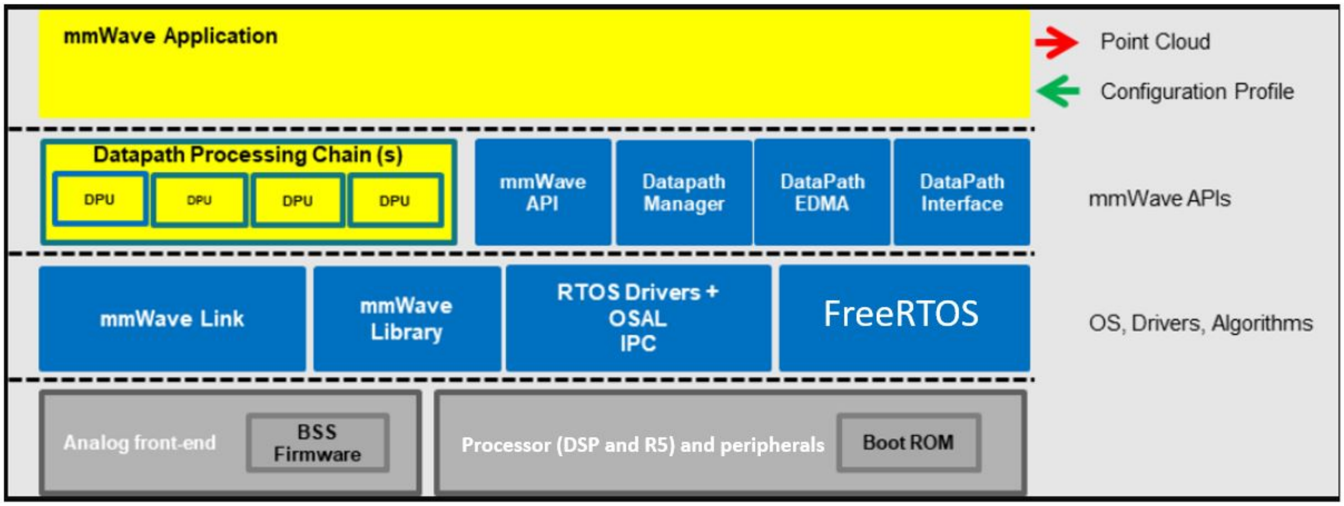
Typical customer deployment for mmWave sensor is shown in the figure below:

a. The appimage (MSS + DSS code) is downloaded from the serial flash memory **attached** to the mmWave device (via QSPI)
b. **Optional** high level control from remote entity
c. Sends **low speed data** output (objects detected) to remote entity



5.1.1.1.1. Figure 4: Autonomous mmWave sensor (Standalone mode)

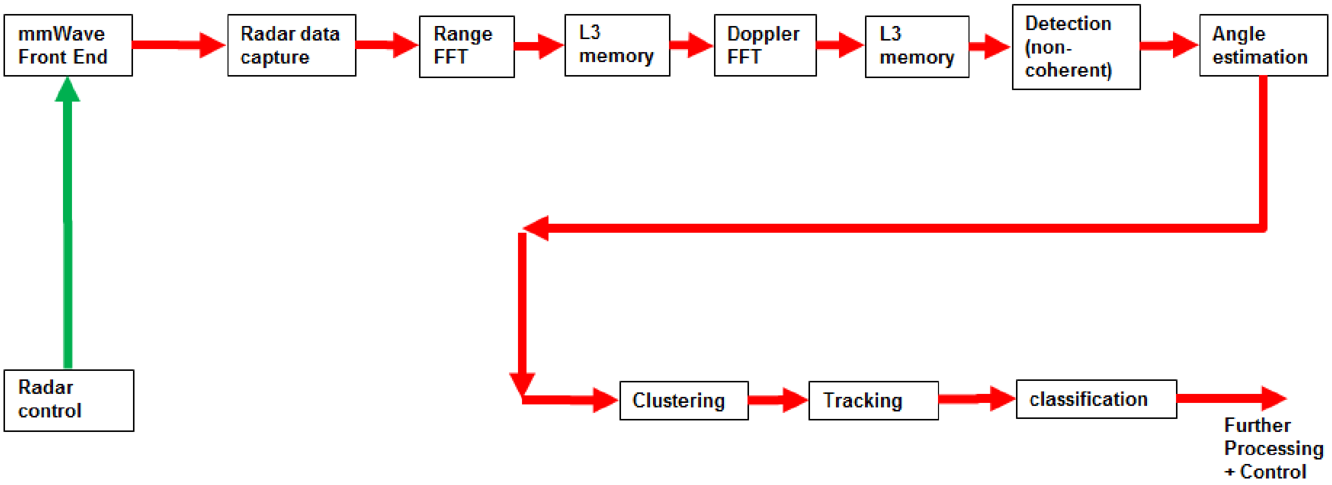
The above deployment can be realized using the mmWave SDK and it components in a layered structure as shown below:



5.1.1.1.2. Figure 5: SDK Layered block diagram

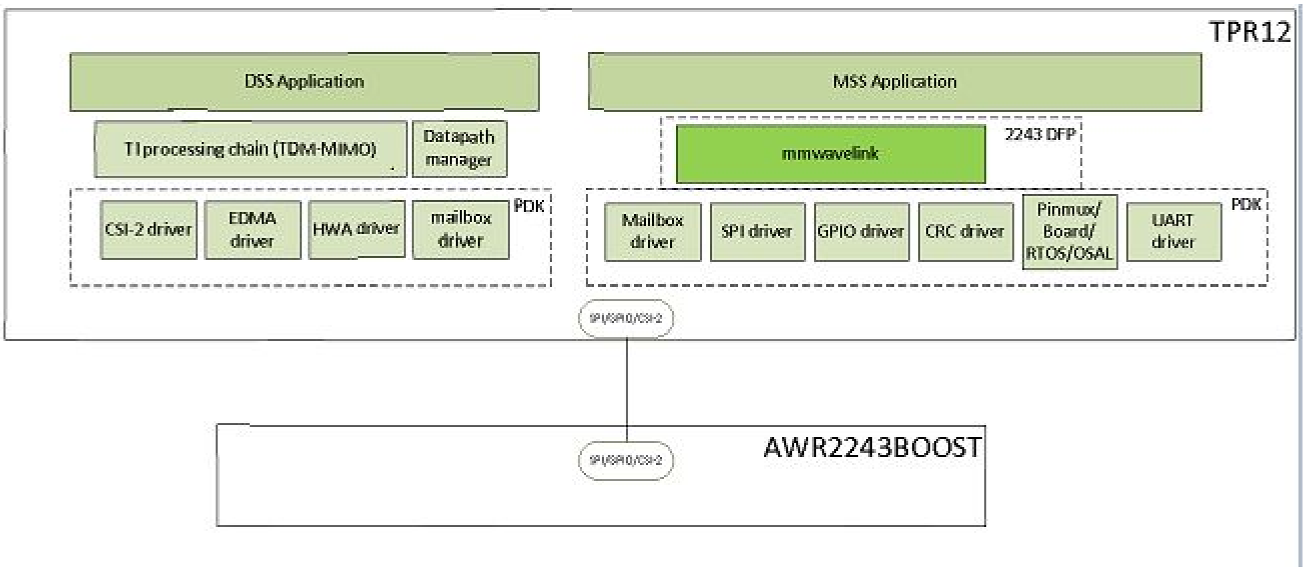
5.2. Typical mmWave Radar Processing Chain

Following figure shows a typical mmWave Radar processing chain that accepts ADC data as input from mmWave Front End and then performs Range and Doppler FFT followed by non-coherent detection using CFAR. Finally angle is estimated using 3D FFT and the detected points represent the point cloud data. The point cloud data can then be post processed using higher layer algorithms such as Clustering, Tracking, Classification to represent real world targets.

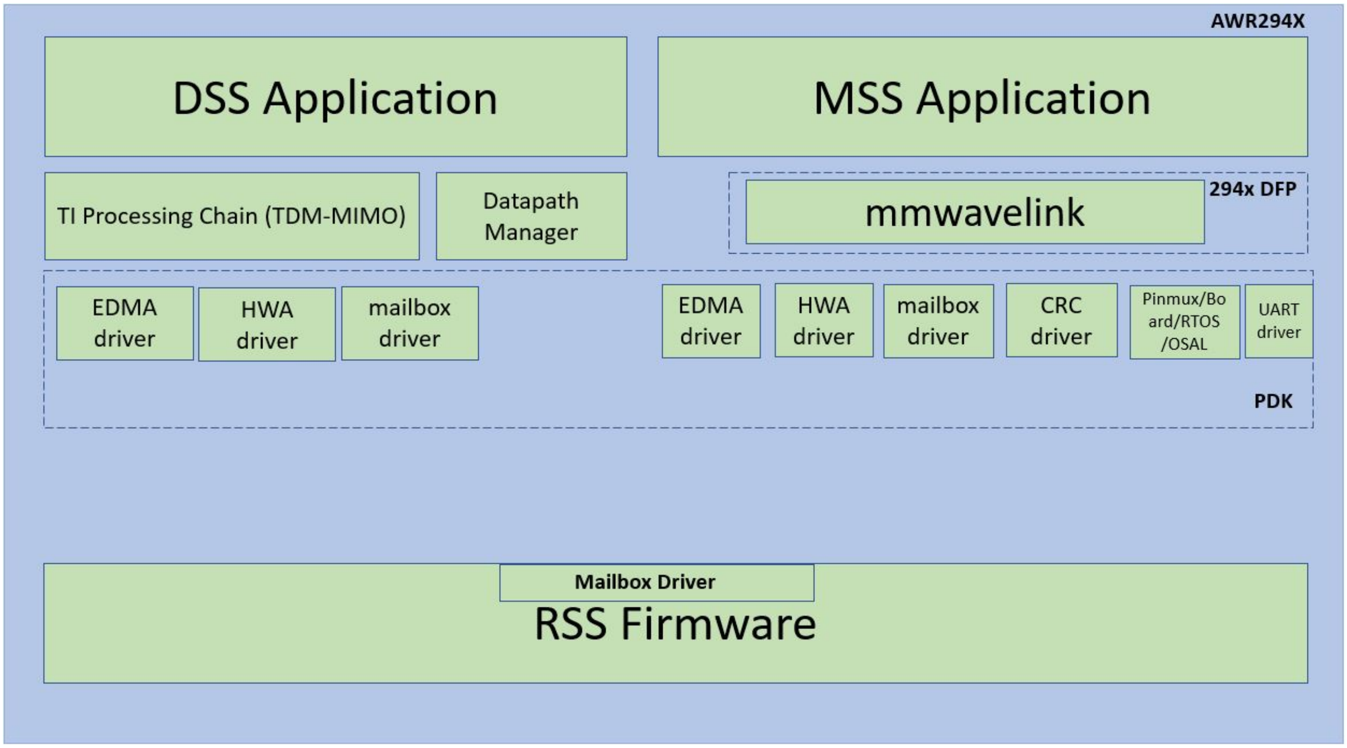


5.2.1.1.1. Figure 6: Typical mmWave radar processing chain

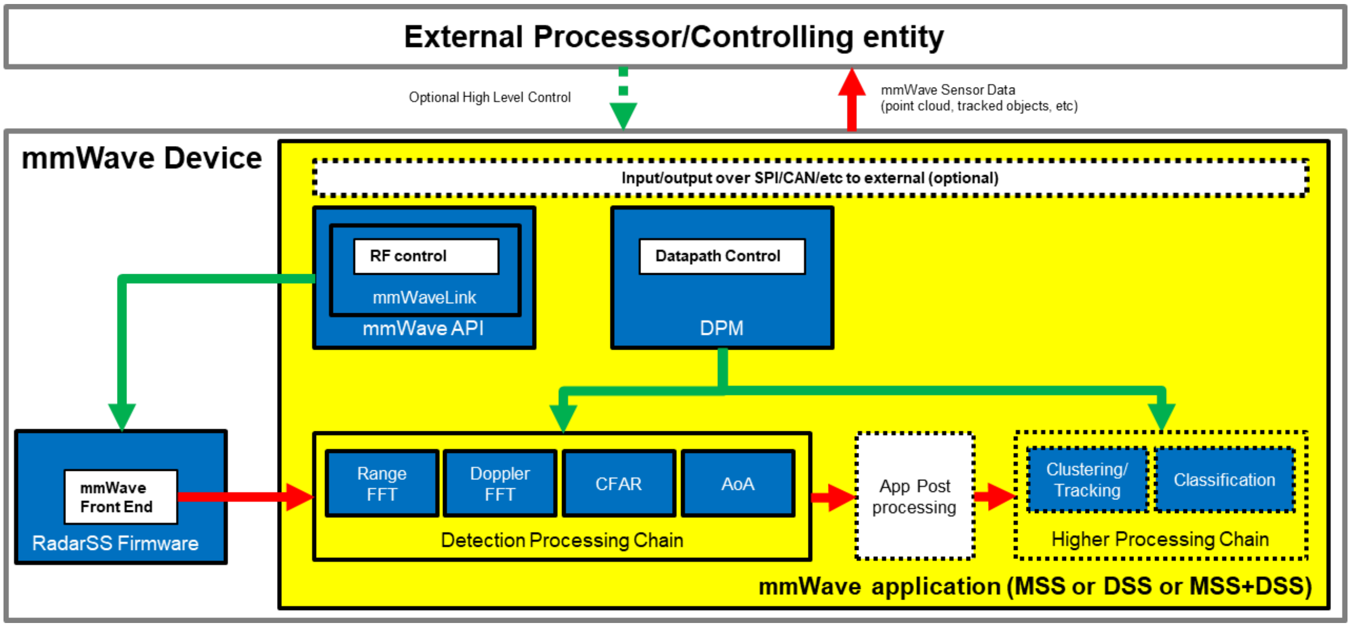
Using mmWave SDK the above chain could be realized as shown in the following figure for devices with HWA as processing nodes. In the following figure, green arrow shows the control path and red arrow shows the data path. Blue blocks are mmWave SDK components and yellow blocks are custom application code. The hierarchy of software flow/calls is shown with embedding boxes. Depending on the complexity of the higher algorithms (such as clustering, tracking, etc) and their memory/mips consumption, they can either be partially realized inside the mmWave device or would run entirely on the external processor.



AM273X+AWR2243 OOB Demo



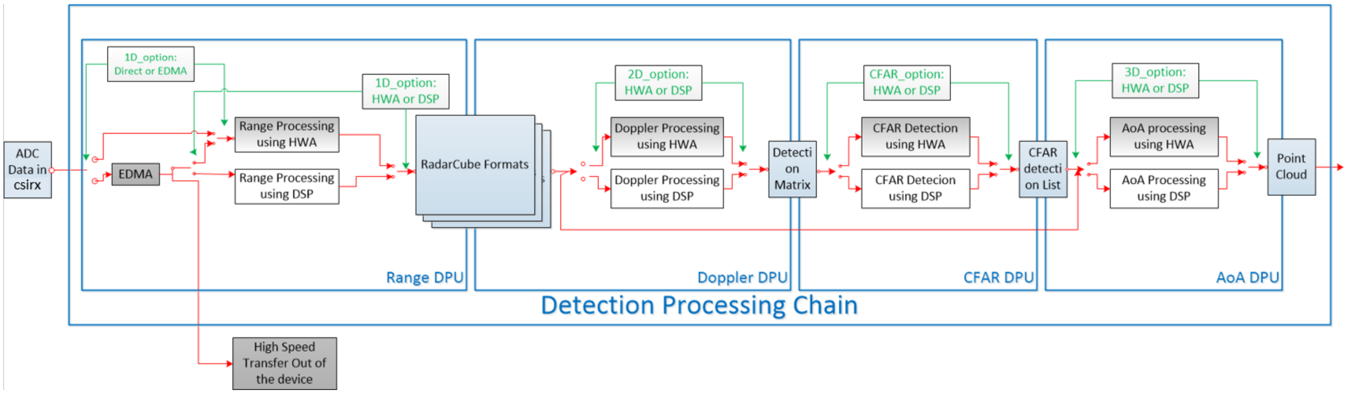
AWR294X OOB Demo


External Processor/Controlling entity
Optional High Level Control
mmWave Sensor Data
(point cloud, tracked objects, etc)
mmWave Device
Input/output over SPI/CAN/etc to external (optional)
RF control
mmWaveLink
mmWave API
Datapath Control
DPM
mmWave
Front End
RadarSS Firmware
Range FFT
Doppler FFT
CFAR
AoA
Detection Processing Chain
App Post processing
Clustering/ Tracking
Classification
Higher Processing Chain
mmWave application (MSS or DSS or MSS+DSS)

5.2.1.1.2. Figure 7: Typical mmWave radar processing chain using mmWave SDK components

Each of the mmWave device offers different processing nodes to realize the mmwave processing. AM273X has HWA+DSP(C66x). For devices with multiple processing nodes, the mmWave detection processing chain can exploit them as needed for performance and scalable reasons. Shown below is an example of detection processing chain that uses various data processing units (DPUs) to perform the typical mmwave processing upto the point cloud. The mmwave data representation in mmWave device memory forms an interface layer between the various DPUs. Each DPU can be realized independently using HWA or DSP processing node - the choice is either driven by usecase or availability of that processing node on a given mmWave device. The various mmWave SDK components shown below are described in the section "mmWave SDK - TI components" below.

5.2.1.1.3.



5.2.1.1.4. Figure 8: Scalable data processing chain using mmWave SDK

Please refer to the code and documentation inside the mmwave_mcuplus_sdk_<ver>\ti\demo\<platform>\mmw folder for more details and example code on how this chain is realized using mmWave SDK components.

