# MMWAVE SDK User Guide

**Product Release 2.0.0**

**Release Date: Apr 26, 2018**

**Document Version: 1.0**

# 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.*

# CONTENTS

1 Out-of-box mmWave Experience
2 System Overview
- 2.1 mmWave Suite
- 2.2 mmWave Demos
- 2.3 External Dependencies
- 2.4 Terms used in this document
- 2.5 Related documentation/links

3 Getting started
- 3.1 Programming xWR14xx/xWR16xx
- 3.2 Loading images onto xWR14xx/xWR16xx EVM
  - 3.2.1 Demonstration Mode
  - 3.2.2 CCS development mode
- 3.3 Running the Demos
  - 3.3.1 mmWave Demo for xWR14xx/xWR16xx
- 3.4 Configuration (.cfg) File Format
- 3.5 Running the prebuilt unit test binaries (.xer4f and .xe674)

4 How-To Articles
- 4.1 How to identify the COM ports for xWR14xx/xWR16xx EVM
- 4.2 How to flash an image onto xWR14xx/xWR16xx EVM
- 4.3 How to erase flash on xWR14xx/xWR16xx EVM
- 4.4 How to connect xWR14xx/xWR16xx EVM to CCS using JTAG
  - 4.4.1 Emulation Pack Update
  - 4.4.2 Device support package Update
  - 4.4.3 Target Configuration file for CCS (CCXML)
    - 4.4.3.1 Creating a CCXML file
    - 4.4.3.2 Connecting to xWR14xx/xWR16xx EVM using CCXML in CCS
- 4.5 Developing using SDK
  - 4.5.1 Build Instructions
  - 4.5.2 Setting up build environment
    - 4.5.2.1 Windows
    - 4.5.2.2 Linux
  - 4.5.3 Building demo
    - 4.5.3.1 Building demo in Windows
    - 4.5.3.2 Building demo in Linux
  - 4.5.4 Advanced build
    - 4.5.4.1 Building drivers/control/alg components
    - 4.5.4.2 "Error on warning" compiler and linker setting

5 MMWAVE SDK deep dive
- 5.1 System Deployment
  - 5.1.1 xWR14xx
  - 5.1.2 xWR16xx
- 5.2 Typical mmWave Radar Processing Chain
- 5.3 Typical Programming Sequence
  - 5.3.1 Control Path
    - 5.3.1.1 xWR14xx (MSS<->RADARSS)
    - 5.3.1.2 xWR16xx
  - 5.3.2 Data Path
    - 5.3.2.1 xWR14xx
    - 5.3.2.2 xWR16xx
- 5.4 mmWave SDK - TI components
  - 5.4.1 Drivers
  - 5.4.2 OSAL
  - 5.4.3 mmWaveLink
  - 5.4.4 mmWave API
    - 5.4.4.1 Full configuration
    - 5.4.4.2 Minimal configuration
  - 5.4.5 mmWaveLib
  - 5.4.6 Group Tracker
  - 5.4.7 RADARSS Firmware

5.4.8 CCS Debug Utility
5.4.9 HSI Header Utility
5.4.10 mmWave SDK - System Initialization
5.4.10.1 ESM
5.4.10.2 SOC
5.4.10.3 Pinmux
5.4.11 Usecases
5.4.11.1 Data Path tests using Test vector method
5.4.11.2 CSI-2 based streaming of ADC data
5.4.11.3 Basic configuration of Front end and capturing ADC data in L3 memory
6 Appendix
6.1 Memory usage
6.2 Register layout
6.3 Enable DebugP logs
6.4 Shared memory usage by SDK demos (xWR1642)
6.5 xWR1xxx Image Creator
6.5.1 xWR14xx
6.5.2 xWR16xx
6.6 xWR16xx mmw Demo: cryptic message seen on DebugP_assert
6.7 How to execute Idle instruction in idle task when using SYSBIOS
6.8 Range Bias and Rx Channel Gain/Offset Measurement and Compensation
6.9 Guidelines on optimizing memory usage
6.10 How to add a .const (table) beyond L3 heap in mmW demo where overlay is enabled
6.11 DSPlib integration in xWR16xx C674x application (Using 2 libraries simultaneously)
6.11.1 Integrating individual functions from each library
6.11.2 Patching the installation
6.12 SDK Demos: miscellaneous information
6.13 Data size restriction for a given session when sending data over LVDS
6.14 CCS Debugging of real time application
6.14.1 Using non-real time chain test code
6.14.2 Using printfs in real time
6.14.3 Viewing expressions/memory in real time

# LIST OF FIGURES

Figure 1: mmWave Demo Visualizer- mmWave Device Connectivity
- Figure 2: Chirp Diagram
- Figure 3: xWR14xx/xWR16xx PC Connectivity - Device Manager - COM Ports

Figure 4: xWR14xx Deployment in Hybrid or Standalone mode
Figure 5: xWR14xx Deployment in Satellite mode
Figure 6: Autonomous xWR16xx sensor (Standalone mode)

- Figure 7: Typical mmWave radar processing chain
- Figure 8: Typical mmWave radar processing chain using xWR14xx mmWave SDK
- Figure 9: Typical mmWave radar processing chain using xWR16xx mmWave SDK

Figure 10: Typical mmWave radar control flow
Figure 11: xWR14xx: Detailed Control Flow (Init sequence)
Figure 12: xWR14xx: Detailed Control Flow (Config sequence)
Figure 13: xWR14xx: Detailed Control Flow (start sequence)
Figure 14: xWR16xx: Detailed Control Flow (Init sequence)
Figure 15: xWR16xx: Detailed Control Flow (Config sequence)
Figure 16: xWR16xx: Detailed Control Flow (Start sequence)
Figure 17: Typical mmWave radar data flow in xWR14xx
Figure 18: Typical mmWave radar data flow in xWR16xx
Figure 19: mmWave SDK Drivers - Internal software design
Figure 20: mmWaveLink - Internal software design
Figure 21: mmWave API - Internal software design
Figure 22: mmWave API - 'Minimal' Config - Sample flow (xWR16xx)
Figure 23: mmWave API - 'Minimal' Config - Sample flow (xWR14xx)

# LIST OF TABLES

Table 1: mmWave SDK Demos - CLI commands and parameters
Table 2: Supported drivers and their functionality

# 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/mmWaveDemoVisualizer 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
- OSAL
- mmWaveLink
- mmWaveLib
- mmWave API
- RADARSS Firmware
- 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 |
|---|---|
| xWR14xx | This is used throughout the document where that section/component/module applies to both AWR14xx and IWR14xx |
| xWR16xx | This is used throughout the document where that section/component/module applies to both AWR16xx and IWR16xx |
| xWR1xxx | This is used throughout the document where that section/component/module applies to all the part: AWR14xx, IWR14xx, AWR16xx and IWR16xx |
| 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. |
| MSS | Master Sub-system. It is also interchangeably referred to as Cortex R4F. |
| DSS | DSP Sub-system. It is also interchangeably referred to as DSS or C674x core. |

## 2. 5. Related documentation/links

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

- SoC links
  - AWR1443
  - AWR1642
  - IWR1443
  - IWR1642

- EVM links (These pages have links for datasheet and TI EVM user guides that this document refers to)
  - AWR1443BOOST
  - AWR1642BOOST
  - IWR1443BOOST
  - IWR1642BOOST
  - MMWAVE-DEVPACK

# 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_sdk_<ver>/packages/ti/demo/<platform> folder.

**mmWave Demo**

This demo is located at mmwave_sdk_<ver>/packages/ti/demo/<platform>/mmw folder. The millimeter wave demo shows some of the radar sensing and object detection capabilities of the xWR14xx/xWR16xx SoC using the drivers in the mmWave SDK (Software Development Kit). It allows user to specify the chirping profile and displays the detected objects and other information in real-time. A detailed explanation of this demo is available in the demo's docs folder: mmwave_sdk_<ver>\packages\ti\demo\<platform>\mmw\docs\doxygen\html\index.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/mmWaveDemoVisualizer. DS3 LED on TI EVM is turned on when the sensor is started successfully and turned off when the sensor is stopped successfully. SW1 switch press on TI EVM will start/stop the demo (sensor needs to be configured atleast once using the CLI). The version of the mmWave Demo running on TI mmWave EVM can be obtained from the Visualier app using the HelpAbout menu.

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

## 3. 1. Programming xWR14xx/xWR16xx

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.

**xWR14xx**

xWR14xx has one cortex R4F core available for user programming and is referred to in this section as MSS or R4F. The demos and the unit tests executable are provided to be loaded on MSS/R4F.

**xWR16xx**

xWR16xx has one cortex R4F core and one DSP C674x core available for user programming and are referred to as MSS/R4F and DSS/C674X 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 execuables 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 R4F without downloading any code on DSS. Similarly, DSS unit test execuable can be loaded and run on DSS C674x without downloading any code on DSS. The only exception to this is the Mailbox unit test named "test_mss_dss_msg_exchange" and mmWave unit tests under full and minimal.

## 3. 2. Loading images onto xWR14xx/xWR16xx EVM

User can choose either one of these modes for loading images onto the EVM.

### 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 xWR14xx/xWR16xx EVM).
    a. For xWR16xx: use the mmwave_sdk_<ver>/packages/ti/demo/xwr16xx/<demo>/xwr16xx_<demo>.bin as the METAIMAGE1 file name.
    b. For xWR14xx: use the mmwave_sdk_<ver>/packages/ti/demo/xwr14xx/<demo>/xwr14xx_<demo>_mss.bin as the MSS_BUILD file name.
2. Remove the "SOP2" jumper and 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. 2. 2. CCS development mode

This mode should be used when debugging with CCS is involved and/or developing an mmWave application where the .bin 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

**boot-up sequence**
When the xWR1xxx boots up in functional mode, the device bootloader starts executing and checks if a serial flash is attached to the device. If yes, then it expects valid MSS application (and a valid RADARSS firmware and/or DSS application) to be present on the flash. During xWR1xxx development phase, flashing the device with the application under development for every small change can be cumbersome. To overcome this, user should perform a one-time flash as mentioned in the steps below. The actual user application under development can then be loaded and reloaded to the MSS program memory (TCMA) and/or DSP L2/L3 memory (xWR16xx only) directly via CCS in the xWR14xx/xWR16xx functional mode.

Refer to Help inside Code Composer Studio (CCS) to learn more about connecting, loading, running the cores, in general.

1. EVM and CCS setup
    a. Follow the procedure mentioned in the section: How to flash an image onto xWR14xx/xWR16xx EVM.
        i. For xWR16xx: use mmwave_sdk_<ver>/packages/ti/utils/ccsdebug /xwr16xx_ccsdebug.bin as the METAIMAGE1 filename for the one-time flash.
        ii. For xWR14xx: use mmwave_sdk_<ver>/packages/ti/utils/ccsdebug /xwr14xx_ccsdebug_mss.bin as the MSS_BUILD filename for the one-time flash.
    b. Follow the steps in How to connect xWR14xx/xWR16xx EVM to CCS using JTAG to setup the environment for CCS connectivity.
2. With "SOP2" jumper removed, after every power cycle/reboot of the EVM, follow these steps to load the application:
    a. Power up the EVM
    b. Launch ccxml file created in step 1.b above.
    c. If the test requires an application to run on MSS
        i. Connect CCS to Cortex_R4_0
        ii. Load the MSS program. (for example: xwr16xx_<module>_mss.xer4f prebuilt executables provided in the SDK release package)
    d. If the test requires an application to run on DSP (xWR16xx only)
        i. Connect CCS to C674X_0
        ii. Load the DSS program. (for example: xwr16xx_<module>_dss.xe674 prebuilt executables provided in the SDK release package)
    e. Run the R4 and/or C674 cores
    f. To reload, disconnect the connected cores, power cycle and connect 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 you have loaded the right demo binary using the section above, set the EVM to functional mode and powered up the device. Connect the EVM to the PC using its XDS110 micro-USB port/cable.

### 3. 3. 1. mmWave Demo for xWR14xx/xWR16xx

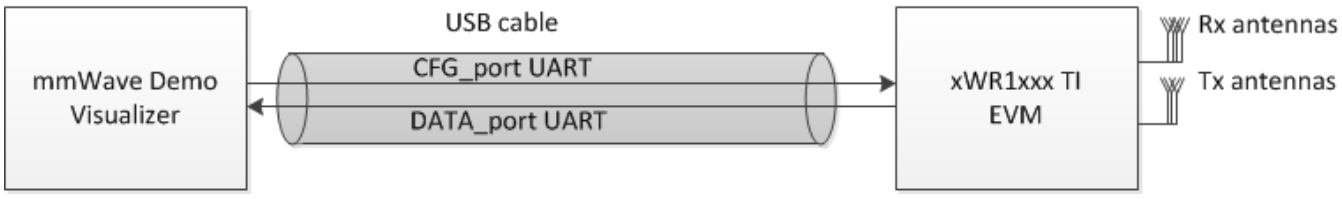


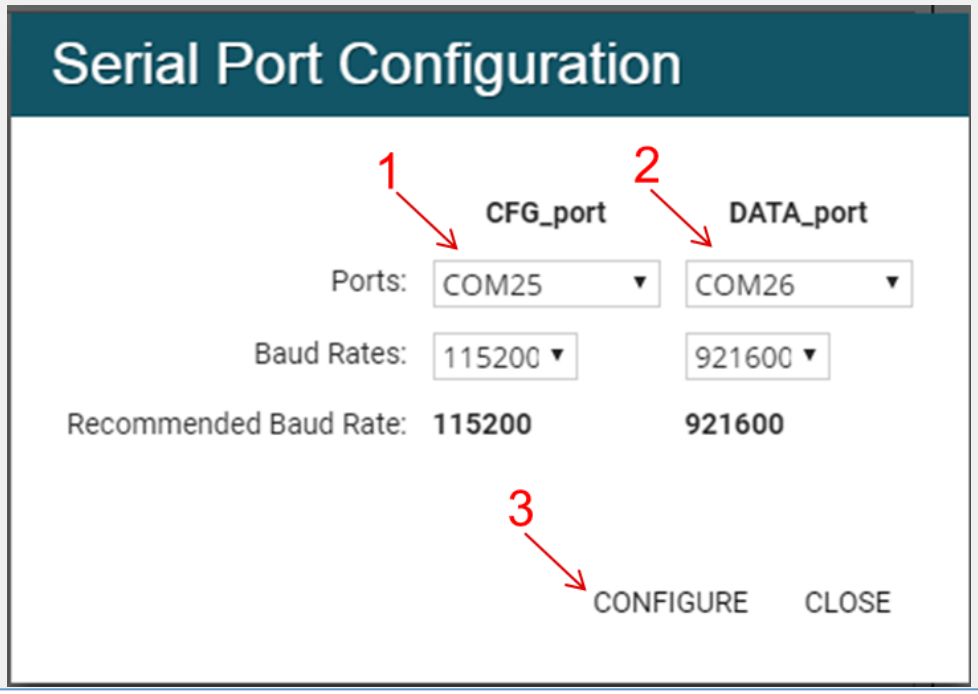

*Figure 1: mmWave Demo Visualizer- mmWave Device Connectivity*

1. Power on the EVM in functional mode with right binary loaded (see section above) and connect it to the PC as shown above with the USB cable.
2. Browse to the TI gallery app "mmWave Demo Visualizer" at http://dev.ti.com/gallery or use the direct link https://dev.ti.com/mmWaveDemoVisualizer. Use HelpREADME.md from inside this app for more information on how to run/configure this app.

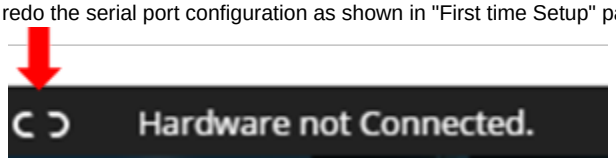
**First Time Setup**

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.
    i. 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 xWR14xx/xWR16xx EVM" below. In the Visualizer App, go to the Menu->Options->Serial Port and perform the settings as shown below.
        - **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: 921600. 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).

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




a. At this point, this app will automatically try to connect to the target (mmWave Sensor). If it 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.




b. After the App is connected to the target, you can select the configuration parameters in this App (Frequency Band, Platform, etc) in the "Scene Selection" and "Object Detection" area of the **CONFIGURE** tab.
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 Platform is changed: make sure the COM ports match the TI EVM/platform you are trying to configure and visualizer
ii. If SDK version is changed: make sure the mmW demo running on the connected TI EVM matches the selected SDK version in the GUI
iii. If Antenna Config is changed: make sure the TI EVM is rebooted before sending the new configuration.

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_sdk_<ver>\packages\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

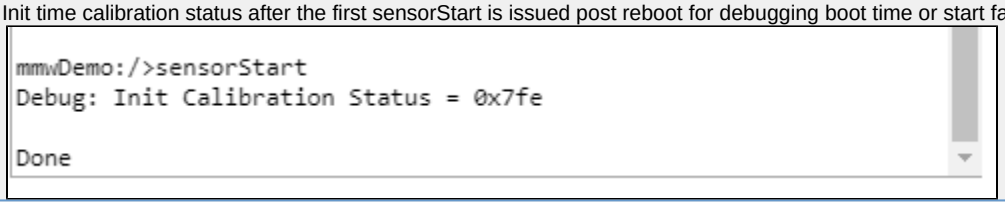
- User can configure the device from their own configuration file or the saved app-generated configuration file by using the "LOAD CONFIG FROM PC AND SEND" button on the **PLOTS** tab. Make sure the first two commands in this config file are "sensorStop" followed by "flushCfg".
- User can temporarily 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).

## Console Messages window in Visualizer

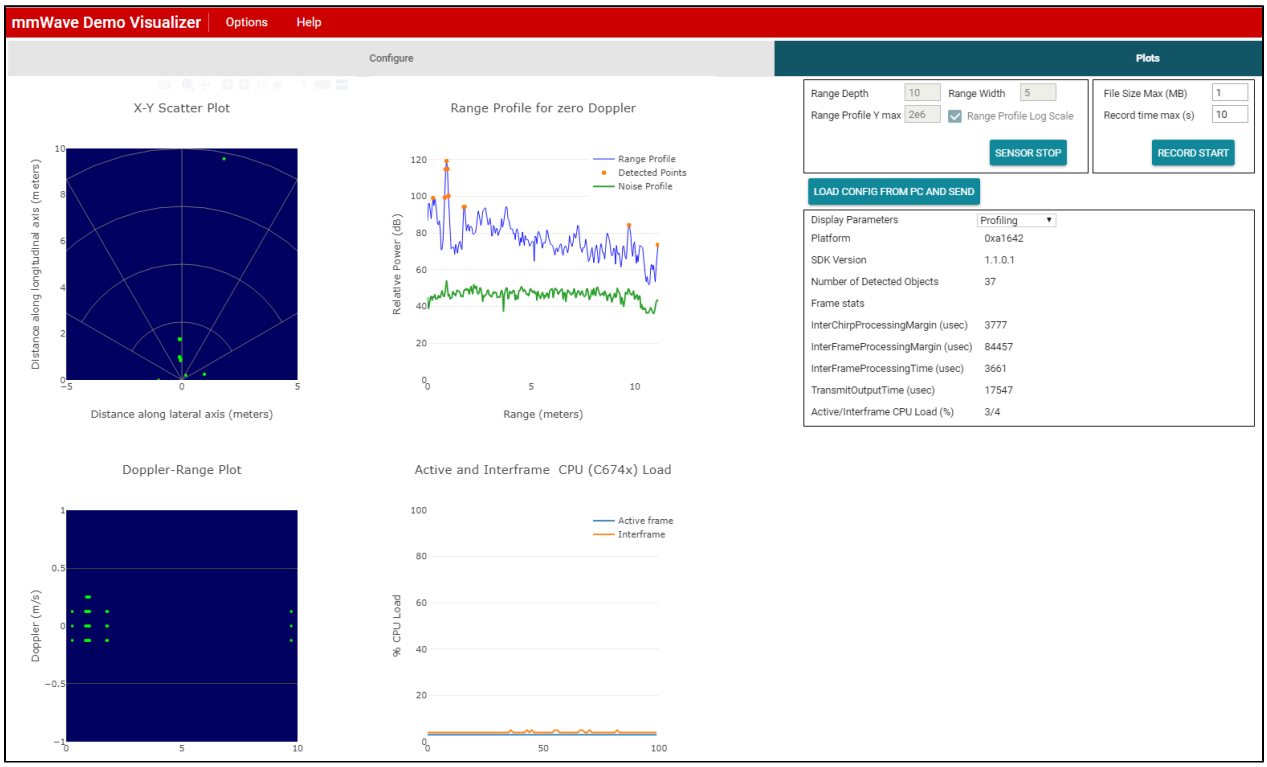
Console message window echoes the following debug information for the users

- 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 ont 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.