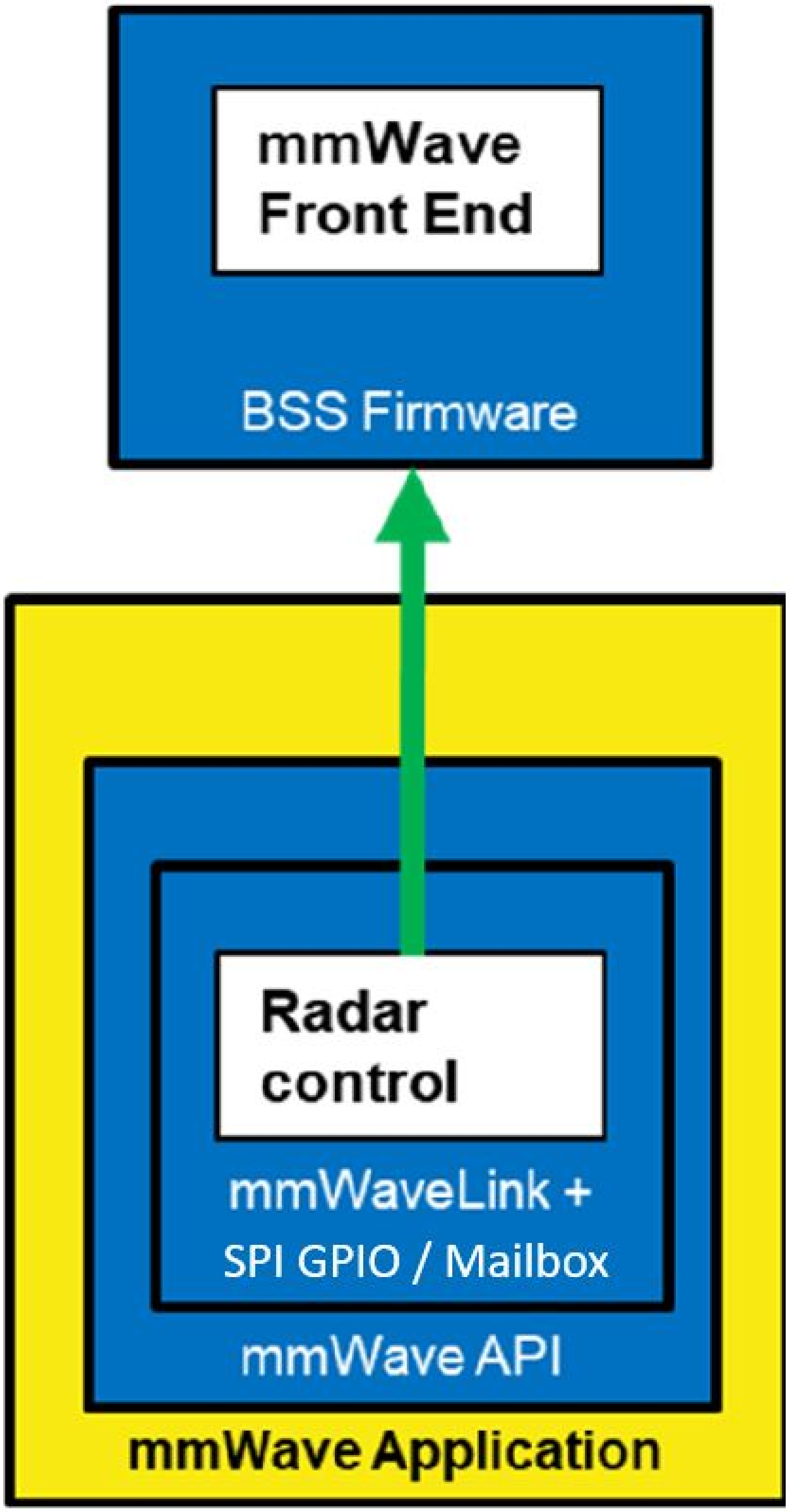
> What must be noted is that in SDK 4.1.0 datapath processing, is done using HWA running on MSS or DSS, not DSP alone (without using the HWA). Also, the ADC data over the csirx can either be directly put into the HWA memory (as is supported by the current demo), or can be first placed into an intermediate memory and then routed to the HWA using the EDMA or transferred out of the device (not supported by the current demo).

5.3. Typical Programming Sequence

The above processing chain can be split into two distinct blocks: RF control path and data path.

5.3.1. RF Control Path

The control path in the above processing chain is depicted by the following blocks.

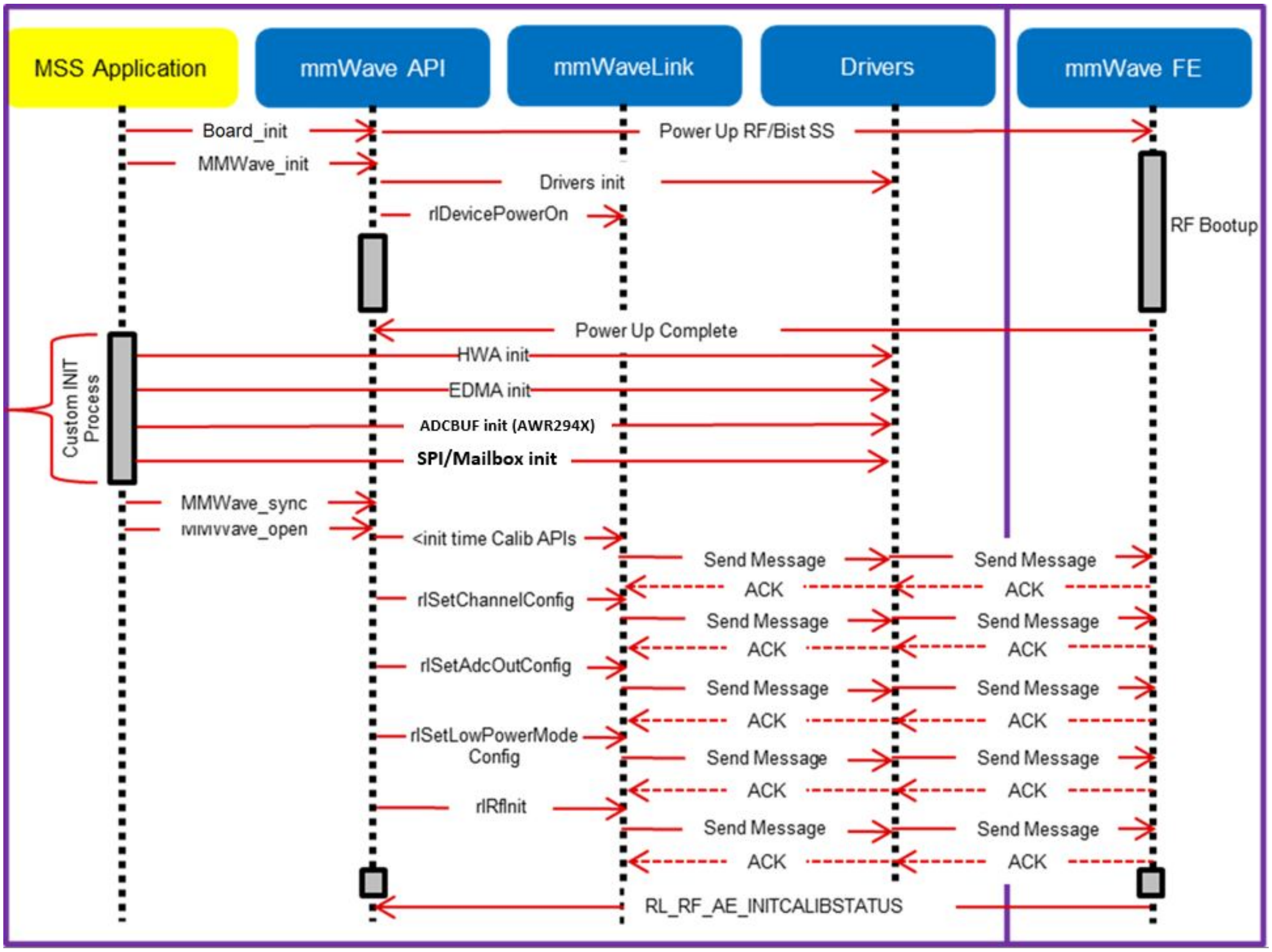


5.3.1.1.1. Figure 9: Typical mmWave radar control flow

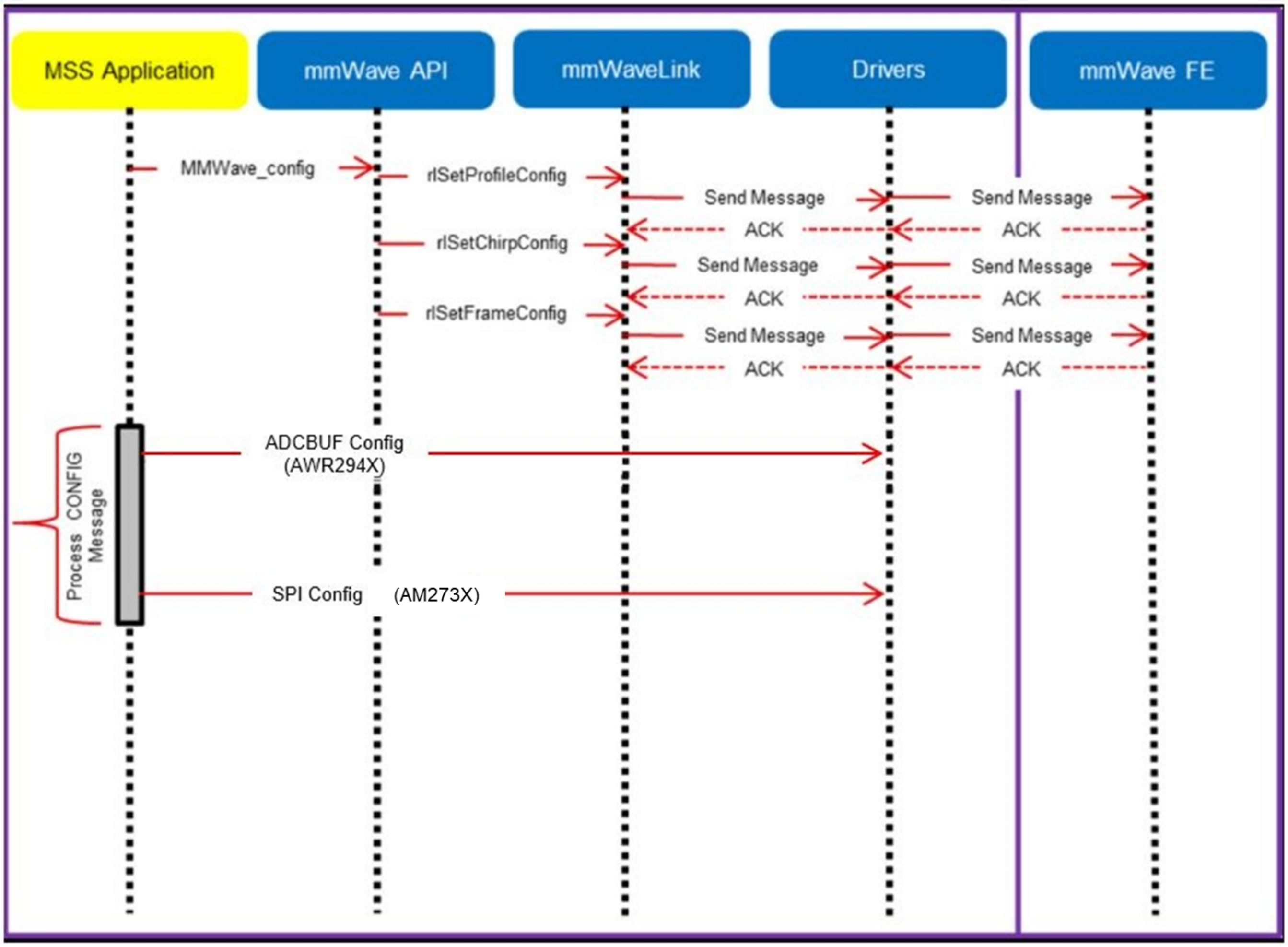
Following set of figures shows how an application programming sequence would look like for setting up the typical control path - init, config, start. This is a high level diagram simplified to highlight the main software APIs and may not show all the processing elements and call flow. For an example implementation of this call flow, please refer to the code and documentation inside the mmwave_mcuplus_sdk_<ver>\ti\demo\<platform>\mmw folder.

5.3.1.2. Single RF Control (MSSRADARSS or DSSRADARSS)

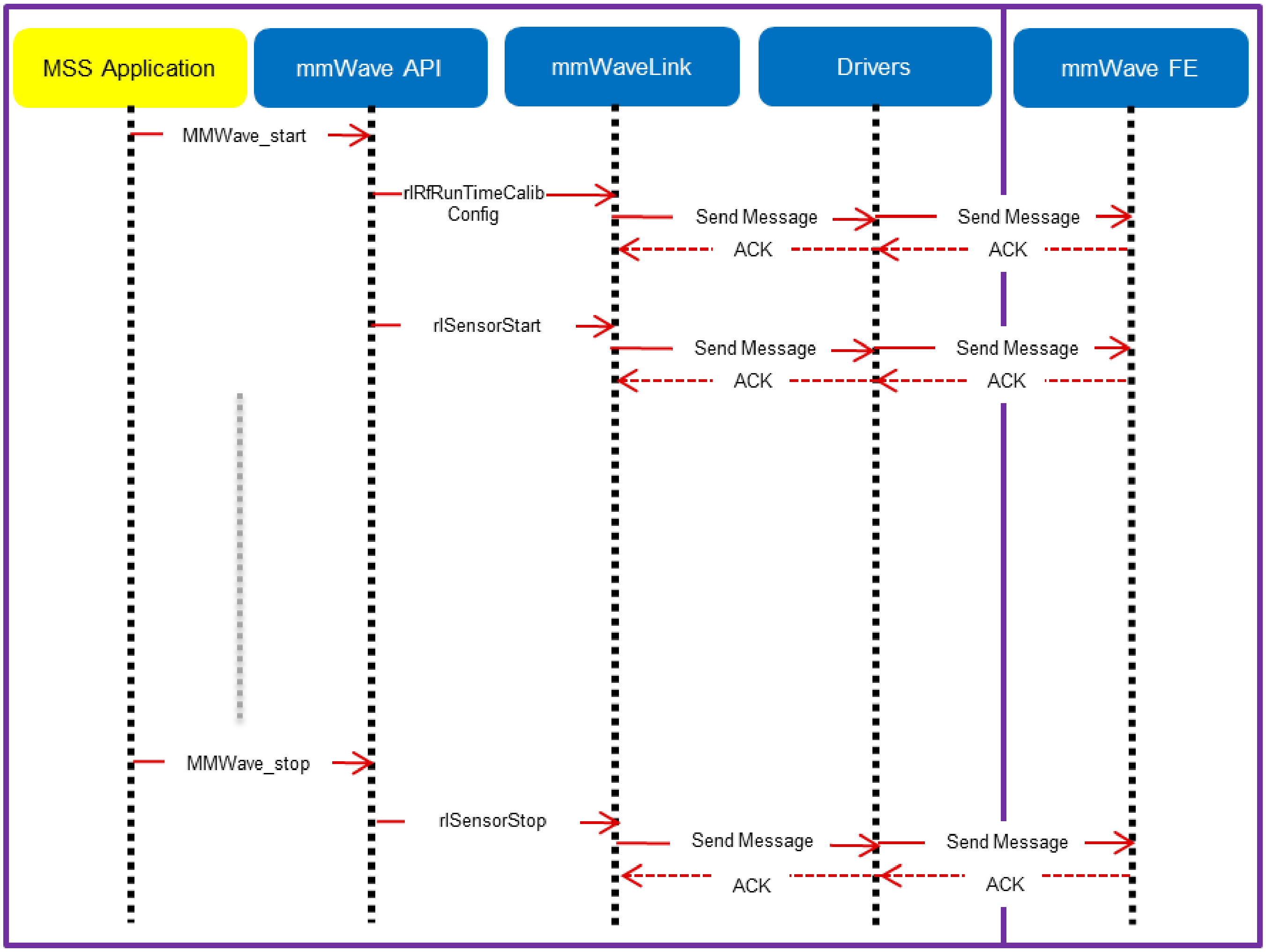
In this scenario, the RF control path runs on either Master subsystem (Cortex-R5F) or DSP subsytem (C66x) and the application can simply call the mmwave APIs in the SDK in isolation mode to realize most of the functionality.



5.3.1.2.1. Figure 10: mmWave Isolation mode: Detailed Control Flow (Init sequence)



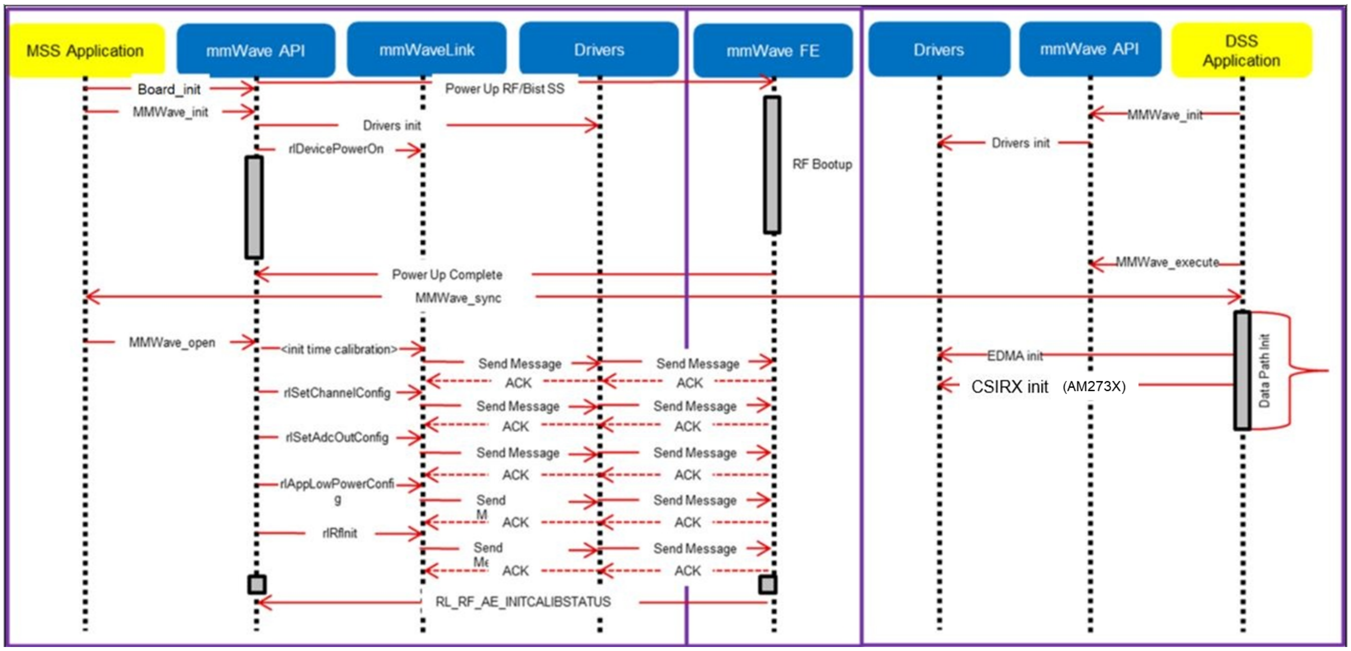
5.3.1.2.2. Figure 11: mmWave Isolation mode: Detailed Control Flow (Config sequence)



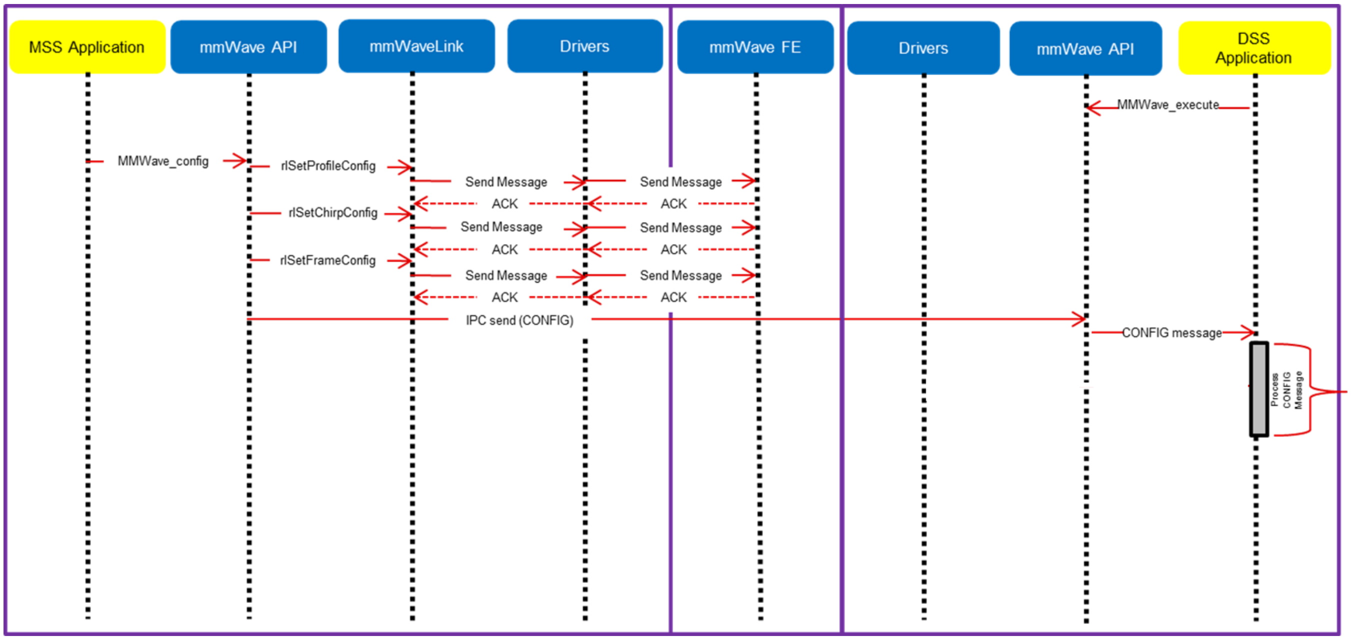
5.3.1.2.3. Figure 12: mmWave Isolation mode: Detailed Control Flow (start sequence)

5.3.1.3. Co-operative RF control ((MSS+DSS)<->RADARSS)

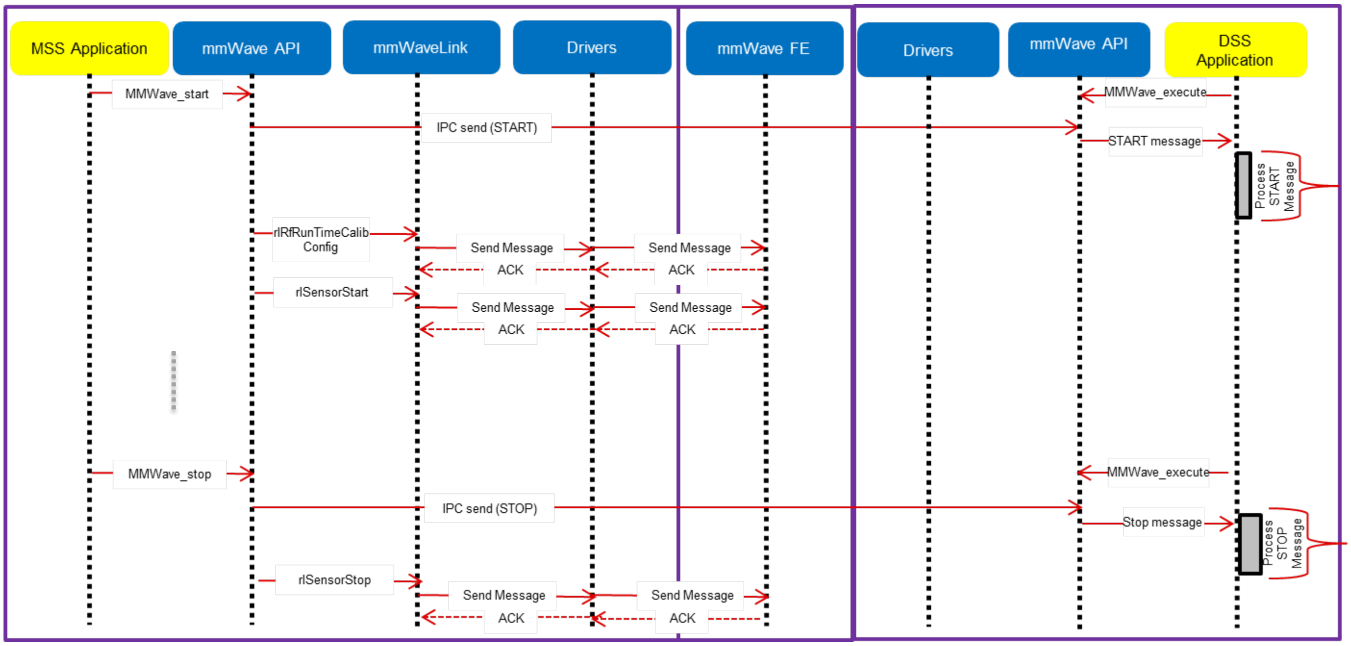
In this scenario the control path can runs in "co-operative" mode where RF control APIs can be interchangeably called by either domains (but the sequence of API needs to be maintained). One such deployment could have the RF init and config initiated by the MSS and the start is initiated by the DSS after the data path configuration is complete. In the figures below, control path runs on MSS entirely and MSS is responsible for properly configuring the RADARSS (RF) and DSS (data processing). The mmWave unit tests provide a sample implementation of this co-operative mode.



5.3.1.3.1. Figure 13: mmWave Co-operative Mode: Detailed Control Flow (Init sequence)



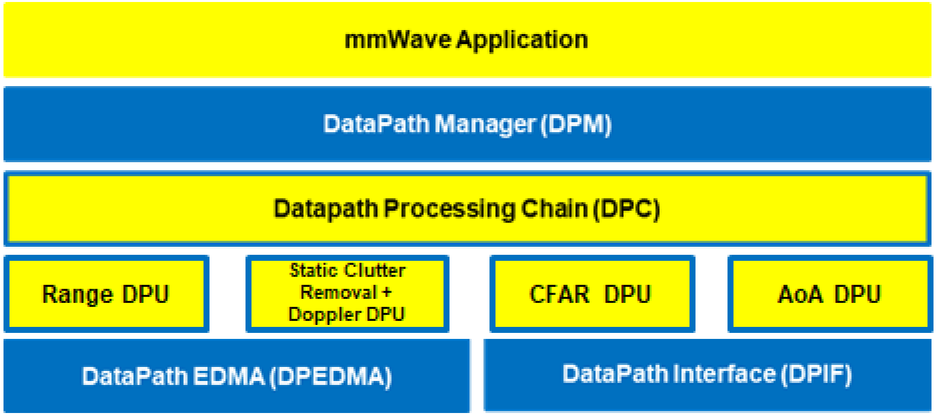
5.3.1.3.2. Figure 14: mmWave Co-operative Mode: Detailed Control Flow (Config sequence)



5.3.1.3.3. Figure 15: mmWave Co-operative Mode: Detailed Control Flow (Start sequence)

5.3.2. Data Path

The mmwave detection processing can be split into following layers of application code, control/management layer to manipulate the data processing elements, processing chain that ties up individual modules to create a data flow and the low level data processing modules and interfaces.


mmWave Application
DataPath Manager (DPM)
Datapath Processing Chain (DPC)
Range DPU
Static Clutter Removal + Doppler DPU
CFAR DPU
AoA DPU
DataPath EDMA (DPEDMA)
DataPath Interface (DPIF)

5.3.2.1.1. Figure 16: Typical mmWave Detection Processing Layers

mmWave devices present a few options on how the data processing layers can be realized using the various control/processing nodes within the device. To allow ease of programming across these deployment types, data path manager (DPM) presents a simplified API structure to the application while hiding the complexity of inter task and inter processor communications. As can be seen from the following figures, application would just need to call the various DPM APIs to control the processing chain (seen as function calls in 'blue' in the ladder diagrams below) and re-act to the outcome of these APIs in the report callback. Data processing chains (DPCs) also present a standardized API structure to the application via DPM and encapsulate the realization of the data flow using data processing units (DPUs) within while presenting simple IOCTL based interface to configure and trigger the data flow. Based on the usecase and the mmWave device hardware capabilities, application can choose from one of the following deployments:

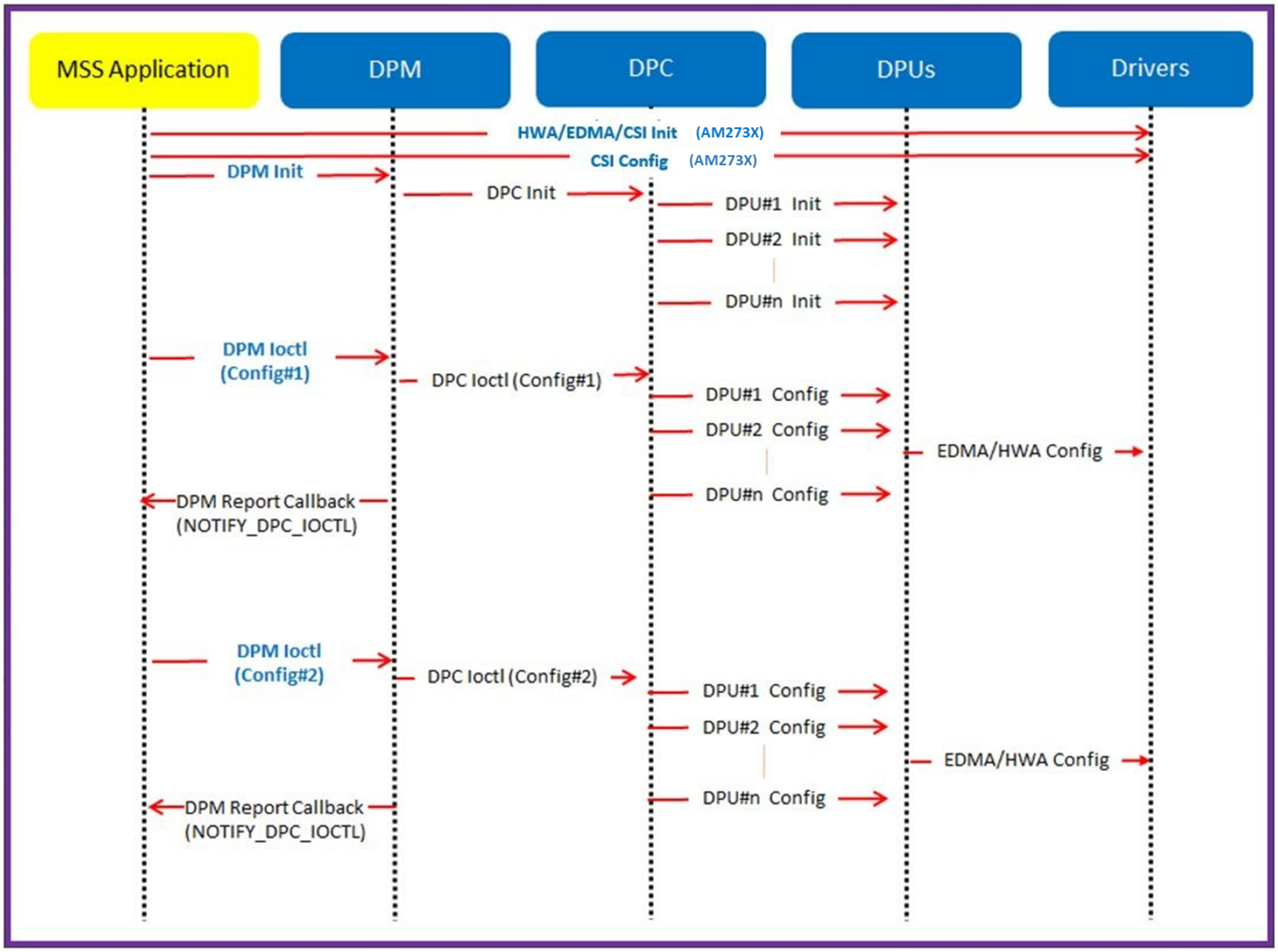
- DPC runs on the same core as control core and the application can control the DPC via DPM in local mode. (See local domain config and processing figures below)
- DPC runs on another core which is different from the controlling core and the application can control the DPC via DPM in remote mode. (See remote domain config and processing figures below)
- DPC is split between two cores and the application can control the DPC via DPM in distributed mode. (See distributed domain config and processing figures below)

The following ladder diagrams show the flow for init, two different forms of config (one initiated on local core and other on remote core), start trigger, chirps/frame events and stop trigger. The choice of MSS and DSS responsibilities are shown as one of the possible examples - their roles can be interchanged as per application needs. These ladder diagrams don't show the corresponding MMWAVE/RF control calls to show independence between RF control and datapath control. Having said that, typical application would follow the following flow for these two form of controls:

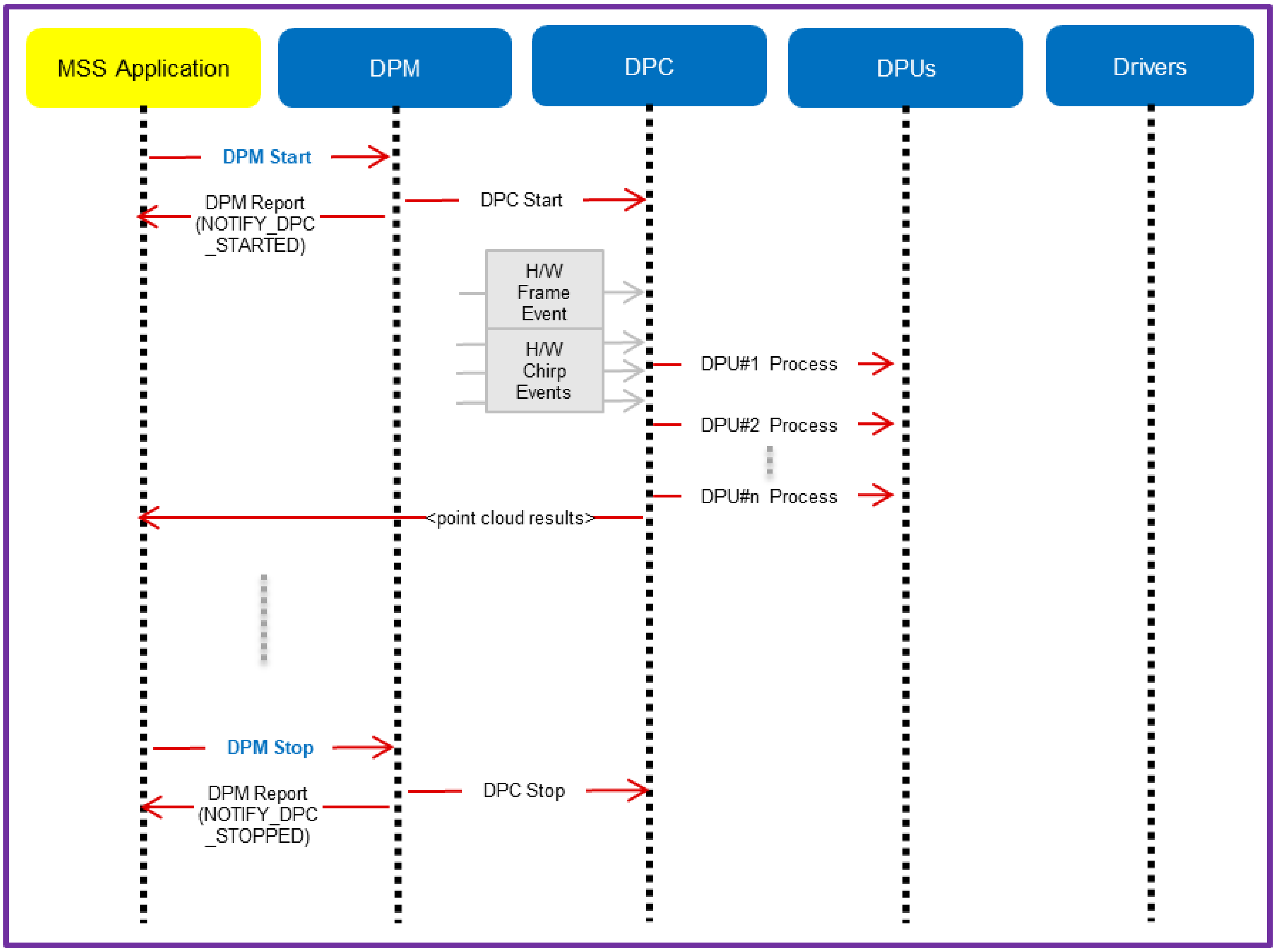
- mmWave init and DPM init (order doesn't matter)
- mmWave config and DPM IOCTL for DPC config (order doesn't matter)
- DPM start and then mmWave start (note this is recommended as DPC should be in started state before the real time frame/chirp H/W events occur due to mmWave start)
- mmWave stop and then DPM stop (note this is recommended as DPC should be stopped after the real time frame/chirp H/W events stop due to mmWave stop)

5.3.2.2. Data processing flow with local domain control

In this deployment, the core (MSS or DSS) that runs the actual data processing chain (DPC) also controls it. Application calls DPM APIs for init, data processing IOCTL for configuration, start and stop. DPM reports back status from DPC using the application registered report callback function. Application provides an execution context (task) for the DPM/DPC to run. DPC provides back the processing results (point cloud, tracked objects, etc) to the application in this execution context.