- Init time calibration status after the first sensorStart is issued post reboot for debugging boot time or start failures



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.



## 3. 4. Configuration (.cfg) File Format

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). Note that some of the commands (ex: guiMonitor) are available for mmWave Demo only. 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_sdk_<ver>\packages\ti\demo\<platform>\mmw\profiles directory.

**Converting configuration from older SDK release to current SDK release**
As new versions of SDK releases are available, there are usually changes to the configuration commands that are supported in the new release. Now, users may have some hand crafted config file which worked perfectly well on older SDK release version but will not work as is with the new SDK release. If user desires to run the same configuration against the new SDK release, then there is a script mmwDemo_<platform>_update_config.pl provided in the mmwave_sdk_<ver>\packages\ti\demo\<platform>\mmw\profiles dir ectory that they can use to convert the configuration file from older release to a compatible version for the new release. Refer to the perl file for details on how to run the script. Note that users will need to install perl on their machine (there is no strict version requirement at this time). For any new commands inserted by the script, there will be a comment preceeding that line which is similar to something like this: "Inserting new mandatory command. Check users guide for details."

Most of the parameters described below are the same as the mmwavelink API specifications (see doxygen mmwave_sdk_<ver>\packages\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.

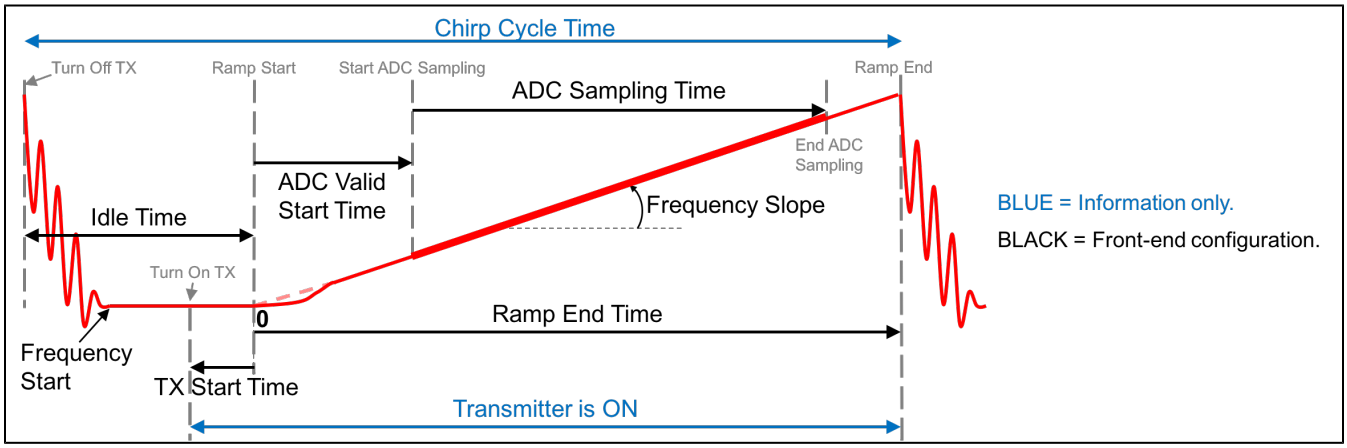



***Figure 2: Chirp Diagram***

| Configuration command | Command details | Command Parameters | Usage in mmW demo xwr14xx | Usage in mmW demo xwr16xx |
|---|---|---|---|---|
| dfeDataOutputMode | The values in this command should not change between sensorStop and sensorStart.<br><br>Reboot the board to try config with different set of values in this command<br><br>This is a mandatory command. | | | |
| | | <modeType><br>1 - frame based chirps<br>2 - continuous chirping<br>3 - advanced frame config | only option 1 is supported | only option 1 and 3 are suppeorted |
| | | | | |
| channelCfg | Channel config message to RadarSS. See mmwavelink doxgen for details.<br><br>The values in this command should not change between sensorStop and sensorStart.<br><br>Reboot the board to try config with different set of values in this command<br><br>This is a mandatory command. | | | |
| | | <rxChannelEn><br>Receive antenna mask e.g for 4 antennas, it is 0x1111b = 15 | 4 antennas supported | 4 antennas supported |
| | | <txChannelEn><br>Transmit antenna mask | The 2 azimuth antennas can be enabled using bitmask 0x5 (i.e. tx1 and tx3)<br><br>The azimuth and elevation antennas can be enabled using bitmask 0x7 (i.e. tx1, tx2 and tx3) | The 2 azimuth antennas can be enabled using bitmask 0x3 (i.e. tx1 and tx2) |

| | | | | |
|---|---|---|---|---|
| | | <cascading><br>SoC cascading, not applicable, set to 0 | n/a | n/a |
| | | | | |
| adcCfg | ADC config message to RadarSS. See mmwavelink doxgen for details.<br><br>The values in this command should not change between sensorStop and sensorStart.<br><br>Reboot the board to try config with different set of values in this command<br><br>This is a mandatory command. | | | |
| | | <numADCBits><br>Number of ADC bits (0 for 12-bits, 1 for 14-bits and 2 for 16-bits) | only 16-bit is supported | only 16-bit is supported |
| | | <adcOutputFmt><br>Output format :<br>0 - real<br>1 - complex 1x (image band filtered output)<br>2 - complex 2x (image band visible)) | only complex modes are supported | only complex modes are supported |
| | | | | |
| adcbufCfg | adcBuf hardware config. The values in this command can be changed between sensorStop and sensorStart.<br><br>This is a mandatory command. | | | |
| | | <subFrameIdx><br>subframe Index (exists only in xwr16xx mmW demo) | doesn't exist | For legacy mode, that field should be set to -1.<br><br>For advanced frame mode, it should be set to either the intended subframe number or -1 to apply same config to all subframes. |
| | | <adcOutputFmt><br>ADCBUF out format<br>0-Complex,<br>1-Real | only complex modes are supported | only complex modes are supported |
| | | <SampleSwap><br>ADCBUF IQ swap selection:<br>0-I in LSB, Q in MSB,<br>1-Q in LSB, I in MSB | only option 0 is supported | only option 0 is supported |
| | | <ChanInterleave><br>ADCBUF channel interleave configuration:<br>0 - interleaved(not supported on XWR16xx),<br>1 - non-interleaved | only option 0 is supported | only option 1 is supported |
| | | <ChirpThreshold><br>Chirp Threshold configuration used for ADCBUF buffer to trigger ping/pong buffer switch.<br><br>Valid values:<br><br>0-8 for xWR16xx (conditions apply, see description in "Usage in mmW demo xwr16xx" column)<br><br>only 1 for xWR14xx | only value of 1 is supported | 0: Determine chirp threshold automatically to either maximize 32 KB usage of ADCBUF memory or cap to value of 8, while meeting requirement that the threshold must divide the number of chirps in a frame.<br><br>n (i.e. any values in the range 1-8): Set to n unless..<br>1. n exceeds what is determined to be possible maximum based on 0 interpretation above.<br><br>In such case, the demo app will cap the value to the max with a diagnostic warning message that threshold has been capped to max.<br>2. n does not meet divisibility requirement stated above and in this case, demo app will throw an assert.<br><br>Note: n = 1 is a valid value. |
| | | | | |
| profileCfg | Profile config message to RadarSS and datapath. See mmwavelink doxgen for details.<br><br>The values in this command can be changed between sensorStop and sensorStart.<br><br>This is a mandatory command. | | | |

| | | |
|---|---|---|
| <profileId><br>profile Identifier | could be any allowed value but only one valid profile per config is supported | Legacy frame (dfeOutputMode=1): could be any allowed value but only one valid profile per config is supported<br><br>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><br>"Frequency Start" in GHz (float values allowed)<br><br>Examples:<br><br>77<br><br>78.1 | any value as per mmwavelink doxgen/device datasheet but represented in GHz | any value as per mmwavelink doxgen/device datasheet but represented in GHz |
| <idleTime><br>"Idle Time" in u-sec (float values allowed)<br><br>Examples:<br><br>7<br><br>7.15 | any value as per mmwavelink doxgen/device datasheet but represented in usec | any value as per mmwavelink doxgen/device datasheet but represented in usec |
| <adcStartTime><br>"ADC Valid Start Time" in u-sec (float values allowed)<br><br>Examples:<br><br>7<br><br>7.34 | any value as per mmwavelink doxgen/device datasheet but represented in usec | any value as per mmwavelink doxgen/device datasheet but represented in usec |
| <rampEndTime><br>"Ramp End Time" in u-sec (float values allowed)<br><br>Examples:<br><br>58<br>216.15 | any value as per mmwavelink doxgen/device datasheet but represented in usec | any value as per mmwavelink doxgen/device datasheet but represented in usec |
| <txOutPower><br>Tx output power back-off code for tx antennas | only value of '0' has been tested | only value of '0' has been tested |
| <txPhaseShifter><br>tx phase shifter for tx antennas | only value of '0' has been tested | only value of '0' has been tested |
| <freqSlopeConst><br>"Frequency slope" for the chirp in MHz/usec (float values allowed)<br><br>Examples:<br><br>68<br><br>16.83 | any value greater than 0 as per mmwavelink doxgen/device datasheet but represented in MHz/usec | any value greater than 0 as per mmwavelink doxgen/device datasheet but represented in MHz/usec |
| <txStartTime><br>"TX Start Time" in u-sec (float values allowed)<br><br>Examples:<br><br>1<br>2.92 | any value as per mmwavelink doxgen/device datasheet but represented in usec | any value as per mmwavelink doxgen/device datasheet but represented in usec |
| <numAdcSamples><br>number of ADC samples collected during "ADC Sampling Time" as shown in the chirp diagram above<br><br>Examples:<br><br>256<br><br>224 | any value as per mmwavelink doxgen/device datasheet | any value as per mmwavelink doxgen/device datasheet but with multiplicity of 4 required due to windowing library function requirement and ADCBuf channel offset requirement (since samples are complex) |

| | | | | |
|---|---|---|---|---|
| | | <digOutSampleRate><br>ADC sampling frequency in ksps.<br><br>(<numAdcSamples> / <digOutSampleRate> = "ADC Sampling Time")<br><br>Examples:<br><br>5500 | any value as per mmwavelink doxgen/device datasheet | any value as per mmwavelink doxgen/device datasheet |
| | | <hpfCornerFreq1><br>HPF1 (High Pass Filter 1) corner frequency<br>0: 175 KHz<br>1: 235 KHz<br>2: 350 KHz<br>3: 700 KHz | any value as per mmwavelink doxgen/device datasheet | any value as per mmwavelink doxgen/device datasheet |
| | | <hpfCornerFreq2><br>HPF2 (High Pass Filter 2) corner frequency<br>0: 350 KHz<br>1: 700 KHz<br>2: 1.4 MHz<br>3: 2.8 MHz | any value as per mmwavelink doxgen/device datasheet | any value as per mmwavelink doxgen/device datasheet |
| | | <rxGain><br>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 | any value as per mmwavelink doxgen/device datasheet |
| | | | | |
| chirpCfg | Chirp config message to RadarSS and datapath. See mmwavelink doxgen for details.<br><br>The values in this command can be changed between sensorStop and sensorStart.<br><br>This is a mandatory command. | | | |
| | | chirp start index | any value as per mmwavelink doxgen | any value as per mmwavelink doxgen |
| | | chirp end index | any value as per mmwavelink doxgen | any value as per mmwavelink doxgen |
| | | profile identifier | should match the profileCfg->profileId | should match the profileCfg->profileId |
| | | start frequency variation in Hz (float values allowed) | only value of '0' has been tested | only value of '0' has been tested |
| | | frequency slope variation in kHz/us (float values allowed) | only value of '0' has been tested | only value of '0' has been tested |
| | | idle time variation in u-sec (float values allowed) | only value of '0' has been tested | only value of '0' has been tested |
| | | ADC start time variation in u-sec (float values allowed) | only value of '0' has been tested | only value of '0' has been tested |
| | | tx antenna enable mask (Tx2,Tx1) e.g (10)b = Tx2 enabled, Tx1 disabled. | See note under "Channel Cfg" command above. Individual chirps should have either only one distinct Tx antenna enabled (MIMO) or same TX antennas should be enabled for all chirps | See note under "Channel Cfg" command above. Individual chirps should have either only one distinct Tx antenna enabled (MIMO) or same TX antennas should be enabled for all chirps |
| | | | | |
| | | | | |
| bpmCfg | BPM MIMO configuration.<br><br>Every frame consists of alternating chirps with pattern TX1_Tx2 and TX1-TX2. This is alternate configuration to TDM-MIMO scheme and provides SNR improvement by running 2Tx simultaneously.When using this scheme, user should enable both the azimuth TX in the chirpCfg. See profile_2d_bpm.cfg profile in the xwr16xx mmW demo profiles directory for example usage.<br><br>This config is supported only for xWR16xx | | | |
| | | <subFrameIdx><br>subframe Index (exists only in xwr16xx mmW demo) | n/a | 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><br><br>0-Disabled<br><br>1-Enabled | n/a | all values supported |

| | | | | |
|---|---|---|---|---|
| | | <chirp0Idx><br><br>BPM enabled:<br>If BPM is enabled in previous argument, this is the chirp index for the first BPM chirp.<br>It will have phase 0 on both TX antennas (TX0+ , TX1+). Note that the chirpCfg command for this chirp index must have both TX antennas enabled.<br><br>BPM disabled:<br>If BPM is disabled, a BPM disable command (set phase to zero on both TX antennas) will be issued for the chirps in the range [chirp0Idx..chirp1Idx] | n/a | value should have a corresponding valid chirpCfg |
| | | <chirp1Idx><br><br>BPM enabled:<br>If BPM is enabled, this is the chirp index for the second BPM chirp.<br>It will have phase 0 on TX0 and phase 180 on TX1 (TX0+ , TX1-). Note that the chirpCfg command for this chirp index must have both TX antennas enabled.<br><br>BPM disabled:<br>If BPM is disabled, a BPM disable command (set phase to zero on both TX antennas) will be issued for the chirps in the range [chirp0Idx..chirp1Idx]. | | value should have a corresponding valid chirpCfg |
| | | | | |
| lowPower | Low Power mode config message to RadarSS. See mmwavelink doxgen for details.<br><br>The values in this command should not change between sensorStop and sensorStart.<br><br>Reboot the board to try config with different set of values in this command.<br><br>This is a mandatory command. | | | |
| | | <don't_care> | set to 0 | set to 0 |
| | | ADC Mode<br>0x00 : Regular ADC mode<br>0x01 : Low power ADC mode | only value of '0' has been tested | only value of '1' is supported |
| | | | | |
| frameCfg | frame config message to RadarSS and datapath. See mmwavelink doxgen for details.<br><br>dfeOutputMode should be set to 1 to use this command<br><br>The values in this command can be changed between sensorStop and sensorStart.<br><br>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 | 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 | 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 should be a power of 2 | any value as per mmwavelink doxgen/device datasheet but should be a power of 2 and should have a minimum value of 16 due to DSP Library requirements |
| | | number of frames (valid range is 0 to 65535, 0 means infinite) | any value as per mmwavelink doxgen | any value as per mmwavelink doxgen |
| | | frame periodicity in ms (float values allowed) | any value as per mmwavelink doxgen and represented in msec. However frame should atleast have 50% duty cycle and allow enough time for selected UART output to be shipped out (selections based on guiMonitor command) | any value as per mmwavelink doxgen and represented in msec. However frame should atleast have 50% duty cycle and allow enough time for selected UART output to be shipped out (selections based on guiMonitor command) |
| | | | | |

| | | | | |
|---|---|---|---|---|
| | | trigger select<br>1: Software trigger<br>2: Hardware trigger. | only option for Software trigger is selected | only option for Software trigger is selected |
| | | Frame trigger delay in ms (float values allowed) | any value as per mmwavelink doxgen and represented in msec. | 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. See profile_advanced_subframe.cfg profile in the xwr16xx mmW demo profiles directory for example usage.<br><br>The values in this command can be changed between sensorStop and sensorStart.<br><br>This is a mandatory command when dfeOutputMode is set to 3. | | command not suported | |
| | | <numOfSubFrames><br>Number of sub frames enabled in this frame | n/a | any value as per mmwavelink doxgen |
| | | <forceProfile><br>Force profile | n/a | only value of 0 is supported |
| | | <numFrames><br>Number of frames to transmit (1 frame = all enabled sub frames) | n/a | any value as per mmwavelink doxgen |
| | | <triggerSelect><br>trigger select<br>1: Software trigger<br>2: Hardware trigger. | n/a | only option for Software trigger is selected |
| | | <frameTrigDelay><br>Frame trigger delay in ms (float values allowed) | n/a | any value as per mmwavelink doxgen and represented in msec. |
| | | | | |
| subFrameCfg | Subframe config message to RadarSS and datapath. See mmwavelink doxgen for details.<br><br>The dfeOutputMode should be set to 3 to use this command. See profile_advanced_subframe.cfg profile in the xwr16xx mmW demo profiles directory for example usage<br><br>The values in this command can be changed between sensorStop and sensorStart.<br><br>This is a mandatory command when dfeOutputMode is set to 3. | | command not suported | |
| | | <subFrameNum><br>subframe Number for which this command is being given | n/a | value of 0 to RL_MAX_SUBFRAMES-1 |
| | | <forceProfileIdx><br>Force profile index | n/a | ignored as <forceProfile> in advFrameCfg should be set to 0 |
| | | <chirpStartIdx><br>Start Index of Chirp | n/a | any value as per mmwavelink doxgen but corresponding chirpCfg should be defined |
| | | <numOfChirps><br>Num of unique Chirps per burst including start index | n/a | any value as per mmwavelink doxgen but corresponding number of chirpCfg should be defined |
| | | <numLoops><br>No. of times to loop through the unique chirps | n/a | any value as per mmwavelink doxgen but corresponding chirpCfg should be defined |
| | | <burstPeriodicity><br><br>Burst periodicty in msec (float values allowed) and meets the criteria<br>burstPeriodicity >= (numLoops)* (numOfChirps) + InterBurstBlankTime | n/a | any value as per mmwavelink doxgen and represented in msec but subframe should atleast have 50% duty cycle and allow enough time for selected UART output to be shipped out (selections based on guiMonitor command) |
| | | <chirpStartIdxOffset><br>Chirp Start address increament for next burst | n/a | set it to 0 since demo supports only one burst per subframe |
| | | <numOfBurst><br>Num of bursts in the subframe | n/a | set it to 1 since demo supports only one burst per subframe |
| | | <numOfBurstLoops><br>Number of times to loop over the set of above defined bursts, in the sub frame | n/a | set it to 1 since demo supports only one burst per subframe |
| | | <subFramePeriodicity><br><br>subFrame periodicty in msec (float values allowed) and meets the criteria<br>subFramePeriodicity >= Sum total time of all bursts + InterSubFrameBlankTime | n/a | set to same as <burstPeriodicity> since demo supports only one burst per subframe |
| | | | | |
| guiMonitor | Plot config message to datapath.<br>The values in this command can be changed between sensorStop and sensorStart. | | | |
| | | All parameters below are flags (1 to enable and 0 to disable) | | |