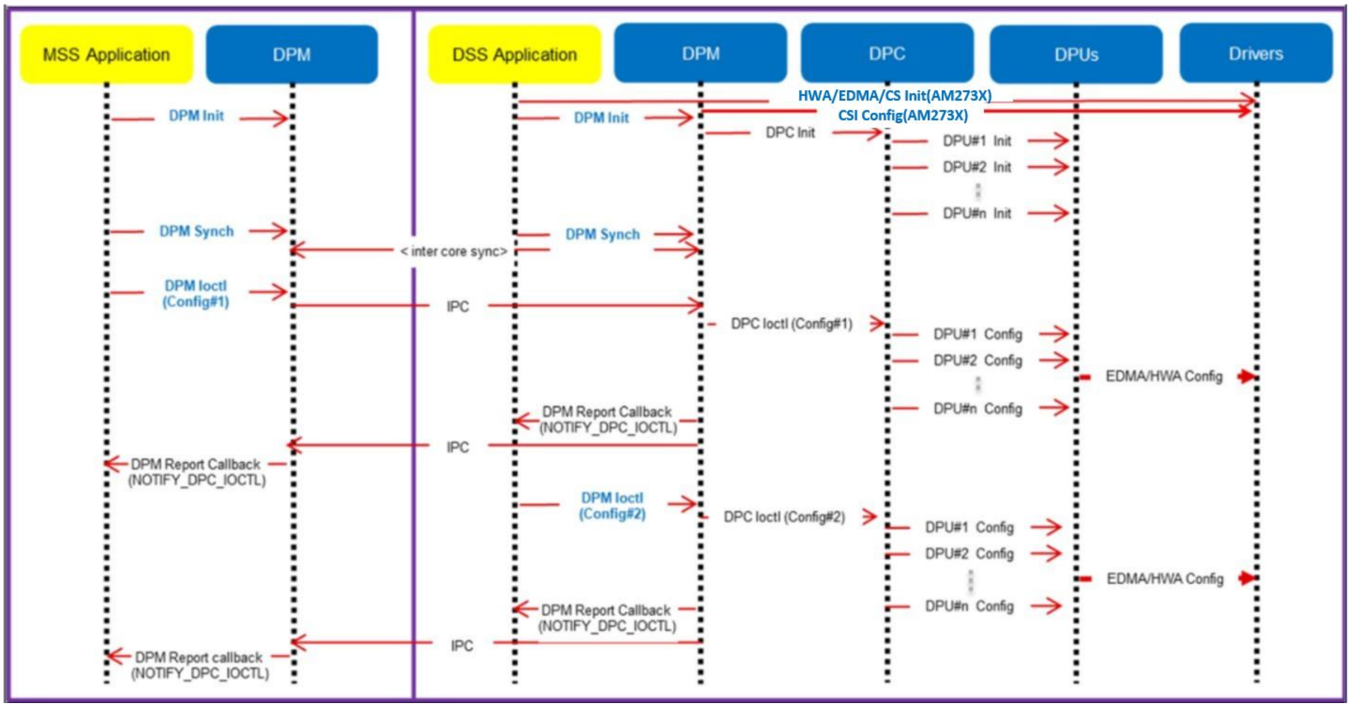
5.3.2.2.1. Figure 17: Data processing flow with local domain control (init/config)



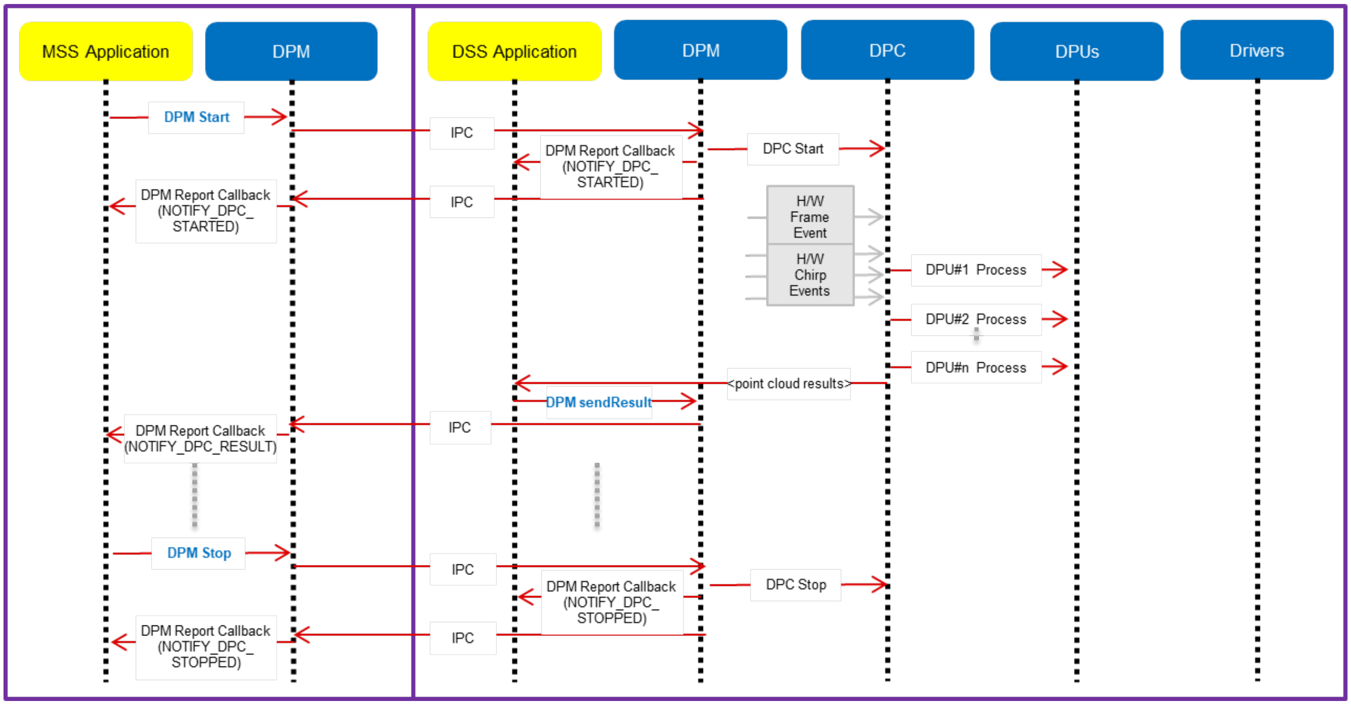
5.3.2.2.2. Figure 18: Data processing flow with local domain control (start/chirp/frame/stop)

5.3.2.3. Data processing flow with remote domain control

In this deployment, the data processing chain runs on a chosen data core while the control for it exists on the other core. Application code on control core and data core calls DPM APIs for init and sync'ing with each other. The control core calls data processing IOCTL for configuration, start and stop APIs. The H/W events are received on the data core. DPM reports back status from DPC using the application registered report callback function on both control and data cores. DPC provides back the processing results (point cloud, tracked objects, etc) to the data core application code which can send the result to the control core using DPM send result API.



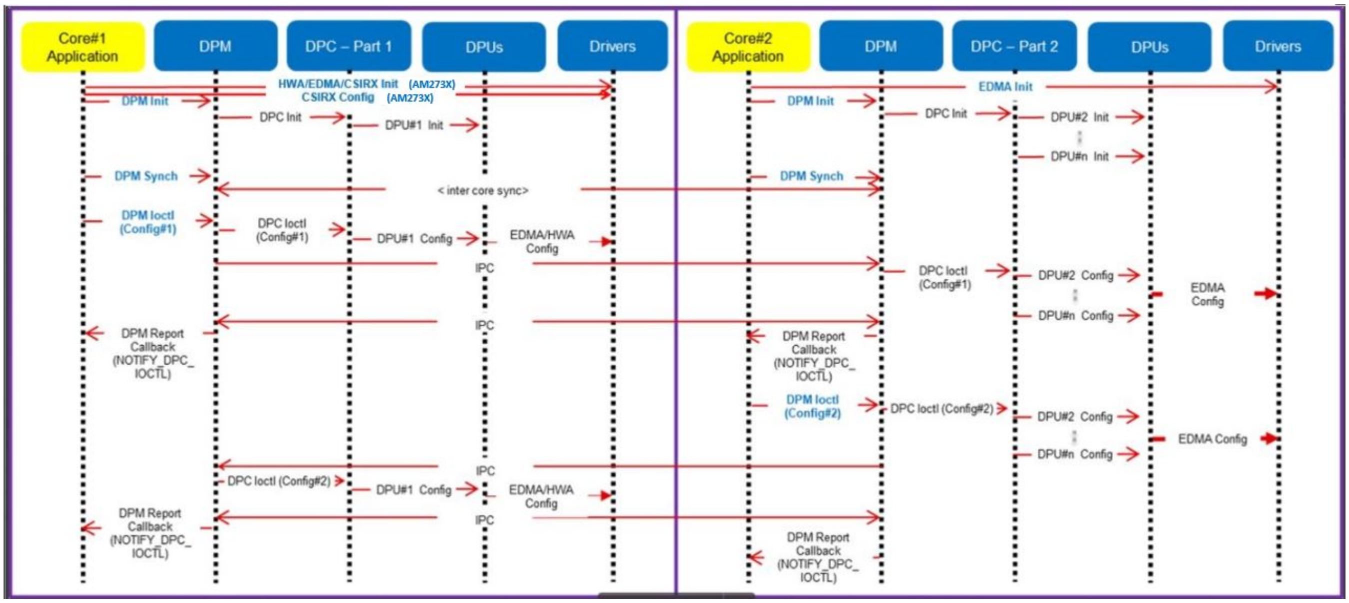
5.3.2.3.1. Figure 19: Data processing flow with remote domain control (init/config)



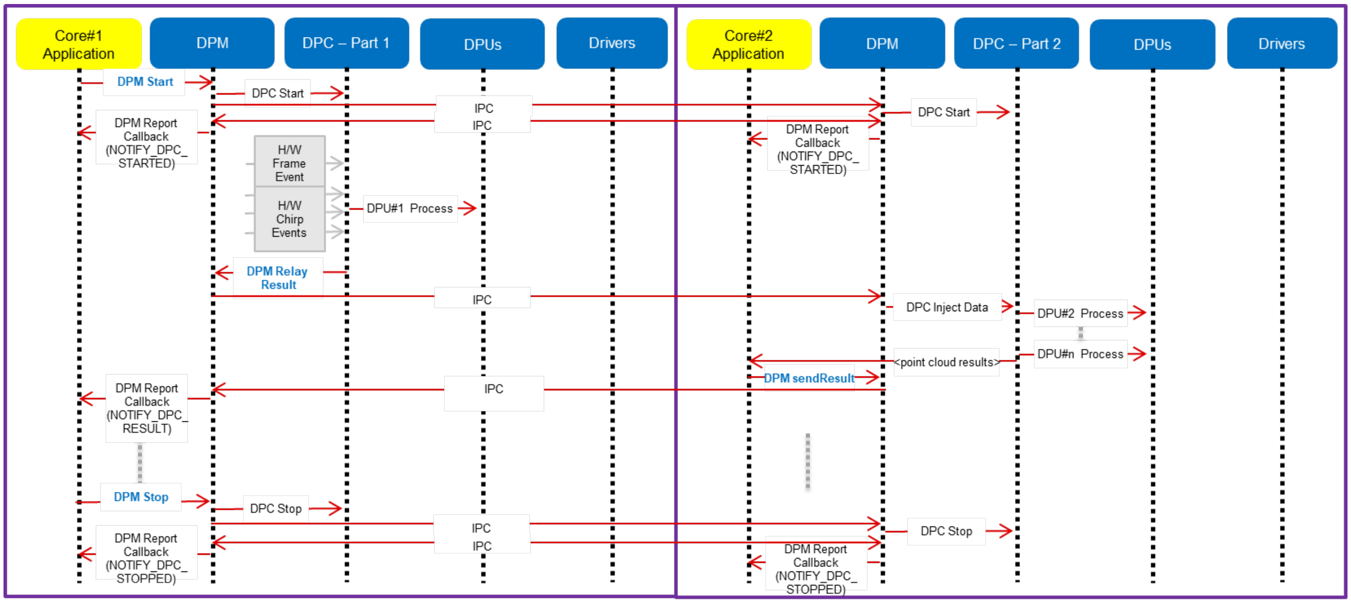
5.3.2.3.2. Figure 20: Data processing flow with remote domain control (start/chirp/frame/stop)

5.3.2.4. Distributed Data processing flow and control

In this deployment, the data processing chain is split across cores along with the control. Application code on both cores call DPM APIs for init and sync'ing with each other. Either core can call data processing IOCTL for configuration, start and stop APIs. DPM reports back status from DPC using the application registered report callback function on both cores. Partial results from the DPC running on one core can be passed onto the DPC running on other core using the DPM relay result API. DPC can provide back the final processing results (point cloud, tracked objects, etc) to the same core's application code which can then send the result to the application running on other core using DPM send result API. Following ladder diagrams shows just one of the many ways of splitting the DPC across two cores.



5.3.2.4.1. Figure 21: Distributed Data processing flow and control (init/config)



|

5.3.2.4.2. Figure 22: Distributed Data processing flow and control (start/chirp/frame/stop)

5.4. mmWave SDK - TI components

The mmWave SDK functionality broken down into components are explained in next few subsections. For detailed documentation on each of these modules, refer to the top level documentation located at mmwave_mcuplus_sdk_<ver>/docs/mmwave_sdk_module_documentation.html.

5.4.1. Demos

5.4.1.1. mmWave Demo

This demo is located at mmwave_mcuplus_sdk_<ver>/ti/demo/<platform>/mmw folder. The millimeter wave demo shows some of the radar sensing and object detection capabilities of the 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. A detailed explanation of this demo is available in the demo's docs folder and can be browsed via mmwave_mcuplus_sdk_<ver>/ docs/mmwave_sdk_module_documentation.html. This section captures the high level layout of the demo supported on various mmWave devices. For details on individual components (control layer, datapath layer, etc), refer to the remaining sub-sections under "mmWave SDK - TI components".

Device Support	AM273X + AWR2243	AWR294X

Demo Directory	ti\demo\am273x\mmw\	ti\demo\awr294x\mmw\
Binary prefix	am273x_mmw_demo	awr294x_mmw_demo
EVM	AM273X EVM + AWR2243BOOST	AWR294X
Platform selection in Visualizer	AM273X-2243	AWR294X
mmWave API/RF control	R5F (MSS)	R5F (MSS)
Instrumentation via LVDS based streaming	Yes	Yes
Ethernet-based streaming of object data	Yes	Yes
Range Proc DPU	HWA based DPU (driven by DSP)	HWA based DPU (driven by DSP)
Doppler Proc DPU	HWA based DPU (driven by DSP)	HWA based DPU (driven by DSP)
CFAR DPU	HWA based DPU (driven by DSP)	HWA based DPU (driven by DSP)
AoA DPU	HWA based DPU (driven by DSP)	HWA based DPU (driven by DSP)
Range Proc DDMA DPU	HWA based DPU (driven by DSP)	HWA based DPU (driven by DSP)
Doppler Proc DDMA DPU	HWA based DPU (driven by DSP)	HWA based DPU (driven by DSP)
Range CFAR Proc DDMA DPU	HWA based DPU (driven by DSP)	HWA based DPU (driven by DSP)

5.4.2.
Drivers

Drivers encapsulate the functionality of the various hardware IPs in the system and provide a well defined API to the higher layers. Unlike the previous versions of the SDK, drivers are a part of the PDK which comes installed with the SDK package. The drivers are designed to be OS-agnostic via the use of OSAL layer. Following figure shows typical internal software blocks used in the SDK. Please refer to the PDK documentation for more details.

5.4.2.1.1. OSAL

An OSAL layer is present within the mmWave SDK to provide the OS-agnostic feature of the foundational components (drivers, mmWaveLink, mmWaveAPI). This OSAL provides an abstraction layer for some of the common OS services: Semaphore, Mutex, Debug, Interrupts, Clock, Memory, CycleProfiler. The source code and documentation of OSAL layer is a part of the PDK, which comes installed with the SDK package (<pdk_path>\packages\ti\osal). Please refer to the PDK documentation for more details.

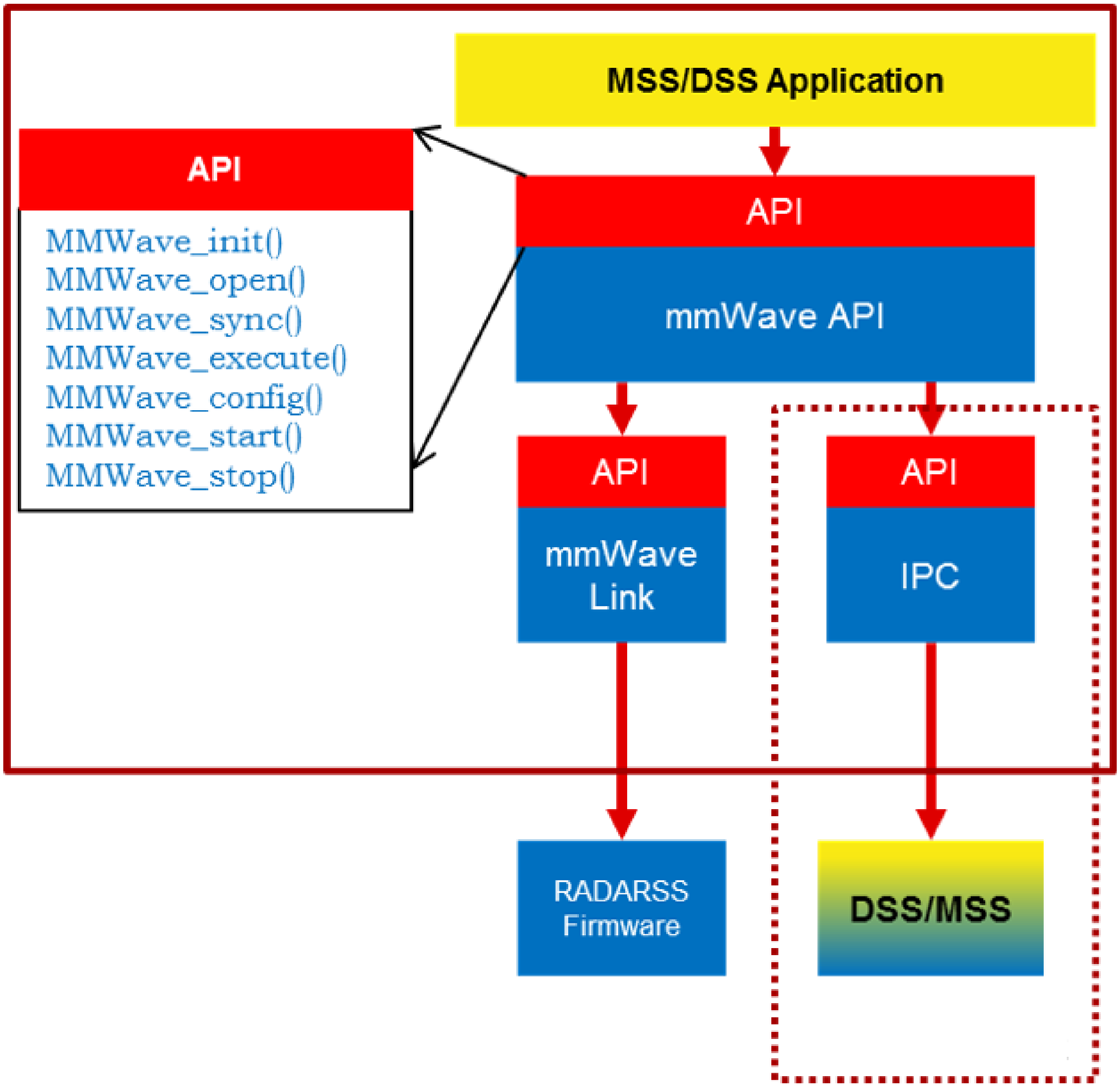
5.4.3. mmWaveLink

mmWaveLink is a control layer and primarily implements the protocol that is used to communicate between the Radar Subsystem (RADARSS) and the controlling entity which can be either Master subsystem (MSS R5F) and/or DSP subsystem (DSS C66x). It provides a suite of low level APIs that the application (or the software layer on top of it) can call to enable/configure/control the RADARSS. It provides a well defined interface for the application to plug in the correct communication driver APIs to communicate with the RADARSS. it acts as driver for Radar SS and exposes services of Radar SS. It includes APIs to configure HW blocks of Radar SS and provides communication protocol for message transfer between MSS/DSS and RADAR SS.