| | | | | |
|---|---|---|---|---|
| | This is a mandatory command. | <subFrameIdx><br>subframe Index (exists only in xwr16xx mmW demo) | doesn't exist | 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><br>1 - enable export of detected objects<br>0 - disable | all values supported | all values supported |
| | | <log magnitude range><br>1 - enable export of log magnitude range profile at zero Doppler<br>0 - disable | all values supported | all values supported |
| | | <noise profile><br>1 - enable export of log magnitude noise profile<br>0 - disable | all values supported | all values supported |
| | | <rangeAzimuthHeatMap><br>range-azimuth heat map related information<br>1 - enable export of zero Doppler radar cube matrix, all range bins, all antennas to calculate and display azimuth heat map.<br>0 - disable<br>(the GUI computes the FFT of this to show heat map) | all values supported | all values supported |
| | | <rangeDopplerHeatMap><br>range-doppler heat map<br>1 - enable export of the whole detection matrix. Note that the frame period should be adjusted according to UART transfer time.<br>0 - disable | all values supported | all values supported |
| | | <statsInfo><br>statistics (CPU load, margins, etc)<br>1 - enable export of stats data.<br>0 - disable | all values supported | all values supported |
| | | | | |
| cfarCfg | CFAR config message to datapath.<br><br>The values in this command can be changed between sensorStop and sensorStart and even when the sensor is running.<br><br>This is a mandatory command. | | | |
| | | <subFrameIdx><br>subframe Index (exists only in xwr16xx mmW demo) | doesn't exist | 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><br>Processing direction:<br>0 – CFAR detection in range direction<br>1 – CFAR detection in Doppler direction | only Range direction is supported | all values supported; 2 separate commands need to be sent; one for Range and other for doppler |
| | | <mode><br>CFAR averaging mode:<br>0 - CFAR_CA (Cell Averaging)<br>1 - CFAR_CAGO (Cell Averaging Greatest Of)<br>2 - CFAR_CASO (Cell Averaging Smallest Of) | all values supported | all values supported |
| | | <noiseWin><br>noise averaging window length:<br>Length of the noise averaged cells in samples | supported | supported |
| | | <guardLen><br>guard length in samples | supported | supported |

| | | | | |
|---|---|---|---|---|
| | | <divShift><br>Cumulative noise sum divisor expressed as a shift.<br><br>Sum of noise samples is divided by 2^<divShift>. Based on platform, <mode> and <noiseWin> , this value should be set as shown in next columns.<br><br>The value to be used here should match the "CFAR averaging mode" and the "noise averaging window length" that is selected above.<br><br>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:<br><divShift> = $\log_2$(2 x <noiseWin>)<br>CFAR_CAGO/_CASO:<br><divShift> = $\log_2$(<noiseWin>)<br><br>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. | CFAR_CA:<br><divShift> = $\log_2$(2 x <noiseWin>)<br>CFAR_CAGO/_CASO:<br><divShift> = $\log_2$(2 x <noiseWin>)<br><br>In profile_2d.cfg, value of 4 means that the noise sum is divided by 2^4=16 to get the average of noise samples with window length of 8 samples in CFAR -CA mode. |
| | | cyclic mode or Wrapped around mode.<br>0- Disabled<br>1- Enabled | used for programming the CFAR engine inside HWA | This control is not supported on xWR16xx, where it is always enabled in CFAR detection in Doppler direction and always disabled in CFAR detection in range direction. |
| | | Threshold scale.<br>This is used in conjuntion with the noise sum divisor (say x).<br>the CUT comparison for log input is:<br><br>CUT > Threshold scale + (noise sum / 2^x) | Detection threshold is specified as log2 value, expressed in Q9 format for xWR14xx. The threshold value can be converted from the value expressed in dB as<br><br>Tcli = 512 x TdB / 6 x N / N'<br><br>where N is numVirtualAntennas<br><br>N' = 2^ceil(log2(N)).<br><br>Note: log input is used for xWR14xx mmw demo | Detection threshold is specified as log2 value, expressed in Q8 format for xWR16xx. The threshold value can be converted from the value expressed in dB as<br><br>Tcli = 256 x numVirtualAntennas x TdB / 6.<br><br>Note: log input is used for xWR16xx mmw demo |
| | | | | |
| peakGrouping | Peak grouping config message to datapath.<br><br>With peak grouping scheme enabled, instead of reporting a cluster of detected neighboring points, only one point, the highest one, will be reported, this reducing the total number of detected points per frame. Only the points between start and end range index are considered. Detected points falling outside this range are dropped and not shipped out as part of point cloud.<br><br>The values in this command can be changed between sensorStop and sensorStart and even when the sensor is running.<br><br>This is a mandatory command. | | | |
| | | <subFrameIdx><br>subframe Index (exists only in xwr16xx mmW demo) | doesn't exist | 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. |
| | | <scheme><br>1 – MMW_PEAK_GROUPING_DET_MATRIX_BASED<br>Peak grouping is based on peaks of the neighboring bins read from detection matrix. CFAR detected peak is reported if it is greater than its neighbors, located in detection matrix.<br>2 – MMW_PEAK_GROUPING_CFAR_PEAK_BASED<br>Peak grouping is based on peaks of neighboring bins that are CFAR detected. CFAR detected peak is reported if it is greater than its neighbors, located in the list of CFAR detected peaks.<br><br>For more detailed look at mmw demo's doxygen documentation. | only option 1 is supported | both options are supported |
| | | peak grouping in Range direction:<br>0 - disabled<br>1 - enabled | supported | supported |
| | | peak grouping in Doppler direction:<br>0 - disabled<br>1 - enabled | supported | supported |
| | | | | |

| | | | | |
|---|---|---|---|---|
| | | Start Range Index<br>Minimum range index of detected object that should be sent out.<br><br>Ex: Value of 1 means Skip 0th bin and start peak grouping from range bin#1 | supported | supported |
| | | End Range Index<br>Maximum range index of detected object that should be sent out.<br><br>Ex: Value of (Range FFT size -1) means skip last bin and stop peak grouping at (Range FFT size -1) | supported | supported |
| | | | | |
| multiObjBeamForming | Multi Object Beamforming config message to datapath.<br><br>This feature allows radar to separate reflections from multiple objects originating from the same range/Doppler detection.<br><br>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.<br><br>The values in this command can be changed between sensorStop and sensorStart and even when the sensor is running.<br><br>This is a mandatory command. | | | |
| | | <subFrameIdx><br>subframe Index (exists only in xwr16xx mmW demo) | doesn't exist | 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. |
| | | <Feature Enabled><br>0 - disabled<br>1 - enabled | supported | supported |
| | | <threshold><br>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 | supported |
| | | | | |
| calibDcRangeSig | DC range calibration config message to datapath.<br><br>Antenna coupling signature dominates the range bins close to the radar. These are the bins in the range FFT output located around DC.<br><br>When this feature is enabled, the signature is estimated during the first N chirps, and then it is subtracted during the subsequent chirps.<br><br>During the estimation period the specified bins around DC are accumulated and averaged. It is assumed that no objects are present in the vicinity of the radar at that time.<br><br>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.<br><br>The values in this command can be changed between sensorStop and sensorStart and even when the sensor is running.<br><br>This is a mandatory command. | | | |
| | | <subFrameIdx><br>subframe Index (exists only in xwr16xx mmW demo) | doesn't exist | 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><br>Enable DC removal using first few chirps<br>0 - disabled<br>1 - enabled | supported | supported |
| | | <negativeBinIdx><br>negative Bin Index (to remove DC from farthest range bins)<br><br>Maximum negative range FFT index to be included for compensation. Negative indices are indices wrapped around from far end of 1D FFT.<br><br>Ex: Value of -5 means last 5 bins | supported | supported |
| | | <positiveBinIdx><br>positive Bin Index (to remove DC from closest range bins)<br>Maximum positive range FFT index to be included for compensation<br><br>Value of 8 means first 8 bins | supported | supported |

| | | | | |
|---|---|---|---|---|
| | | <numAvg><br>number of chirps to average to collect DC signature (which will then be applied to all chirps beyond this).<br>The value must be power of 2, and also in xWR14xx, it must be greater than the number of Doppler bins.<br><br>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. | The value must be power of 2 |
| | | | | |
| extendedMaxVelocity | Velocity disambiguation config message to datapath.<br><br>A simple technique for velocity disambiguation is implemented. It corrects target velocities up to (2*vmax). Enabling this feature results in loss of multiObjBeamForming feature.<br><br>See mmW demo doxygen for xwr16xx for more details.<br><br>The values in this command can be changed between sensorStop and sensorStart and even when the sensor is running.<br><br>This is a mandatory command. | | command doesn't exist | |
| | | <subFrameIdx><br>subframe Index (exists only in xwr16xx mmW demo) | n/a | 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><br>Enable velocity disambiguation technique<br>0 - disabled<br>1 - enabled | n/a | |
| | | | | |
| clutterRemoval | Static clutter removal config message to datapath.<br><br>Static clutter removal algorithm implemented by subtracting from the samples the mean value of the input samples to the 2D-FFT<br><br>The values in this command can be changed between sensorStop and sensorStart and even when the sensor is running.<br><br>This is a mandatory command. | | | |
| | | <enabled><br>Enable static clutter removal technique<br>0 - disabled<br>1 - enabled | supported | supported |
| | | | | |
| compRangeBias AndRxChanPhase | Command for datapath to compensate for bias in the range estimation and receive channel gain and phase imperfections.<br>Refer to the procedure mentioned here<br><br>The values in this command can be changed between sensorStop and sensorStart and even when the sensor is running.<br><br>This is a mandatory command. | | | |
| | | <rangeBias><br>Compensation for range estimation bias in meters | supported | 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)><br><br>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 | 12 pairs of values should be provided here since the device has 4 Rx and 3 Tx (total of 12 virtual antennas) | 8 pairs of values should be provided here since the device has 4 Rx and 2 Tx (total of 8 virtual antennas) |
| | | | | |
| measureRangeBias AndRxChanPhase | Command for datapath to enable the measurement of the range bias and receive channel gain and phase imperfections. Refer to the procedure mentioned here<br><br>The values in this command can be changed between sensorStop and sensorStart and even when the sensor is running.<br><br>This is a mandatory command. | | | |
| | | <enabled><br>1 - enable measurement. This parameter should be enabled only using the profile_calibration.cfg profile in the mmW demo profiles directory<br>0 - disable measurement. This should be the value to use for all other profiles. | supported | supported |

| | | | | |
|---|---|---|---|---|
| | | <targetDistance><br>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 | supported |
| | | <searchWin><br>distance in meters of the search window around <targetDistance> where the peak will be searched | supported | supported |
| | | | | |
| nearFieldCfg | OOB processing chain assumes that the object of interests are located in the far field so that the rays between the object and the multiple TX/RX antennas are parallel. However for very close by objects this assumption (of parallel lines) is not valid and can induce a significant phase error when processed using regular FFT techniques. User can use this command to enable the near field correction algorithm.<br><br>See mmW demo doxygen for xwr16xx for more details. | | | |
| | | <subFrameIdx><br><br>subframe Index (exists only in xwr16xx mmW demo) | n/a | 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><br><br>Enable near field correction<br><br>0 - disabled<br><br>1 - enabled | n/a | supported |
| | | <startRangeIndex><br><br>This is the first range bin index at which the algorithm would start correcting | n/a | supported |
| | | <endRangeIndex><br><br>This is the last range bin index beyond which the algorithm would stop correcting. | n/a | supported |
| | | | | |
| CQRxSatMonitor | Rx Saturation Monitoring config message for Chirp quality to RadarSS and datapath. See mmwavelink doxgen for details on rlRxSatMonConf_t. | | | |
| | | <profile><br><br>Valid profile Id for this monitoring configuraiton.<br><br>This profile ID should have a matching profileCfg | any value as per mmwavelink doxgen | any value as per mmwavelink doxgen |
| | | <satMonSel><br><br>RX Saturation monitoring mode | any value as per mmwavelink doxgen | any value as per mmwavelink doxgen |
| | | <priSliceDuration><br><br>Duration of each slice, 1LSB=0.16us, range: 4 - number of ADC samples | any value as per mmwavelink doxgen | any value as per mmwavelink doxgen |
| | | <numSlices><br><br>primary + secondary slices , range 1-127. Maximum primary slice is 64. | any value as per mmwavelink doxgen | any value as per mmwavelink doxgen |
| | | <rxChanMask><br><br>RX channgel mask, 1 - Mask, 0 - unmask | any value as per mmwavelink doxgen | any value as per mmwavelink doxgen |
| | | | | |
| CQSigImgMonitor | Signal and image band energy Monitoring config message for Chirp quality to RadarSS and datapath. See mmwavelink doxgen for details on rlSigImgMonConf_t.<br><br>The enable/disable for this command is controlled via the "analogMonitor" CLI command | | | |
| | | <profile><br><br>Valid profile Id for this monitoring configuraiton.<br><br>This profile ID should have a matching profileCfg | any value as per mmwavelink doxgen | any value as per mmwavelink doxgen |

| | | | | |
|---|---|---|---|---|
| | | <numSlices><br><br>primary + secondary slices , range 1-127. Maximum primary slice is 64. | any value as per mmwavelink doxgen | any value as per mmwavelink doxgen |
| | | <numSamplePerSlice><br><br>Possible range is 4 to "number of ADC samples" in the corresponding profileCfg. It must be an even number. | any value as per mmwavelink doxgen | any value as per mmwavelink doxgen |
| | | | | |
| analogMonitor | Controls the enable/disable of the various monitoring features supported in the demos. | | | |
| | | <rxSaturation><br><br>CQRxSatMonitor enable/disable<br><br>1:enable<br><br>0: disable | all values supported | all values supported |
| | | <sigImgBand><br><br>CQSigImgMonitor enable/disable<br><br>1:enable<br><br>0: disable | all values supported | all values supported |
| | | | | |
| lvdsStreamCfg | Enables the streaming of various data streams over LVDS lanes (xWR16xx). | | | |
| | | <subFrameIdx><br><br>subframe Index (exists only in xwr16xx mmW demo) | n/a | 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><br><br>0 - Disable HSI header for all active streams<br><br>1 - Enable HSI header for all active streams | n/a | all values supported |
| | | <dataFmt><br><br>Controls HW streaming. Specifies the HW streaming data format.<br><br>0-HW STREAMING DISABLED<br><br>1-ADC<br><br>2-CP_ADC<br><br>3-ADC_CP<br><br>4-CP_ADC_CQ | n/a | all values supported |
| | | <enableSW><br><br>0 - Disable user data (SW session)<br><br>1 - Enable user data | n/a | all values supported |
| | | | | |
| sensorStart | sensor Start command to RadarSS and datapath.<br>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.<br><br>This is a mandatory command. | | | |
| | | Optionally, user can provide an argument 'doReconfig'<br>1 - Do full reconfiguration of the device<br>0 - Skip reconfiguration and just start the sensor using already provided configuration. | supported | supported |
| | | | | |

| | | | | |
|---|---|---|---|---|
| sensorStop | sensor Stop command to RadarSS and datapath.<br>Stops the sensor.<br>If the sensor is running, it will stop the mmWave Front End and the processing chain.<br>After the command is acknowledged, a new 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.<br><br>This is mandatory before any reconfiguration is performed post sensorStart. | | supported | supported |
| | | | | |
| flushCfg | This command should be issued after 'sensorStop' command to flush the old configuration and provide a new one.<br><br>This is mandatory before any reconfiguration is performed post sensorStart. | | supported | supported |
| | | | | |
| % | | Any line starting with '%' character is considered as comment line and is skipped by the CLI parsing utility. | supported | supported |

***Table 1: mmWave SDK Demos - CLI commands and parameters***

## 3. 5. Running the prebuilt unit test binaries (.xer4f and .xe674)

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, UART test code that runs on TI RTOS is in mmwave_sdk_<ver>/packages/ti/drivers/uart/test/<platform>. In this test directory, you will find .xer4f and .xe674 files (either prebuilt or build as a part of instructions mentioned in "Building drivers/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 xWR14xx/xWR16xx 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/mmWaveDemoVisualizer) and follow the one-time setup instructions.
2. If you are trying to flash the board, using Uniflash tool and following the cloud or desktop version installation instructions would also install the right drivers for the COM ports.
3. If above methods didnt work and if TI code composer studio is not installed on that PC, then download and install the stand alone 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

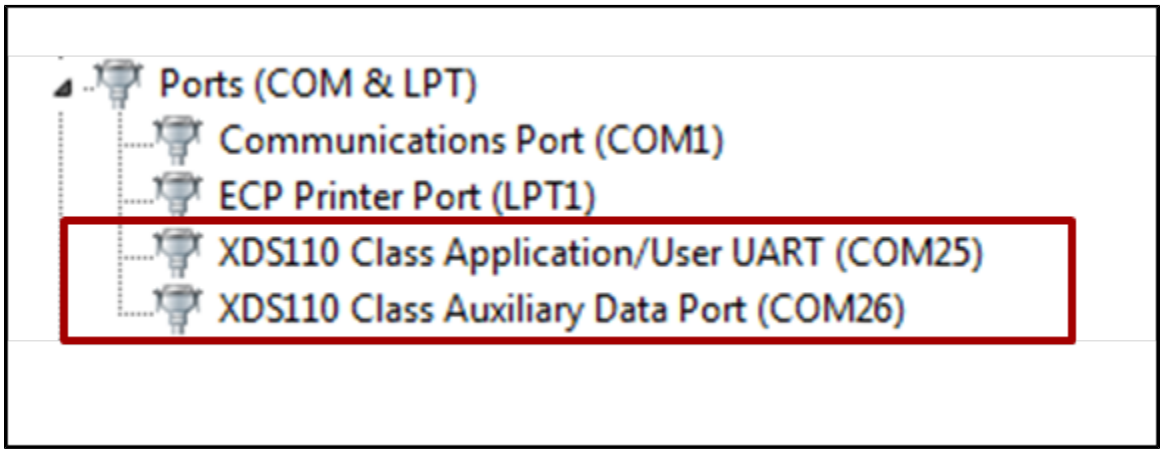



*Figure 3: xWR14xx/xWR16xx PC Connectivity - Device Manager - COM Ports*

**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.

## 4. 2. How to flash an image onto xWR14xx/xWR16xx EVM

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

1. **Setup the Booster Pack EVM for Flashing**

   Refer to the EVM User Guide to understand the bootup modes of the EVM and the SOP jumper locations (See "Sense-on-Power (SOP) Jumpers" section in mmWave device's EVM user guide). To put the EVM in flashing mode, power off the board and place jumpers on pins marked as SOP2 and SOP0 .