The mmwavelink comes as a part of the DFP package. Please refer to the folder mmwave_dfp_<version>\docs for more information. The DFP versions are different for AWR294X and AM273X+AWR2243.

5.4.4. mmWave API

mmWaveAPI is a higher layer control running on top of mmWaveLink and LLD API (drivers API). It is designed to provide a layer of abstraction in the form of simpler and fewer set of APIs for application to perform the task of mmWave radar sensing. In mmwave devices with dual cores, it also provides a layer of abstraction over IPC to synchronize and pass configuration between the MSS and DSS domains. The source code for mmWave API layer is present in the mmwave_mcuplus_sdk_<ver>\ti\control\mmwave folder. Documentation of the API is available via doxygen placed at mmwave_mcuplus_sdk_<ver>\ti\control\mmwave\docs\doxygen\html\index.html and can be easily browsed via mmwave_mcuplus_sdk_<ver>\docs\mmwave_sdk_module_documentation.html. The component's unit test code, running on top of SYSBIOS is also provided as part of the package: mmwave_mcuplus_sdk_<ver>\ti\control\mmwave\test\.



5.4.4.1.1. Figure 24: mmWave API - Internal software design

There are two modes of configurations which are provided by the mmWave module.

5.4.4.2. Full configuration

The "full" configuration mode implements the basic chirp/frame sequence of mmWave Front end and is the recommended mode for application to use when using the basic chirp/frame configuration. In this mode the application will use the entire set of services provided by the mmWave control module. These features includes:-

- Initialization of the mmWave Link
- Synchronization services between the MSS and DSS
- Asynchronous Event Management
- Start & Stop services
- Configuration of the RADARSS for Frame, advanced frame & Continuous mode
- Configuration synchronization between the MSS and DSS

In the full configuration mode; it is possible to create multiple profiles with multiple chirps. The following APIs have been added for this purpose:-

Chirp Management:

- MMWave_addChirp

- MMWave_delChirp

Profile Management:

- MMWave_addProfile
- MMWave_delProfile

mmWave Front End Calibrations

mmWave API, by default, enables all init/boot time time calibrations for mmWave Front End. There is a provision for user to provide custom calibration mask in MMWave_open API and/or to provide a buffer that has pre-stored calibration data.

When application requests the one-time and periodic calibrations in MMWave_start API call, mmWave API enables all the available one-time and periodic calibrations for mmWave Front End.

mmWave API doesn't expose the mmwavelink's LDO bypass API (rlRfSetLdoBypassConfig/rlRfLdoBypassCfg_t) via any of its API. If this functionality is needed by the application (either because of the voltage of RF supply used on the TI EVM/custom board or from monitoring point of view), user should refer to mmwavelink doxygen (mmwave_mcuplus_sdk_<ver>\ti\control\mmwavelink\docs\doxygen\html\index.html) on the usage of this API and call this API from their application before calling MMWave_open().

mmWave_open

Although mmWave_close API is provided, it is recommended to perform mmWave_open only once per power-cycle of the sensor.

5.4.5. Datapath Interface (DPIF)

DPIF defines the standard interface points in the detection processing chain that will correspond to the "blue" boxes in the scalable chain shown in the figure above. Key interfaces defined in this layer are:

- Input ADC data
- Radar Cube
- Detection Matrix
- Point cloud and its side info

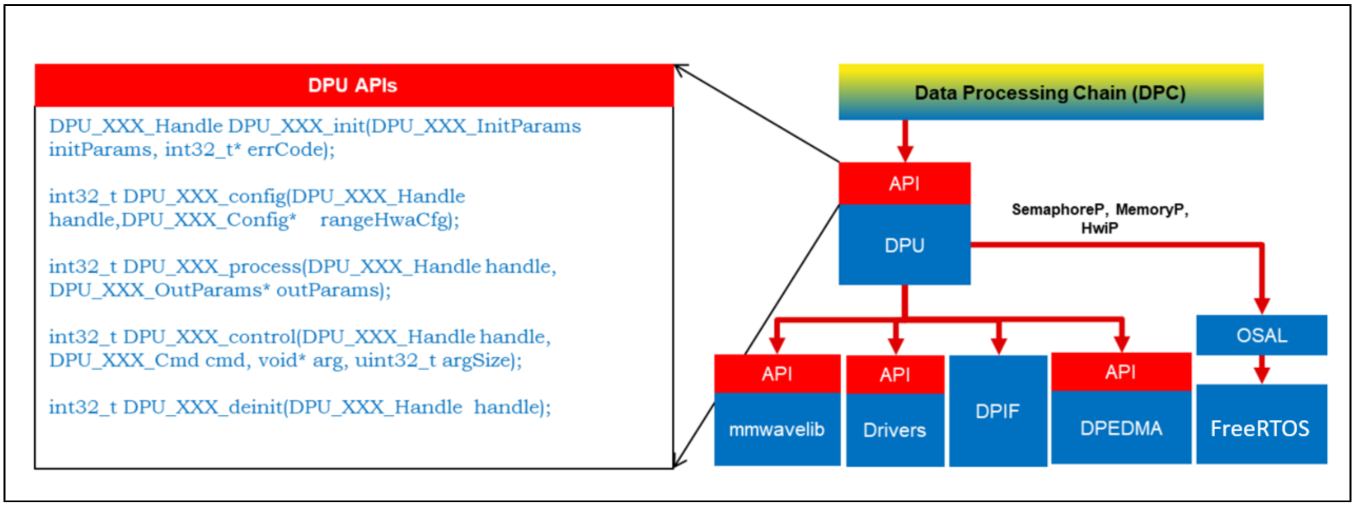
The source code for DPIF is present in the mmwave_mcuplus_sdk_<ver>\ti\datapath\dpif folder.

5.4.6. Data Processing Units (DPUs)

Data Translating function(s) from one interface point to the other are called “Data Processing Units”. Splitting the data processing chain into processing units promote re-use of certain processing blocks across multiple chains. Detailed documentation on these modules can be easily browsed via mmwave_mcuplus_sdk_<ver>/ docs/mmwave_sdk_module_documentation.html .

- Range Processing (ADC data to Radar Cube): This processing unit performs (1D FFT+ optional DC Range Calib) processing on the chirp (RF) data during the active frame time and produces the processed data in the radarCube. This processing unit is standardized in the mmWave SDK. It provides implementation based on HWA. HWA based implementation can be instantiated either on R5F or C66x. The source code for Range DPU is present in the mmwave_mcuplus_sdk_<ver>\ti\datapath\dpu\rangeproc folder. Documentation of the API is available via doxygen and placed at mmwave_mcuplus_sdk_<ver>\ti\datapath\dpu\rangeproc\docs\doxygen\html\index.html. The component's unit test code, running on top of SYSBIOS is also provided as part of the package: mmwave_mcuplus_sdk_<ver>\ti\datapath\dpu\rangeproc\ test\ .
- Doppler Processing (Radar Cube to Detection Matrix): This processing unit performs (2D FFT + Energy Sum) processing on the radar Cube during the inter frame and produced detection matrix. In addition to this, when static clutter removal is enabled, this processing unit reads Range FFT out data from the radar cube and performs static clutter removal, before performing the 2D FFT + Energy Sum processing. This processing unit is offered as reference implementation and users of SDK could either re-use these as is in their application/processing chain or create variations of these units based on their specific needs. HWA based implementation can be instantiated either on R5F or C66x. DSP based implementation incorporates static clutter algorithm for optimal memory/mips usage and user can skip using the standalone static clutter DPU. The source code for Doppler DPU is present in the mmwave_mcuplus_sdk_<ver>\ti\datapath\dpc\dpu\dopplerproc folder. Documentation of the API is available via doxygen and placed at mmwave_mcuplus_sdk_<ver>\ti\datapath\dpc\dpu\dopplerproc\docs\doxygen\html\index.html.
- CFAR + AoA (Detection Matrix to Point Cloud): They are offered as two independent DPUs and collectively run CFAR algorithm, peak grouping, field-of-view filtering, doppler compensation, max velocity enhancement and angle (azimuth+elevation) estimation on the detection matrix during inter frame to produce the point cloud. These processing units are offered as reference implementation and users of SDK could either re-use these as is in their application/processing chain or create variations of these units based on their specific needs. HWA based implementation can be instantiated either on R5F or C66x.
  - The source code for CFAR DPU is present in the mmwave_mcuplus_sdk_<ver>\ti\datapath\dpc\dpu\cfarproc folder. Documentation of the API is available via doxygen and placed at mmwave_mcuplus_sdk_<ver>\ti\datapath\dpc\dpu\cfarproc\docs\doxygen\html\index.html. The component's unit test code, running on top of SYSBIOS is also provided as part of the package: mmwave_mcuplus_sdk_<ver>\ti\ datapath\dpc\dpu\cfarproc\ test\.
  - The source code for AoA DPU is present in the mmwave_sdk_<ver>\ti\datapath\dpc\dpu\aoaproc folder. Documentation of the API is available via doxygen and placed at mmwave_mcuplus_sdk_<ver>\ti\datapath\dpc\dpu\aoaproc\docs\doxygen\html\index.html . The component's unit test code, running on top of SYSBIOS is also provided as part of the package: mmwave_mcuplus_sdk_<ver>\ti\ datapath\dpc\dpu\aoaproc\ test\ .

Each DPU presents the following high level design:

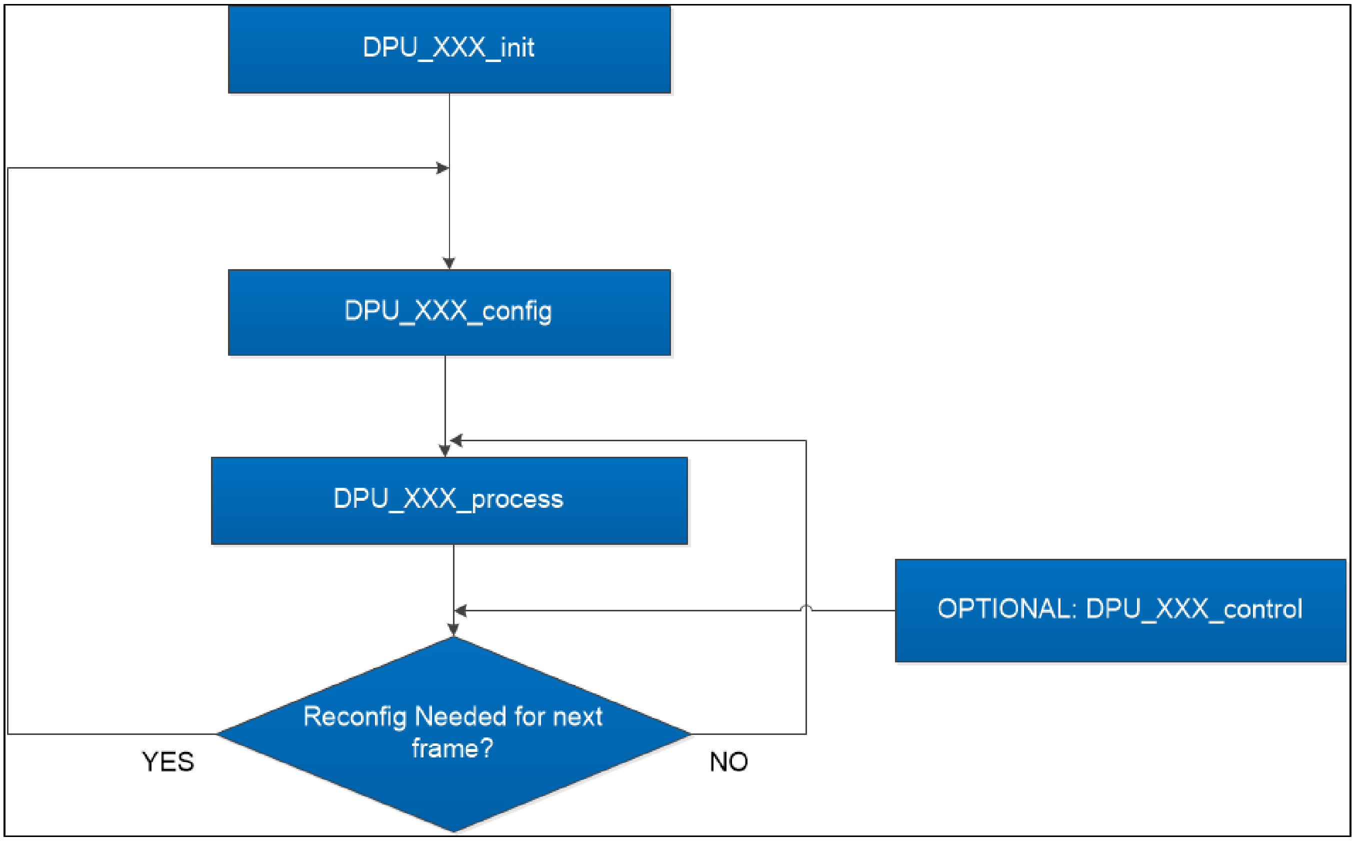


```
DPU_XXX_Handle DPU_XXX_init(DPU_XXX_InitParams
initParams, int32_t* errCode);

int32_t DPU_XXX_config(DPU_XXX_Handle
handle,DPU_XXX_Config*    rangeHwaCfg);

int32_t DPU_XXX_process(DPU_XXX_Handle handle,
DPU_XXX_OutParams* outParams);

int32_t DPU_XXX_control(DPU_XXX_Handle handle,
DPU_XXX_Cmd cmd, void* arg, uint32_t argSize);

int32_t DPU_XXX_deinit(DPU_XXX_Handle  handle);
```

5.4.6.1.1. Figure 25: DPU - Internal software design

- All external DPU APIs start with the prefix DPU_. DPU unique name follows next.
    - Ex: DPU_RangeProcHWA_init
- Standard external APIs: init, config, process, ioctl, deinit   are provided by each DPU.
    - Init: one time initialization of DPU
    - Config: complete configuration of the DPU: hardware resources, static and dynamic (if supported by DPU)
        - static config: config that is static during ongoing frames
        - dynamic config: config that can be changed from frame to frame but only when process is not ongoing - ideally interframe time after DPC has exported the results for the frame
    - Process: the actual processing function of the DPU
    - Ioctl: control interface that allows higher layer to switch dynamic configuration during interframe time
    - De-init: de-initialization of DPU
- All memory allocations for I/O buffers and scratch buffers are outside the DPU since mmWave applications rely on memory overlay technique for optimization and that is best handled at application level
- All H/W resources must be allocated by application and passed to the DPU. This helps in keeping DPU platform agnostic as well as allows application to share the resources across DPU when DPU processing doesn't overlap in time.
- DPUs are OS agnostic and use OSAL APIs for needed OS services.

A typical call flow for DPUs could be represented as follows. The timing of config and process API calls with respect to chirp/frame would vary depending on the DPU functionality, its usage in the chain, DPC implementation and overlap of hardware resources.

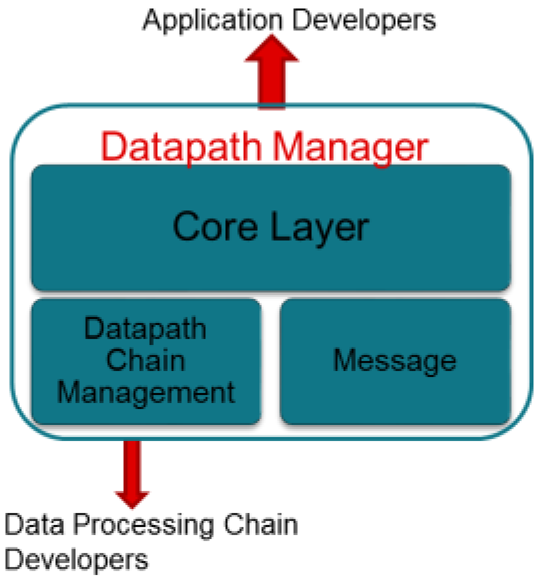


5.4.6.1.2. Figure 26: DPU - typical call flow

5.4.7. Data Path Manager (DPM)

DPM is the foundation layer that enables the "scalability" aspect of the architecture. This layer absorbs all the messaging complexities (cross core and intra core) and provide standard APIs for integration at the application level and also for integrating any "data processing chain". Application layer will be able to call the DPM APIs from any domain (MSS or DSS) and control the configuration and execution of the "data processing chain". The APIs offered by DPM will be available on both MSS and DSS. The various deployments that it can cater to (but not limited to) are:

- Datapath control on R5F and datapath execution is split between R5F/HWA and DSP (Distributed)
- Datapath control on R5F and datapath execution is on R5F using HWA (Local)
- Datapath control on R5F and datapath execution is on DSP (with and without HWA) (Remote)
- Datapath control on DSP and datapath execution is on DSP+HWA (Local)
- Datapath control on DSP and datapath execution is on DSP (Local)



5.4.7.1.1. Figure 27: Datapath manager (DPM) - internal software design

The source code for DPM is present in the mmwave_mcuplus_sdk_<ver>\ti\control\dpm folder. Documentation of the API is available via doxygen and placed at mmwave_mcuplus_sdk_<ver>\ti\ control\dpm \docs\doxygen\html\index.html. The component's unit test code, running on top of SYSBIOS is also provided as part of the package: mmwave_mcuplus_sdk_<ver>\ti\ control\dpm\ test\ .