| SOP2 jumper | SOP1 jumper | SOP0 jumper | Bootloader mode & operation |
|---|---|---|---|

| | | | |
|---|---|---|---|
| **0** | 0 | 1 | **Functional Mode**<br><br>Device Bootloader loads user application from QSPI Serial Flash to internal RAM and switches the control to it |
| **1** | 0 | 1 | **Flash Programming Mode**<br><br>Device Bootloader spins in loop to allow flashing of user application to the serial flash. |

2. **Procure the Images**
   For flashing xWR1xxx devices, TI Uniflash tool should be used. Users can either use the cloud version available at https://dev.ti.com/uniflash/ or download the desktop version available at http://www.ti.com/tool/UNIFLASH. Detailed instructions on how to use the GUI are described in the Uniflash document " **UniFlash User Guide for mmWave Devices** " located at http://processors.wiki.ti.com/index.php/Category:CCS_UniFlash. This document talks about the steps from the perspective of desktop GUI but the flashing steps (except for installation) should apply for cloud version as well
   Select the following images in the Uniflash tool.
   a. **xWR16xx:**
      For the SDK packaged xWR16xx demos and ccsdebug utility, there is a bin file provided in their respective folder: xwr16xx_<demo|ccsdebug>.bin which is the metaImage to be used for flashing. The metaImage already has the MSS, BSS (RADARSS) and DSS application combined into one file. Users can use this for flashing their own metaImage as well.
      i. For demo mode, mmwave_sdk_<ver>\ti\demo\xwr16xx\mmw\xwr16xx_mmw_demo.bin should be selected.
      ii. For CCS development mode, mmwave_sdk_<ver>\ti\ utils\ccsdebug\xwr16xx_ccsdebug.bin should be selected.
   b. **xWR14xx**:
      For correct operation of mmWave SDK demos, this utility needs 2 binary images:
      i. BSS **:** mmwave_sdk_<ver>\firmware\radarss\xwr12xx_xwr14xx_radarss.bin
      ii. MSS **:** For the SDK packaged xWR14xx demos and ccsdebug utility, there is a bin file provided in their respective folder
      For demo mode, choose the binary file mmwave_sdk_<ver>\packages\ti\demo\xwr14xx\mmw\xwr14xx_mmw_demo_mss.bin from the SDK
      For CCS development mode, mmwave_sdk_<ver>\ti\utils\ccsdebug\xwr14xx_ccsdebug_mss.bin should be selected.
3. **Flashing procedure**

   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. Lets say for this example, it showed up as COM25. Use this COM port in the TI Uniflash tool. Follow the remaining instructions in the " **UniFlash v4 User Guide for mmWave Devices** " to complete the flashing.
4. **Switch back to Functional Mode**

   Refer to the EVM User Guide to understand the bootup modes of the EVM and the SOP jumpers (See "Sense-on-Power (SOP) Jumpers" section in mmWave device's EVM user guide). To put the EVM in functional mode, power off the board and remove jumpers from "SOP2" pin and leave the jumper on "SOP0" pin.

## 4. 3. How to erase flash on xWR14xx/xWR16xx EVM

1. Setup the Booster Pack EVM for flashing as mentioned in step 1 of the section: How to flash an image onto xWR14xx/xWR16xx EVM
2. Follow the instructions in " **UniFlash v4 User Guide for mmWave Devices** " section "**Format SFLASH Button**".
3. Switch back to Functional Mode as mentioned in step 4 of the section: How to flash an image onto xWR14xx/xWR16xx EVM

## 4. 4. How to connect xWR14xx/xWR16xx 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. Refer to the release notes for TI code composer studio and emulation pack version that would be needed.

### 4. 4. 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. 2. Device support package Update

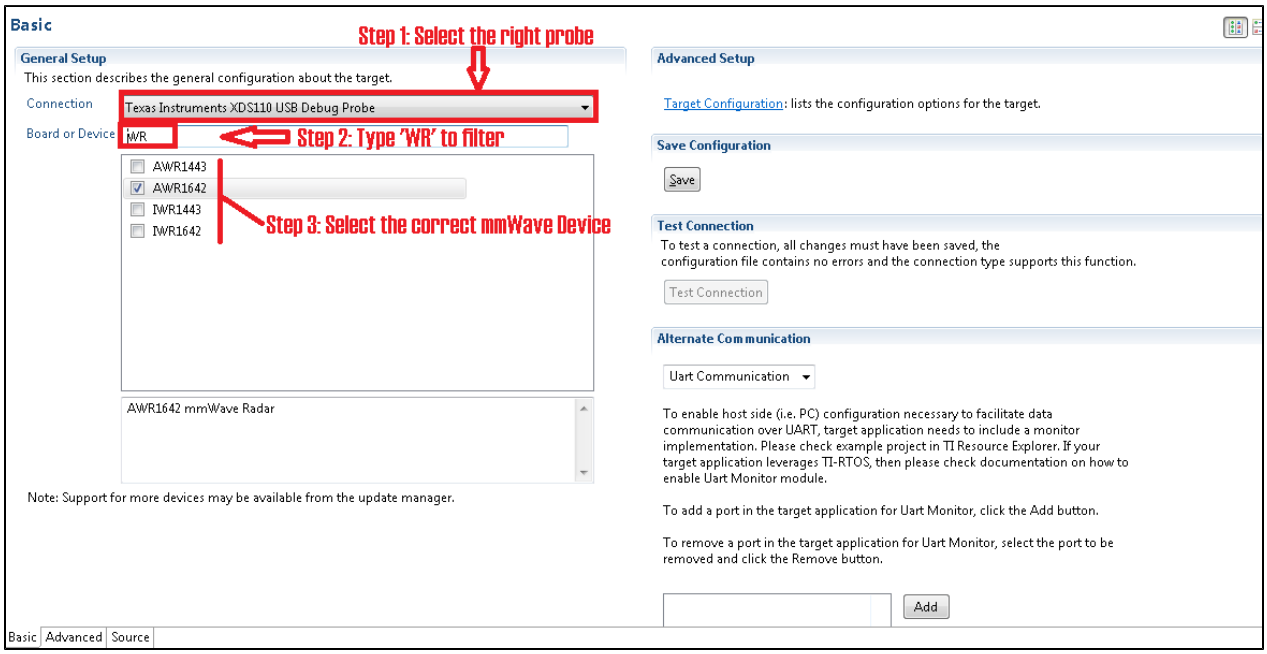
To create the ccxml file for connecting to the EVM, you will need to first update the device support package within CCS. Refer to the mmWave SDK release notes for the device support package version that would be needed within CCS to connect to the EVM. Check if that particular or its later version of "mmWave Radar Device Support" 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. 3. Target Configuration file for CCS (CCXML)

### 4. 4. 3. 1. Creating a CCXML file

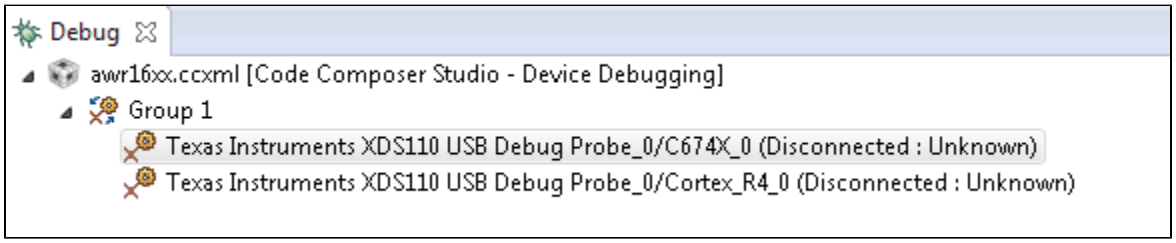
Assuming you have updated the device support package and Emulation pack as mentioned in the section above, follow the steps mentioned below to create a target configuration file in CCS.

1. If your CCS does not already show "Target Configurations" window, do View->Target Configurations
2. This will show the "Target Configurations" window, right click in the window and select "New Target Configuration"
3. Give an appropriate name to the ccxml file you want to create for the EVM
4. Scroll the "Connection" list and select "Texas Instruments XDS110 USB Debug Probe", when this is done, the "Board or Device" list will be filtered to show the possible candidates, find and choose AWR1642 or AWR1443 and check the box. Click Save and the file will be created.




### 4. 4. 3. 2. Connecting to xWR14xx/xWR16xx EVM using CCXML in CCS

Follow steps in above section to create a ccxml file. Once created, the target configuration file will be seen in the "Target Configurations" list and you can launch the target by selecting it and with right-click select the "Launch Selected Configuration" option. This will launch the target and the Debug window will show all the cores present on the device. You can connect to the target with right-click and doing "Connect Target".