5.4.8. Data processing chain (DPC)

DPC is a separate layer within the datapath that encapsulates all the data processing needs of a mmwave application and provides a well defined interface for integration with the application. In the SDK, there is a reference implementation that corresponds to the generic "object detection" chain which was already a part of the OOB demo in past releases. This chain will conform to the standard DPM dictated API definitions. Internally this layer will use the functionality exposed by Data processing units (DPUs), datapath interface and datapath manager (DPM) to realize the data flow needed for the "object detection" chain. The source code for objectdetection DPC is present in the mmwave_mcuplus_sdk_<ver>\ti\datapath\dpc\objectedetection folder. Documentation of the API is available via doxygen placed at mmwave_mcuplus_sdk_<ver>\ti\datapath\dpc\objectedetection\<deployment_type>\docs\doxygen\html\index.html and can be easily browsed via mmwave_mcuplus_sdk_<ver>/ docs/mmwave_sdk_module_documentation.html . The component's unit test code, running on top of SYSBIOS is also provided as part of the package mmwave_mcuplus_sdk_<ver>\ti\ datapath\dpc\objectedetection\objdethwa \test\. See section on Data Path tests using Test vector method for more details on this test.

5.4.9. Board/EVM Configuration

For Board/EVM specific information, refer to the PDK documentation and <pdk_path>\packages\ti\board directory for more details. For antenna configuration of the Board/EVM, refer to <sdk_path>\ti\board folder.

5.4.10. mmWaveLib

mmWaveLib is a collection of algorithms that provide basic functionality needed for FMCW radar-cube processing. It contains optimized library routines for C66x DSP architecture only. This component is not available for cortex R5F (MSS). These routines do not encapsulate any data movement/data placement functionality and it is the responsibility of the application code to place the input and output buffers in the right memory (ex: L2) and use EDMA as needed for the data movement. The source code for mmWaveLib is present in the mmwave_mcuplus_sdk_<ver>\packages\ti\alg\mmwavelib folder. Documentation of the API is available via doxygen placed at mmwave_mcuplus_sdk_<ver>\packages\ti\alg\mmwavelib\docs\doxygen\html\index.html and can be easily browsed via mmwave_mcuplus_sdk_<ver> /docs/mmwave_sdk_module_documentation.html. The component's unit test code, running on top of SYSBIOS is also provided as part of the package: mmwave_mcuplus_sdk_<ver>\packages\ti\alg\mmwavelib\test\ .

Functionality supported by the library:

- Collection of algorithms that provide basic functionality needed for FMCW radar-cube processing.
  - Windowing (16-bit complex input, 16 bit complex output, 16-bit windowing real array)
  - Windowing (16-bit complex input, 32 bit complex output, 32-bit windowing real array)
  - log2 of absolute value of 32-bit complex numbers
  - vector arithmetic (accumulation)
  - CFAR-CA, CFAR-CASO, CFAR-CAGO (logarithmic input samples)
  - 16-point FFT of input vectors of length 8 (other FFT routines are provided as part of DSPLib)
  - single DFT value for the input sequences at one specific index
  - Twiddle table generation for 32x32 and 16x16 FFTs: optimized equivalent functions of dsplib for generating twiddle factor
  - FFT Window coefficients generation
  - DFT sine/cosine table generation for DFT single bin calculation
  - Single bin DFT with windowing.
  - Variation of the windowing functions with I/Q swap since most of the fixed point FFT functions in DSPLib only support one format of complex types.
- CFAR algorithms
  - Floating-point CFAR-CA:
    - mmwavelib_cfarfloat_caall supports CFAR cell average, cell accumulation, SO, GO algorithms, with input signals in floating point formats;
    - mmwavelib_cfarfloat_caall_opt implements the same functionality as mmwavelib_cfarfloat_caall except with less cycles, but the detected objects will not be in the ascending order.
    - mmwavelib_cfarfloat_wrap implements the same functionality as mmwavelib_cfarfloat_caall except the noise samples for the samples at the edges are the circular rounds samples at the other edge.
    - mmwavelib_cfarfloat_wrap_opt implements the same functionality as mmwavelib_cfarfloat_wrap except with less cycles, but the detected objects will not be in the ascending order.
  - CFAR-OS: Ordered-Statistic CFAR algorithm
    - mmwavelib_cfarOS accepts fixed-point input data (16-bit log-magnitude accumulated over antennae). Search window size is defined at compile time.
  - Peak pruning for CFAR post-processing
    - mmwavelib_cfarPeakPruning: Accepts detection matrix and groups neighboring peaks into one.
    - mmwavelib_cfarPeakQualifiedInOrderPruning: Accepts the list of CFAR detected objects and groups neighboring peaks into one.
    - mmwavelib_cfarPeakQualifiedPruning: Same as mmwavelib_cfarPeakQualifiedInOrderPruning, but with no assumption for the order of cfar detected peaks
- Floating-point AOA estimation:
  - mmwavelib_aoaEstBFSinglePeak implements Bartlett beamformer algorithm for AOA estimation with single object detected, it also outputs the variance of the detected angle.
  - mmwavelib_aoaEstBFSinglePeakDet implements the save functionality as mmwavelib_aoaEstBFSinglePeak without the variance of detected angle calculation.
  - mmwavelib_aoaEstBFMultiPeak also implements the Bartlett beamformer algorithm but with multiple detected angles, it also outputs the variances for every detected angles.

    - mmwavelib_aoaEstBFMultiPeakDet implements the same functionality as mmwavelib_aoaEstBFMultiPeak but with no variances output for every detected angles.
- DBscan Clustering:
    - mmwavelib_dbscan implements density-based spatial clustering of applications with noise (DBSCAN) data clustering algorithm.
    - mmwavelib_dbscan_skipFoundNeiB also implements the DBSCAN clustering algorithm but when expanding the cluster, it skips the already found neighbors.
- Clutter Removal:
    - mmwavelib_vecsum: Sum the elements in 16-bit complex vector.
    - mmwavelib_vecsubc: Subtract const value from each element in 16-bit complex vector.
- Windowing:
    - mmwavelib_windowing16xl6_evenlen: Supports multiple-of-2 length(number of input complex elements), and mmwavelib_windowing16x16 supports multiple-of-8 length.
    - mmwavelib_windowing16x32: This is updated to support multiple-of-4 length(number of input complex elements). It was multiple-of-8 previously.
- Floating-point windowing:
    - mmwavelib_windowing1DFltp: support fixed-point signal in, and floating point signal out windowing, prepare the floating point data for 1D FFT.
    - mmwavelib_chirpProcWin2DFxdpinFltOut, mmwavelib_dopplerProcWin2DFxdpinFltOut: prepare the floating point data for 2D FFT, with fixed point input. The difference is mmwavelib_chirpProcWin2DFxdpinFltOut is done per chip bin, while mmwavelib_dopplerProcWin2DFxdpinFltOut is done per Doppler bin.
    - mmwavelib_windowing2DFltp: floating point signal in, floating point signal out windowing to prepare the floating point data for 2D FFT.
- Vector arithmetic
    - Floating-point and fixed point power accumulation: accumulates signal power. Alternate API to right shift the output vector along with accumulation is also provided.
    - Histogram: mmwavelib_histogram right-shifts unsigned 16-bit vector and calculates histogram.
    - Right shift operation on signed 16-bit vector or signed 32-bit vector
        - mmwavelib_shiftright16 shifts each signed 16-bit element in the input vector right by k bits.
        - mmwavelib_shiftright32 shifts each signed 32-bit element in the input vector right by k bits.
        - mmwavelib_shiftright32to16 right shifts 32-bit vector to 16-bit vector
    - Complex vector element-wise multiplication.
        - mmwavelib_vecmul16x16: multiplies two 16-bit complex vectors element by element. 16-bit complex output written in place to first input vector.
        - mmwavelib_vecmul16x32, mmwave_vecmul16x32_anylen : multiplies a 16-bit complex vector and a 32-bit complex vector element by element, and outputs to the 32-bit complex output vector.
        - mmwave_vecmul32x16c: multiplies 32bit complex vector with 16bit complex constant.
    - Sum of absolute value of 16-bit vector elements
        - mmwavelib_vecsumabs returns the 32-bit sum.
    - Max power search on 32-bit complex data
        - mmwavelib_maxpow outputs the maximum power found and returns the corresponding index of the complex sample
        - mmwavelib_powerAndMax : Power computation combined with max power search
    - Peak search for Azimuth estimation on 32-bit float vector
        - mmwavelib_multiPeakSearch : Multiple peak search in the azimuth FFT output
        - mmwavelib_secondPeakSearch : Second peak search in the azimuth FFT output
    - DC (antenna coupling signature) Removal on 32-bit float complex vector
    - Vector subtraction for 16-bit vectors
- Matrix utilities
    - Matrix transpose for 32-bit matrix: Similar to DSPLib function but optimized for matrix with rows larger than columns

5.4.11. Group Tracker

The algorithm is designed to track multiple targets, where each target is represented by a set of measurement points (point cloud output of CFAR detection layer). Each measurement point carries detection information, for example, range, angle, and radial velocity. Instead of tracking individual reflections, the algorithm predicts and updates the location and dispersion properties of the group. The group is defined as the set of measurements (typically, few tens; sometimes few hundreds) associated with a real life target. This algorithm is supported for both R5F and C66x. The source code for gtrack is present in the mmwave_mcuplus_sdk_<ver>\packages\ti\alg\gtrack folder. Documentation of the API is available via doxygen placed at mmwave_mcuplus_sdk_<ver>\packages\ti\alg\gtrack \docs\doxygen<2d|3D>\html\index.html and can be easily browsed via mmwave_mcuplus_sdk_<ver>\docs\mmwave_sdk_module_documentation.html.. The component's unit test code, running on top of SYSBIOS is also provided as part of the package: mmwave_mcuplus_sdk_<ver>\packages\ti\alg\gtrack\test\.

5.4.12. CCS Debug Utility

This is a simple binary that can flashed onto the board to facilitate the development phase of mmWave application using TI Code Composer Studio (CCS). See section CCS Development mode for more details. There is an executable for both R5F (MSS) and C6 (DSS) and is combined into one metaImage for flashing along with RADARSS firmware. When the appimage for CCS Debug is flashed onto the EVM, the cores can be connected to, reset, and have programs be loaded to them on Code Composer Studio.

5.4.13. HSI Header Utility

An optional utility library is provided for user to create a header that it can attach to the data being shipped over LVDS. This library accepts the CBUFF session configuration and creates a header with appropriate information filled in and passes it back to the calling application. The calling application can then supply this created header to CBUFF APIs. This config inside the header is intended to help user parse the LVDS on the receiving end. The source code for this utility is present in the mmwave_mcuplus_sdk_<ver>\packages\ti\utils\hsiheader folder. Documentation of the API is available via doxygen placed at mmwave_mcuplus_sdk_<ver>\packages\ti\utils\hsiheader\docs\doxygen\html\index.html and can be easily browsed via mmwave_mcuplus_sdk_<ver>\docs\mmwave_sdk_module_documentation.html. Note that HSI Header Utility is currently available for the CBUFF streaming test case but not the OOB Demo for AWR294X.

5.4.14. Secondary Bootloader

A simple metaimage creation utility is provided in the SDK (mmwave_mcuplus_sdk_<ver>\scripts\unix\generateMetaImage.sh for UNIX and mmwave_mcuplus_sdk_<ver>\scripts\windows\generateMetaImage.bat for Windows). The source code for the SBL comes as a part of the PDK can can be found in the <pdk_path>\packages\ti\boot\sbl folder. Please refer to the PDK documentation for more details.

5.4.15. mmWave SDK - System Initialization

Application should call Board_init API to enable correct operation of the device. For more information, refer to the PDK documentation.

5.4.15.1. Pinmux

The source code for Pinmux module is generated by sysconfig. Please refer to the MCU PLUS SDK documentation and user guide for more details.

5.4.16. Usecases

5.4.16.1. Data Path tests using Test vector method

The data path processing on mmWave device for 1D, 2D and 3D processing consists of a coordinated execution between the MSS, HWA/DSS and EDMA. This is demonstrated as part of the object detection processing chain and millimeter wave demo. The demo runs in real-time and has all the associated framework for RADARSS control etc with it.

The unit tests located at mmwave_mcuplus_sdk_<ver>\ti\datapath\dpc \objectdetection\<chain_type>\test) are stand-alone tests that allow data path processing chain to be executed in non real-time. This allows developer to use it as a debug/development aid towards eventually making the data path processing real-time with real chirping. Developer can easily step into the code and test against knowns input signals. The core data path processing source code in object detection chain and the processing modules (DPUs) is shared between this test and the mmw demo. Most of the documentation is therefore shared as well and can be looked up in the object detection DPC and mmw demo documentation.

The tests also provide a test generator, which allows user to set objects artificially at desired range, doppler and azimuth bins, and noise level so that output can be checked against these settings. It can generate one frame of data. The test generation and verification are integrated into the tests, allowing developer to run a single executable that contains the input vector and also verifies the output (after the data path processing chain), thereby declaring pass or fail at the end of the test. The details of test generator can be seen in the doxygen documentation of these tests located at mmwave_mcuplus_sdk_<ver>\ti\datapath\dpc \objectdetection\<chain_type>\test\docs\doxygen\html\index.html and can be easily browsed via mmwave_mcuplus_sdk_<ver>/ docs/mmwave_sdk_module_documentation.html .

6. Appendix

6.1. Memory usage

The map files of demo and driver unit test application captures the memory usage of various components in the system. They are located in the same folder as the corresponding .xer5f/.xe66 and .bin files. Additionally, the doxygen for mmW demo summarizes the usage of various memories available on the device across the demo application and other SDK components. Refer to the section "Memory Usage" in the mmwave_mcuplus_sdk_<ver>\ti\demo\<platform>\mmw\docs\doxygen\html\index.html documentation.

6.2. Shared memory usage by SDK demos

Unlike earlier devices like the 18xx, AM273X and AWR294X do not have a specific HSRAM (Handshake RAM). For the purpose of the demo, 32 KB of the L3 memory (defined in mmwave_mcuplus_sdk_<ver>\ti\common\sys_common.h) is reserved as shared memory between the MSS and the DSS so that the MSS can read the results populated post the DPC execution.

mmWave Device Image Creator

This section outlines the tools used for image creation needed for flashing the mmWave devices. The application executable generated after the compile and link step needs to be converted into a bin form for the bootloader to understand and burn it onto the serial flash present on the device. The demos inside the mmWave SDK already incorporate the step of bin file generation as part of their makefile and no further steps are required. This section is helpful for application writers that do not have makefiles similar to the SDK demos. Once the compile and link step is done, application image generation is described as follows.

The Application Image interpreted by the bootloader is a consolidated Multicore image file that includes the RPRC image file of individual subsystems along with a Meta header. The Meta Header is a Table of Contents like information that contains the offsets to the individual subsystem RPRC images along with an integrity check information using CRC. In addition, the allocation of the shared memory to the various memories of the subsystems also has to be specified. The bootloader performs the allocation accordingly. It is recommended that the allocation of shared memory is predetermined and not changed dynamically.

Use the generateMetaImage script present under mmwave_mcuplus_sdk_<ver>\scripts\windows or mmwave_mcuplus_sdk_<ver>\scripts\linux for merging the MSS .xer5f and DSS .xe66 into one metaImage and appending correct CRC. The RPRC image for MSS and DSS are generated internally in this script from the input ELF formatted files for those subsystem (i.e. output of linker command - .xer5f, .xe66). Call setenv.bat or setenv.sh present under mmwave_mcuplus_sdk_<ver>\scripts\windows or mmwave_mcuplus_sdk_<ver>\scripts\linux for setting up the environment before calling this script. This script needs 5 parameters:

- FLASHIMAGE: [output] multicore file that will be generated by this script and should be used for flashing onto the board
- MSS_IMAGE_OUT: [input] MSS input image in ELF (.xer5f) format as generated by the linker. Use keyword NULL if not needed
- BSS_IMAGE_BIN: [input] RADARSS (BSS) input image in RPRC (.bin) format. Use keyword NULL in the context of the AM273X. In case of AWR294X, use the BSS image path from the setenv.bat file.
- DSS_IMAGE_OUT: [input] DSP input image in ELF (.xe66) format as generated by the linker. Use keyword NULL if not needed

The FLASHIMAGE file generated by this script should be used during flashing step (How to flash an image onto mmWave EVM).

6.3. Range Bias and Rx Channel Gain/Offset Measurement and Compensation (TDM Demo)

Refer to the section "Range Bias and Rx Channel Gain/Offset Measurement and Compensation" in the mmwave_mcuplus_sdk_<ver>\ti\datapath\dpc\objectdetection\<chain_type>\docs\doxygen\html\index.html documentation for the procedure and internal implementation details. To execute the procedure using Visualizer GUI, here are the steps:

- Set the target as explained in the demo documentation and update the mmwave_mcuplus_sdk_<ver>\ti\demo\<platform>\mmw\profiles\tdm<>\profile_calibration.cfg appropriately.
- Set up Visualizer and mmW demo as mentioned in the section Running the Demos.
- Use the "Load Config From PC and Send" button on plots tab to send the mmwave_mcuplus_sdk_<ver>\ti\demo\<platform>\mmw\profiles\tdm<>\profile_calibration.cfg.
- The Console messages window on the Configure tab will dump the "compRangeBiasAndRxChanPhase" command to be used for subsequent runs where compensation is desired.
- Copy and save the string for that particular mmWave sensor to your PC. You can use it in the "Advanced config" tab in the Visualizer and tune any running profile in real time. Alternatively, you can add this to your custom profile configs and use it via the "Load Config From PC and Send" button.

6.4. Guidelines on optimizing memory usage

Depending on requirements of a given application, there may be a need to optimize memory usage, particularly given the fact that the mmWave devices do not have external RAM interfaces to augment on-chip memories. Below is a list of some optimizations techniques, some of which are illustrated in the mmWave SDK demos (mmW demo). It should be noted, however, that the demo application memory requirements are dictated by requirements like ease/flexibility of evaluation of the silicon etc, rather than that of an actual embedded product deployed in the field to meet specific customer user cases.

1. On R5F, compile your application with ARM thumb option (depending on the compiler use). If using the TI ARM compiler, the option to do thumb is `code_state=16` . Another relevant compiler option (when using TI compiler) to play with to trade-off code size versus speed is `--opt_for_speed=0-5` . For more information, refer to ARM Compiler Optimizations and ARM Optimizing Compiler User's Guide. The pre-built drivers in the SDK are already built with the thumb option. The demo code and BIOS libraries are also built with thumb option. Note the code_state=16 flag and the ti.targets.arm.elf.**R5Ft** target in the mmwave_mcuplus_sdk_<ver>\ti\ common\mmwave_sdk.mak.

2. On C66X, compile portions of code that are not in compute critical path with appropriate -mfX option. The -mf3 options is presently used in the SDK drivers, demos and BIOS cfg file. This option does cause compiler to favor code size over performance and hence some cycles impact are to be expected. However, on mmWave device, using mf3 option only caused about 1% change in the CPU load during active and interframe time and around 3-5% increase in config cycles when benchmarked using driver unit tests. For more details on the "mf" options, refer to The TI C6000 compiler user guide at C6000 Optimizing Compiler Users Guide. Another option to consider is -mo (this is used in SDK) and for more information, see section "Generating Function Subsections (--gen_func_subsections Compiler Option)" in the compiler user guide. A link of references for optimization (both compute and memory) is at Optimization Techniques for the TI C6000 Compiler.
3. Even with aggressive code size reduction options, the C66X tends to have a bigger footprint of control code than the same C code compiled on R5F. So if feasible, partition the software to use C66X mainly for compute intensive signal-processing type code and keep more of the control code on the R5F. An example of this is in the mmw demo, where we show the usage of mmwave API to do configuration (of RADARSS) from R5F instead of the C66X (even though the API allows usage from either domain). In mmw demo, this prevents linking of `mmwave` (in mmwave_mcuplus_sdk_<ver>\ ti\control) and mmwavelink (in mmwave_mcuplus_sdk_<ver>\ti\control) code that is involved in configuration (profile config, chirp config etc) on the C66X side as seen from the .map files of mss and dss located along with application binary.
4. If there is no requirement to be able to restart an application without reloading, then following suggestions may be used:
    a. one time/first time only program code can be overlaid with data memory buffers used after such code is executed. Note: Ability to place code at function granularity requires to use the aforementioned -mo option.
    b. the linker option `--ram_model` may be used to eliminate the `.cinit` section overhead. For more details, see compiler user guide referenced previously. Presently, ram model cannot be used on R5F due to bootloader limitation but can be used on C66X. The SDK uses ram model when building C66X executable images (unit tests and demos).
5. On C66X, smaller L1D/L1P cache sizes may be used to increase static RAM. The L1P and L1D can be used as part SRAM and part cache. Smaller L1 caches can increase compute time due to more cache misses but if appropriate data/code is allocated in the SRAMs, then the loss in compute can be compensated (or in some cases can also result in improvement in performance). In the demos, the caches are sized to be 16 KB, allowing 16 KB of L1D and 16 KB of L1P to be used as SRAM. Typically, the L1D SRAM can be used to allocate some buffers involved in data path processing whereas the L1P SRAM can be used for code that is frequently and more fully accessed during data path processing. Thus we get overall 32 KB more memory. The caches can be reduced all the way down to 0 to give the full 32 KB as SRAM: how much cache or RAM is a decision each application developer can make depending on the memory and compute (MIPS) needs.
6. When modeling the application code using mmW demo as reference code, it might be useful to trim the heaps in mmW demo to claim back the unused portion of the heaps and use it for code/data memory. Ideally, a user can run their worst case profile that they would like to support using mmW demo, record the heap usage/free metrics for (L1D, L2)/TCMB and L3 heaps on 'sensorStart'. These values can then be used to resize or re-allocate heap globals (example: gDPC_ObjDetTCM, gMmwL3, etc) in mmwave_mcuplus_sdk_<ver>\ti\demo\<platform>\mmw. The freed up space in DSS could be used as follows:

    a. Free heap space in L1D could be used to move some of the L2 buffers to L1D. The freed L2 space can be used for code/data.
    b. Free heap space in L2 could be trimmed by changing the heap's global variable (ex: gMmwL2) definition and used for code/data memory (note that code memory by default is L2 so no other change is required to get more code space).
    c. Free heap space in L3 could be trimmed by changing the heap's global variable (ex: gMmwL3) definition and used for code/data space.

When using TI compilers for both R5F and C66x, the map files contain a nice module summary of all the object files included in the application. Users can use this as a guide towards identifying components/source code that could be optimized. See one sample snapshot below:

Module summary inside application's .map file

```
MODULE SUMMARY

       Module                          code     ro data   rw data
       ------                          ----     -------   -------


    C:/mmwave_app_sdk/ti/control/dpm/lib/libdpm_am273x.ae66
       dpm_msg.oe66                 2592     80        0
       dpm_core.oe66                2080     32        0
       dpm_mailbox.oe66             736      20        24
       dpm_pipe.oe66                704      16        0
       dpm_listlib.oe66             224      0         0
       dpm_am273x.oe66               64       0         0
    +--+----------------------------+--------+---------+---------+
       Total:                       6400     148       24
```

6.5. How to add a .const (table) beyond L3 heap in mmWave application where overlay is enabled

To achieve L3 heap overlaid with the code to be copied into L1P at init time, L3 heap is in PAGE 1 and code is in Page 0. PAGE 0 is the only loadable page whereas PAGE 1 is just a dummy page to allocate uninitialized sections to implement overlay. As a result the ".const" section (which is loadable section) cannot simply be allocated to PAGE 1 to go after the heap. If the .const is allocated in PAGE 0, then it will overlap the heap and will be overwritten once heap is allocated. To resolve this, the HIGH feature of the linker could be used is used to push the const table to the highest address ensuring no overlap with L3 heap. The suggested changes would be as follows:
1. Shrink the L3 heap by the size of the table (but L3 heap must still be bigger than the size of the L1P cache).
2. Place the table in a named section and allocate the named section in the HIGH space of PAGE 0 of L3RAM.

This ensures that the table will be allocated at the high address and will not be overlapping with L3 heap or the L1P intended code which are located at the low address.

Sample code is shown below.

```
In application C file:

#define TABLE_LENGTH 4
#define TABLE_ALIGNMENT 8 /* bytes */

/*! L3 RAM buffer, shrunk by table */
#pragma DATA_SECTION(gMmwL3, ".l3data");
#pragma DATA_ALIGN(gMmwL3, 8);
uint8_t gMmwL3[<DSS L3 RAM size> - TABLE_LENGTH*sizeof(float) - TABLE_ALIGNMENT];

#pragma DATA_SECTION(gArray, ".l3data_garray");
#pragma DATA_ALIGN(gArray, TABLE_ALIGNMENT);
const float gArray[TABLE_LENGTH] = {1.5, 3.2, 0.8, -9.6};

In linker command file:
    .l3data_garray:  load=L3SRAM PAGE 0 (HIGH)
```

6.6. Enabling L3 cache for DSP/C66x on mmWave devices

In a given usecase for mmWave devices, if L3 RAM is not fully utilized for Radar Cube storage, then the remaining free L3 memory could ideally be used for code and other internal data storages for the application. However, access to L3 memory from DSP/C66x core in mmWave devices is slower than accessing L1/L2. The cache-based memory system of C66x can be efficiently used in such cases. Refer to C66x DSP Cache User Guide (https://www.ti.com/lit/ug/sprugy8/sprugy8.pdf) for more details on the L1P/L1D/L2 two-level hierarchy that exists within the C66x memory architecture. L1P, L1D and L2D can be partitioned into SRAM and cache. L1P, L1D and L2 cache size can be set through linker command file -please refer to mmwave_mcuplus_sdk_<ver>/ti/platform/<platform>/c66x_linker.cmd for more details. L2 SRAM addresses are always cached in L1P and L1D. However, external memory addresses (ex: code/data in L3) by default are configured as non-cacheable in L1D and L2 caches. Cacheability for external memory addressed (ex: L3) must first be explicitly enabled by the user using the MAR registers. Note that L1P cache is not affected by this configuration and always caches external memory addresses.

- **Cache writeback**: To maintain cache coherency between different masters (CPU, DMA, R5F, etc), content in cache needs to be written back to memory after it is changed before triggering the other master to access that memory location.
- **Cache Invalidate**: Before reading the content from the physical memory that was updated by another master, the content in cache needs to be invalidated, so that updated data from memory can be loaded in cache.
- **Code in L3**: mmWave code can be placed from L2 to L3 (via linker command file) with no explicit need for cache enablement and/or cache operations during real time. The only setting that needs to be adjusted is the size of L1P cache and that should be balanced against the need for L1P SRAM to place real time optimized functions (and avoid any cache misses, etc).
- **Data in L3**: If data cache is enabled for L3 memory via the MAR registers, then at first, one needs to take care of cache invalidates and writebacks for existing data structures in L3 memory. Radarcube and detection matrix are the primary data structures placed in L3 memory in case of a typical mmwave application on our device. Typically Radarcube is accessed (read/write) only via EDMA during the Range and Doppler FFT. Post that, it is more common for the DSP core to access the radarcube directly (i.e. no EDMA) and primarily it is a read access. In such scenario, the Radarcube can be invalidated at the end of current frame but before the start of next frame (i.e. when EDMA master begins to access radarcube). If the Radarcube was modified by the core directly (write operation) during the interframe time, then cache writeback_invalidate is needed at the end of current frame but before the start of next frame. Same consideration would apply for detection matrix. Next, mmWave internal data structures that are accessed purely by DSP can also be moved from L2 to L3 (via linker command file). No explicit cache writeback/invalidations are required for such structures. If user chooses to place the frame results structures in L3 (point cloud, etc) which are shared with MSS (R5F), then cache writeback+invalidate needs to be performed before signaling the MSS about availability of frame results. Note: If the analysis of L3 data access pattern between the DSP, MSS and EDMA shows that cache writeback /invalidate of all L3 data content can be done towards the end of the current frame, then performing writeback+invalidate on entire L1D cache might be a better option than calling such API on individual structures.

6.7. SDK Demos: miscellaneous information

A detailed explanation of the mmW demo is available in the demo's docs folder: mmwave_mcuplus_sdk_<ver>\ti\demo\<platform>\mmw\docs\doxygen\html\index.html. Some miscellaneous details are captured here:

- In demos that use HWA as the only processing node and elevation is enabled during run-time via configuration file, the number of detected objects are limited by the amount of HWA memory that is available for post processing.
- Output packet of mmW demo data over UART is in TLV format and its length is a multiple of 32 bytes. This enables post processing elements on the remote side (PC, etc) to process TLV format with header efficiently.

6.8. CCS Debugging of real time application

It is relatively easier to debug code before real-time starts because single-stepping or adding break-points does not affect the debugging since there is no real-time data and deadline to process the data. But once real-time starts, which is after sensor is started, such debugging can be intrusive and problematic. Below are some tips that may be helpful in real-time debugging, some of them are relevant to the out of box demos but may be applied in user applications if relevant.

6.8.1. Inter-chirp debugging

In out of box demos and many application specialized demos based on the SDK provided by TI (through the TI resource explorer), the inter-chirp processing is based on HWA or DSP but not a mix of the two. In the case of HWA which also is what is used for processing in the current demo, the CPU/CPUs are idling with respect to inter-chirp processing so there is no need to halt. If one intends to stop and examine the state of HWA-EDMA during any of the intermediate processing steps, the design would have to be changed to issue a HWA or EDMA interrupt to the CPU that configured these (typically MSS CPU) at this intermediate state and the interrupt could read out some state and store in global variables that could be examined later. If code is halted using a break-point in the interrupt, the EDMA will automatically halt but HWA will not unless HWA is waiting on EDMA, so HWA could continue to run even if the CPU is halted. The current radar SoCs do not have the feature to halt the HWA when any of the CPUs are halted.

In case of DSP doing the inter-chirp processing, there can be a need to single-step/break the processing. However, (unlike the MSS CPU) when DSP is halted, the RadarSS (front end) doesnt halt and the chirping activity does not stop. Because of this, the DSP will miss the chirp processing deadline and the code is typically written to throw an exception. So basically halted debug is not useful unless a single chirp is configured and problem can be recreated with a single chirp. There might be other limitations in the demo code that may prevent a single chirp configuration (e.g minimum number of doppler bins). Other techniques shown in below sections (real-time logging, using non real-time unit test bench) may be more practical but have their own limitations. In most implementations however, 1D processing uses a hardened component from the SDK - the range DPU - so the need for real-time debugging in the active chirping period is low.

6.8.2. Inter-frame debugging

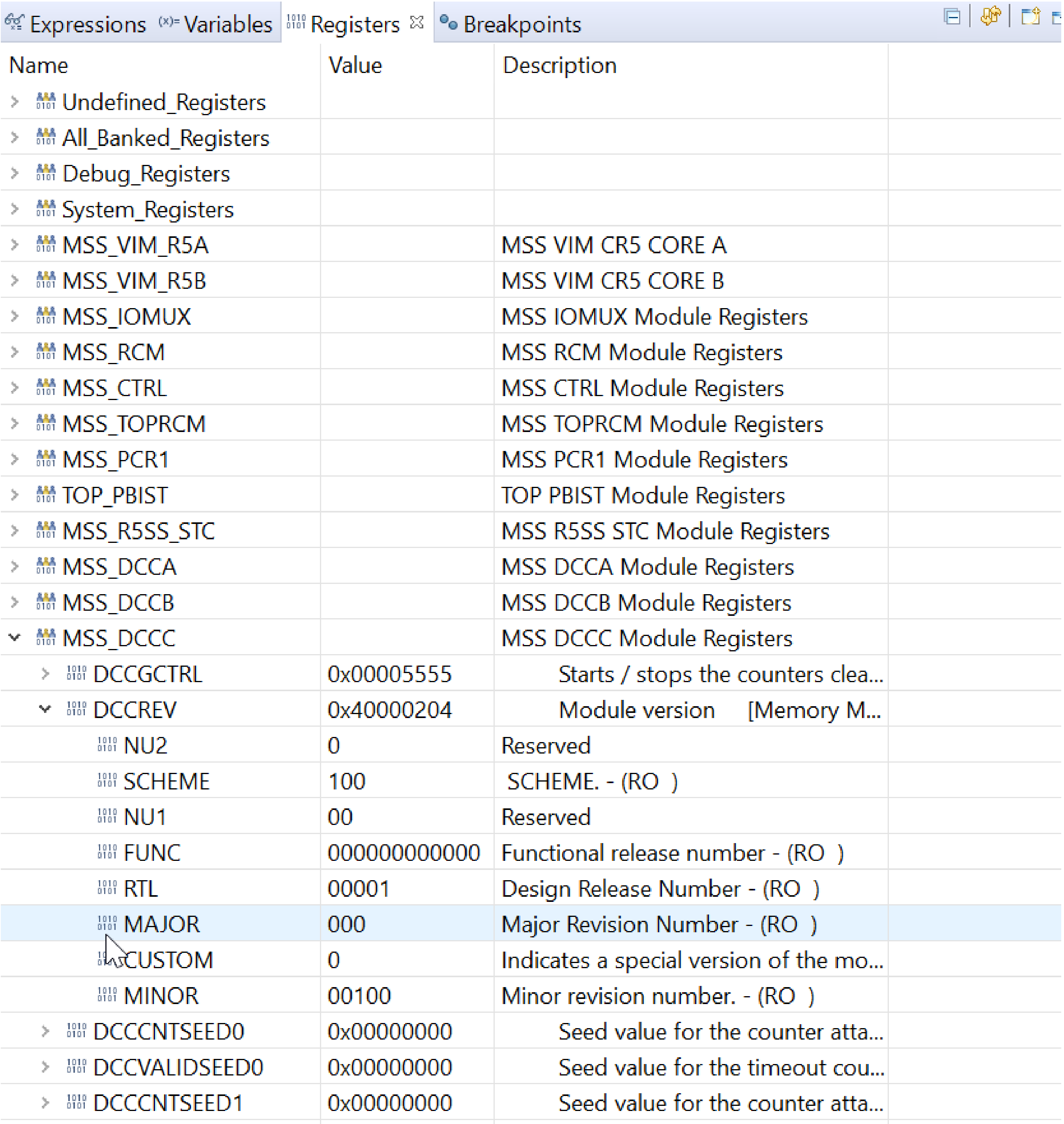
As there is no RadarSS chirping activity when MSS CPU is halted, it is possible to do halted debug in MSS during inter-frame debugging without running out of real-time. But on DSP, the device behavior is the opposite i.e. the chirping will continue even if DSP is in halted state, so stepping in the DSP will cause an inter-frame deadline miss exception when running the out of box demo and other special demos that are implemented similarly. One technique that may be helpful in this situation is if the problem can be observed in the first frame itself, configure the chirping profile to do only one frame (frameCfg CLI command). This way after the active frame period, there is no chirp overrun (of the next frame) pressure when single-stepping in the inter-frame processing.

6.8.3. Using non-real time chain test code

See section "Data Path tests using Test vector method" on details about the non-real time chain that is provided with the mmWave SDK. Users can use these tests to step through the OOB processing chain in non-real time mode and debug or learn the components of the OOB processing chain.

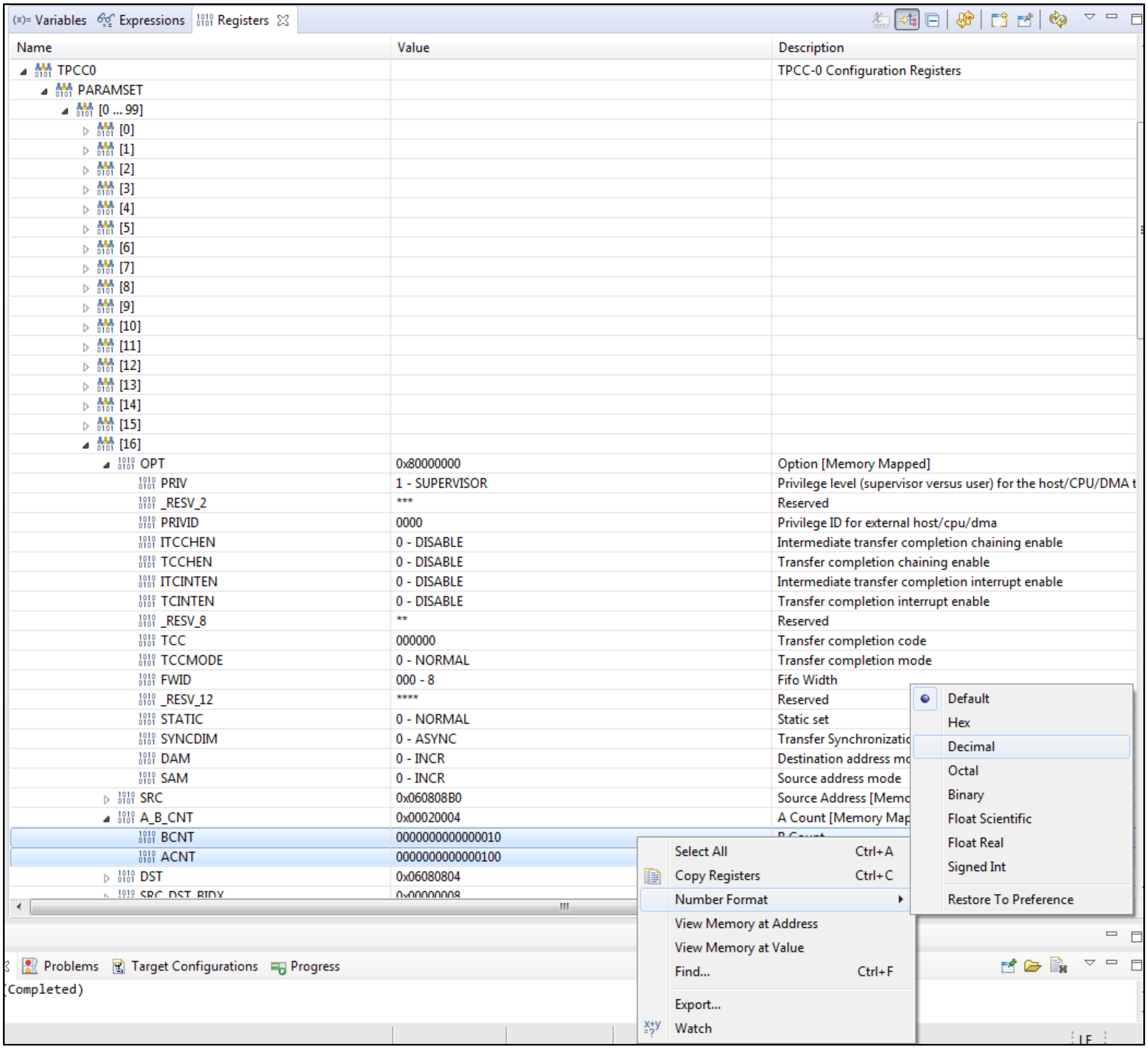
6.8.4. Viewing hardware registers

During debug, there may be a need to examine registers of HWA, EDMA, external I/O peripherals etc. These can be done using View->Registers menu and when a core is selected, the register view will display all registers that the core can see organized into various categories. An example is shown below:



Name	Value	Description
Undefined_Registers		
All_Banked_Registers		
Debug_Registers		
System_Registers		
MSS_VIM_R5A		MSS VIM CR5 CORE A
MSS_VIM_R5B		MSS VIM CR5 CORE B
MSS_IOMUX		MSS IOMUX Module Registers
MSS_RCM		MSS RCM Module Registers
MSS_CTRL		MSS CTRL Module Registers
MSS_TOPRCM		MSS TOPRCM Module Registers
MSS_PCR1		MSS PCR1 Module Registers
TOP_PBIST		TOP PBIST Module Registers
MSS_R5SS_STC		MSS R5SS STC Module Registers
MSS_DCCA		MSS DCCA Module Registers
MSS_DCCB		MSS DCCB Module Registers
MSS_DCCC		MSS DCCC Module Registers
DCCGCTRL	0x00005555	Starts / stops the counters clea...
DCCREV	0x40000204	Module version [Memory M...
NU2	0	Reserved
SCHEME	100	SCHEME. - (RO)
NU1	00	Reserved
FUNC	000000000000	Functional release number - (RO)
RTL	00001	Design Release Number - (RO)
MAJOR	000	Major Revision Number - (RO)
CUSTOM	0	Indicates a special version of the mo...
MINOR	00100	Minor revision number. - (RO)
DCCCNTSEED0	0x00000000	Seed value for the counter atta...
DCCVALIDSEED0	0x00000000	Seed value for the timeout cou...
DCCCNTSEED1	0x00000000	Seed value for the counter atta...

Individual hardware entities can be expanded further in the view to see registers specific to the hardware entity. The following picture illustrates viewing a certain PARAM set in instance #0 of the EDMA (TPCC0), note how the bit fields are automatically parsed and displayed in a user friendly manner which saves the burden of manually parsing or developing special parsing tools and facilitates quick debugging. Default number formats of bit fields are binary which is not always convenient, this can be changed by selecting the field/fields and right-clicking to see the number format menu as shown in the example below where the A and B counts of EDMA are about to be chosen for Decimal format. Once chosen, the GUI will remember the user choice for that specific field so user does not have to repeat this action in future debug sessions.



Name	Value	Description
TPCC0		TPCC-0 Configuration Registers
PARAMSET		
[0 ... 99]		
[0]		
[1]		
[2]		
[3]		
[4]		
[5]		
[6]		
[7]		
[8]		
[9]		
[10]		
[11]		
[12]		
[13]		
[14]		
[15]		
[16]		
OPT	0x80000000	Option [Memory Mapped]
PRIV	1 - SUPERVISOR	Privilege level (supervisor versus user) for the host/CPU/DMA t
_RESV_2	***	Reserved
PRIVID	0000	Privilege ID for external host/cpu/dma
ITCCHEN	0 - DISABLE	Intermediate transfer completion chaining enable
TCCHEN	0 - DISABLE	Transfer completion chaining enable
ITCINTEN	0 - DISABLE	Intermediate transfer completion interrupt enable
TCINTEN	0 - DISABLE	Transfer completion interrupt enable
_RESV_8	**	Reserved
TCC	000000	Transfer completion code
TCCMODE	0 - NORMAL	Transfer completion mode
FWID	000 - 8	Fifo Width
_RESV_12	****	Reserved
STATIC	0 - NORMAL	Static set
SYNCDIM	0 - ASYNC	Transfer Synchronizatic
DAM	0 - INCR	Destination address mc
SAM	0 - INCR	Source address mode
SRC	0x060808B0	Source Address [Memc
A_B_CNT	0x00020004	A Count [Memory Map
BCNT	0000000000000010	
ACNT	0000000000000100	
DST	0x06080804	
SRC_DST_BIDX	0x00000008	

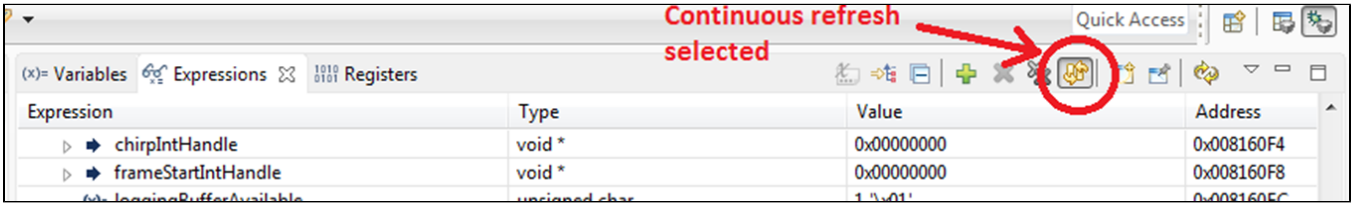
```
(Completed)
```

In the above picture, one can also see the "Watch" menu item. If this is selected, then the two fields of interest will appear in the Expressions view, this is a convenient way to see some fields of interest during debug without having to navigate the register structure again (although when a particular structure such as PARAM set #16 above is expanded, if the top level TPCC0 is shrunk and expanded again, the PARAM #16 is shown expanded as before because GUI remembers sub-structure expansion/non-expansion state).

6.8.5. Viewing expressions/memory in real time

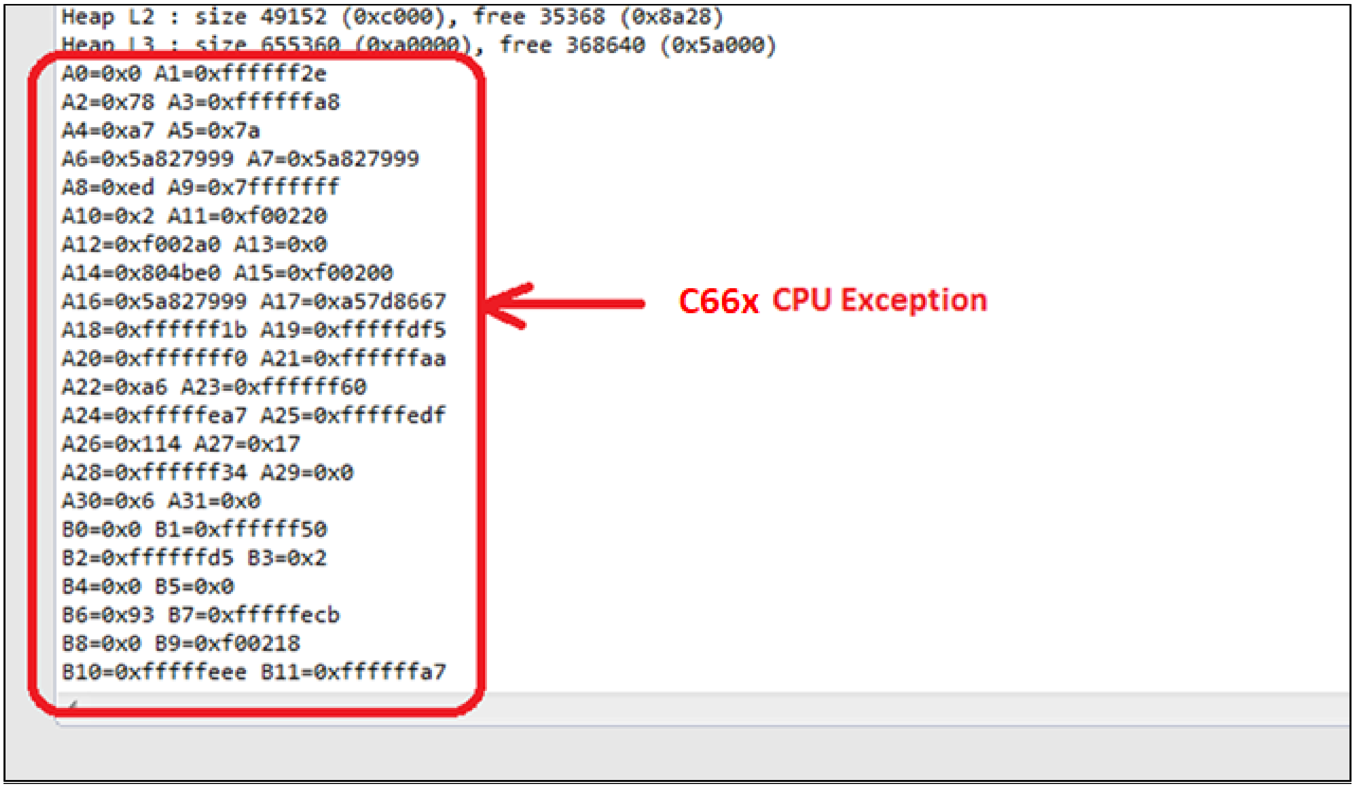
When debugging real time application (for example: mmw demo) in CCS, if the continuous refresh of variables in the Expression or Memory browser window is enabled without enabling the silicon real-time mode as shown in the picture, the code may crash at a random time showing the message in the console window. To avoid this crash, please put CCS in to “Silicone Real-time” mode after selecting the target core.

Continuous refresh:



Expression	Type	Value	Address
chirpIntHandle	void *	0x00000000	0x008160F4
frameStartIntHandle	void *	0x00000000	0x008160F8
loggingBufferAvailable	unsigned char	1 '\x01'	0x008160FC

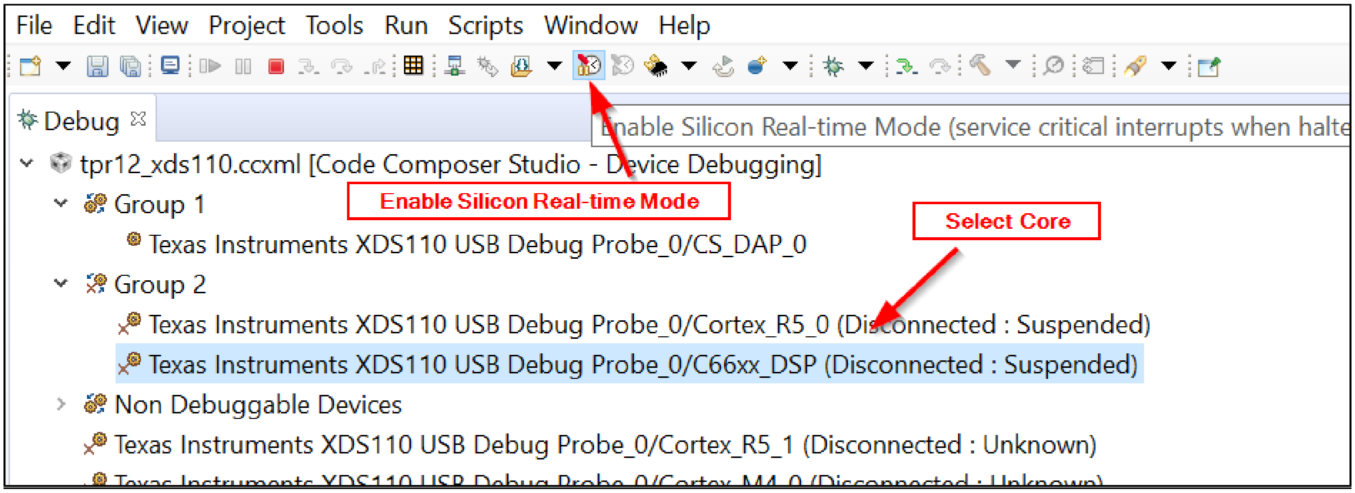
Crash in Console window:



```
Heap L2 : size 49152 (0xc000), free 35368 (0x8a28)
Heap L3 : size 655360 (0xa0000), free 368640 (0x5a000)
A0=0x0 A1=0xffffff2e
A2=0x78 A3=0xffffffa8
A4=0xa7 A5=0x7a
A6=0x5a827999 A7=0x5a827999
A8=0xed A9=0x7fffffff
A10=0x2 A11=0xf00220
A12=0xf002a0 A13=0x0
A14=0x804be0 A15=0xf00200
A16=0x5a827999 A17=0xa57d8667
A18=0xffffff1b A19=0xffffdf5
A20=0xfffffff0 A21=0xffffffaa
A22=0xa6 A23=0xffffff60
A24=0xffffea7 A25=0xfffffedf
A26=0x114 A27=0x17
A28=0xffffff34 A29=0x0
A30=0x6 A31=0x0
B0=0x0 B1=0xffffff50
B2=0xffffffd5 B3=0x2
B4=0x0 B5=0x0
B6=0x93 B7=0xffffecb
B8=0x0 B9=0xf00218
B10=0xfffffeee B11=0xffffffa7
```

C66x CPU Exception

Enable “Silicone Real-time” mode:



Enable Silicon Real-time Mode (service critical interrupts when halte

tpr12_xds110.ccxml [Code Composer Studio - Device Debugging]
Group 1
Texas Instruments XDS110 USB Debug Probe_0/CS_DAP_0
Group 2
Texas Instruments XDS110 USB Debug Probe_0/Cortex_R5_0 (Disconnected : Suspended)
Texas Instruments XDS110 USB Debug Probe_0/C66xx_DSP (Disconnected : Suspended)
Non Debuggable Devices
Texas Instruments XDS110 USB Debug Probe_0/Cortex_R5_1 (Disconnected : Unknown)
Texas Instruments XDS110 USB Debug Probe_0/Cortex_M4_0 (Disconnected : Unknown)

Enable Silicon Real-time Mode

Select Core

6.9. Shared memory

AM273X and AWR294X devices don't have a separate shared Handshake RAM (HSRAM) memory. Hence part of the L3 RAM that is to be populated in one core and read in the other needs to be made shareable. For the context of the current demo example, the L3 RAM is made shareable, but for best performance, only the desired portion of the RAM should be made shareable. Refer to the file mmwave_mcuplus_sdk_<ver>/ti/demo/utils/mss_mpu.c and the PDK documentation for more details.

6.10. Size of Enum

If a variable of enum type is used to exchange information between ARM and DSP core, then it is necessary to make sure the enum size matches for the same variable compiled on the two cores. TI ARM compiler's default type for enum is packed, which causes the underlying enumeration type to be the smallest integer type that accommodates the enumeration constants. By default, the TI C6000 (DSP) compiler uses a 32-bit integer to store enum objects with enumeration values smaller than 32 bits. This could cause an enum define that takes values 1 to 4 (for example) to be of size 1 byte on R5F and of size 4 bytes on C66x. For devices where DSP and ARM coexist such as , they must be set to ensure that enum types are consistent between ARM and DSP. In mmWave SDK command makefile, flags R5F_XSFLAGS_ENUM_TYPE and R5F_CFLAGS_ENUM_TYPE are used in conjunction to enforce that enum types are compiled as 32bit integers. It is necessary that all libraries and the application code for a given core are compiled with the same compiler option for enum type else there will be a linker warning and one will encounter errors that cannot be detected until run time.

Linker warning for incompatible enumeration type

```
warning #16027-D: object files have incompatible enumeration types ("xxxx" = packed, "yyyy" = 32-bit)

(xxxx and yyyy will be the names of actual object files that do not have matching enum type)
```

⚠ Please note that the R5F custom application using mmwave SDK pre-built libraries should be compiled with "--enum_type=int" option specified to the compiler.