## 4. 5. Developing using SDK

### 4. 5. 1. Build Instructions

Follow the mmwave_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. 5. 2. Setting up build environment

#### 4. 5. 2. 1. Windows

1. Create command prompt at <mmwave_sdk_<ver> install path>\packages\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: awr14xx, iwr14xx, awr16xx, iwr16xx
set MMWAVE_SDK_DEVICE=awr16xx

@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

@REM If using a secure device this variable needs to be updated with the path to
mmwave_secdev_<ver> folder
set MMWAVE_SECDEV_INSTALL_PATH=

@REM If using a secure device, this variable needs to be updated with the path to hsimage.cfg file
that
@REM has customer specific certificate/key information. A sample hsimage.cfg file is in the secdev
package
set MMWAVE_SECDEV_HSIMAGE_CFG=%MMWAVE_SECDEV_INSTALL_PATH%/hs_image_creator/hsimage.cfg
```

Refer to the MMWAVE-SECDEV User Guide to setup environment needed for builds for high secure (HS) devices. For non secure devices the MMWAVE_SECDEV_INSTALL_PATH environment variable should be empty.

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

**Run setenv.bat**

```
setenv.bat
```

#### 4. 5. 2. 2. Linux

1. Open a terminal and cd to <mmwave_sdk_<ver> install path>/packages/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)
###############################################################################
# Select your device. Options (case sensitive) are: awr14xx, iwr14xx, awr16xx, iwr16xx
export MMWAVE_SDK_DEVICE=awr16xx

# 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 using a secure device, this variable needs to be updated with the path to mmwave_secdev_<ver>
folder
export MMWAVE_SECDEV_INSTALL_PATH=

# If using a secure device, this variable needs to be updated with the path to hsimage.cfg file
that
# has customer specific certificate/key information. A sample hsimage.cfg file is in the secdev
package
export MMWAVE_SECDEV_HSIMAGE_CFG=${MMWAVE_SECDEV_INSTALL_PATH}/hs_image_creator/hsimage.cfg
```

Refer to the MMWAVE-SECDEV User Guide to setup environment needed for builds for high secure (HS) devices. For non secure devices the MMWAVE_SECDEV_INSTALL_PATH environment variable should be empty.

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 mono. One of the Image Creator binaries (out2rprc.exe) is a windows executable that needs mono to run in Linux environment

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

4. Run **setenv.sh** as shown below. This should not give errors and should print the message **"mmWave Build Environment Configured"**. The build environment is now setup.

**Run setenv.sh**

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

### 4. 5. 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 xWR14xx/xWR16xx EVM".

#### 4. 5. 3. 1. Building demo in Windows

**Building demo in windows**

```
REM Use xwr14xx or xwr16xx for <device type> below. Use mmw for <demo> below
cd %MMWAVE_SDK_INSTALL_PATH%/ti/demo/<device type>/<demo>

REM Clean and build
gmake clean
gmake all

REM Incremental build
gmake all

REM For example to build the mmw demo for awr14xx or iwr14xx
cd %MMWAVE_SDK_INSTALL_PATH%/ti/demo/xwr14xx/mmw
gmake clean
gmake all
REM This will create xwr14xx_mmw_demo_mss.xer4f & xwr14xx_mmw_demo_mss.bin binaries under
REM %MMWAVE_SDK_INSTALL_PATH%/ti/demo/xwr14xx/mmw folder

REM For example to build the mmw demo for awr16xx or iwr16xx
cd %MMWAVE_SDK_INSTALL_PATH%/ti/demo/xwr16xx/mmw
gmake clean
gmake all
REM This will create xwr16xx_mmw_demo_mss.xer4f, xwr16xx_mmw_demo_dss.xe674 & xwr14xx_mmw_demo.bin
REM binaries under %MMWAVE_SDK_INSTALL_PATH%/ti/demo/xwr16xx/mmw folder
```

#### 4. 5. 3. 2. Building demo in Linux

**Building demo in linux**

```
# Use xwr14xx or xwr16xx for <device type> below. Use mmw for <demo> below
cd ${MMWAVE_SDK_INSTALL_PATH}/ti/demo/<device type>/<demo>

# Clean and build
make clean
make all

# Incremental build
make all

# For example to build the mmw demo for awr14xx or iwr14xx
cd ${MMWAVE_SDK_INSTALL_PATH}/ti/demo/xwr14xx/mmw
make clean
make all
# This will create xwr14xx_mmw_demo_mss.xer4f & xwr14xx_mmw_demo_mss.bin binaries under
# ${MMWAVE_SDK_INSTALL_PATH}/ti/demo/xwr14xx/mmw folder

# For example to build the mmw demo for awr16xx or iwr16xx
cd ${MMWAVE_SDK_INSTALL_PATH}/ti/demo/xwr16xx/mmw
make clean
make all
# This will create xwr16xx_mmw_demo_mss.xer4f, xwr16xx_mmw_demo_dss.xe674 & xwr14xx_mmw_demo.bin
# binaries under ${MMWAVE_SDK_INSTALL_PATH}/ti/demo/xwr16xx/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. 5. 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. 5. 4. 1. Building drivers/control/alg components

To clean build driver, control or alg component and its unit test, 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/<path to the component>
gmake clean
gmake all

REM For example to build the adcbuf lib and unit test
cd %MMWAVE_SDK_INSTALL_PATH%/ti/drivers/adcbuf
gmake clean
gmake all
REM If MMWAVE_SDK_DEVICE is set to awr14xx or iwr14xx, the commands will create
REM     libadcbuf_xwr14xx.aer4f library under ti/drivers/adcbuf/lib folder
REM     xwr14xx_adcbuf_mss.xer4f unit test binary under ti/drivers/adcbuf/test/xwr14xx folder
REM If MMWAVE_SDK_DEVICE is set to awr16xx or iwr16xx, the commands will create
REM     libadcbuf_xwr16xx.aer4f library for MSS under ti/drivers/adcbuf/lib folder
REM     xwr16xx_adcbuf_mss.xer4f unit test binary for MSS under ti/drivers/adcbuf/test/xwr16xx folder
REM     libadcbuf_xwr16xx.ae674 library for DSS under ti/drivers/adcbuf/lib folder
REM     xwr16xx_adcbuf_dss.xe674 unit test binary for DSS under ti/drivers/adcbuf/test/xwr16xx folder
REM Above paths are relative to %MMWAVE_SDK_INSTALL_PATH%/

REM For example to build the mmwavelink lib and unit test
cd %MMWAVE_SDK_INSTALL_PATH%/ti/control/mmwavelink
gmake clean
gmake all
REM If MMWAVE_SDK_DEVICE is set to awr14xx or iwr14xx, the commands will create
REM     libmmwavelink_xwr14xx.aer4f library under ti/control/mmwavelink/lib folder
REM     xwr14xx_link_mss.xer4f unit test binary under ti/drivers/control/mmwavelink/test/xwr14xx folder
REM If MMWAVE_SDK_DEVICE is set to awr16xx or iwr16xx, the commands will create
REM     libmmwavelink_xwr16xx.aer4f library for MSS under ti/control/mmwavelink/lib folder
REM     xwr16xx_link_mss.xer4f unit test binary for MSS under ti/control/mmwavelink/test/xwr16xx folder
REM     libmmwavelink_xwr16xx.ae674 library for DSS under ti/control/mmwavelink/lib folder
REM     xwr16xx_link_dss.xe674 unit test binary for DSS under ti/control/mmwavelink/test/xwr16xx folder
REM Above paths are relative to %MMWAVE_SDK_INSTALL_PATH%/

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/<path to the component>
make clean
make all

# For example to build the adcbuf lib and unit test
cd ${MMWAVE_SDK_INSTALL_PATH}/ti/drivers/adcbuf
make clean
make all
# If MMWAVE_SDK_DEVICE is set to awr14xx or iwr14xx, the commands will create
#     libadcbuf_xwr14xx.aer4f library under ti/drivers/adcbuf/lib folder
#     xwr14xx_adcbuf_mss.xer4f unit test binary under ti/drivers/adcbuf/test/xwr14xx folder
# If MMWAVE_SDK_DEVICE is set to awr16xx or iwr16xx, the commands will create
#     libadcbuf_xwr16xx.aer4f library for MSS under ti/drivers/adcbuf/lib folder
#     xwr16xx_adcbuf_mss.xer4f unit test binary for MSS under ti/drivers/adcbuf/test/xwr16xx folder
#     libadcbuf_xwr16xx.ae674 library for DSS under ti/drivers/adcbuf/lib folder
#     xwr16xx_adcbuf_dss.xe674 unit test binary for DSS under ti/drivers/adcbuf/test/xwr16xx folder
# Above paths are relative to ${MMWAVE_SDK_INSTALL_PATH}/

# For example to build the mmwavelink lib and unit test
cd ${MMWAVE_SDK_INSTALL_PATH}/ti/control/mmwavelink
make clean
make all
# If MMWAVE_SDK_DEVICE is set to awr14xx or iwr14xx, the commands will create
#     libmmwavelink_xwr14xx.aer4f library under ti/control/mmwavelink/lib folder
#     xwr14xx_link_mss.xer4f unit test binary under ti/drivers/control/mmwavelink/test/xwr14xx folder
# If MMWAVE_SDK_DEVICE is set to awr16xx or iwr16xx, the commands will create
#     libmmwavelink_xwr16xx.aer4f library for MSS under ti/control/mmwavelink/lib folder
#     xwr16xx_link_mss.xer4f unit test binary for MSS under ti/control/mmwavelink/test/xwr16xx folder
#     libmmwavelink_xwr16xx.ae674 library for DSS under ti/control/mmwavelink/lib folder
#     xwr16xx_link_dss.xe674 unit test binary for DSS under ti/control/mmwavelink/test/xwr16xx folder
# Above paths are relative to ${MMWAVE_SDK_INSTALL_PATH}/

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

**example output of make help for drivers and mmwavelink**

```
******************************************************************
* Makefile Targets for the ADCBUF
clean                -> Clean out all the objects
drv                  -> Build the Driver only
drvClean             -> Clean the Driver Library only
test                 -> Builds all the unit tests for the SOC
testClean            -> Cleans the unit tests for the SOC
******************************************************************
```

**example output of make help for mmwave control and alg component**

```
******************************************************************
* Makefile Targets for the mmWave Control
clean                -> Clean out all the objects
lib                  -> Build the Core Library only
libClean             -> Clean the Core Library only
test                 -> Builds all the Unit Test
testClean            -> Cleans all the Unit Tests
******************************************************************
```

Please note that not all drivers are supported for all devices. List of supported drivers for each device is listed in the Release Notes.

#### 4. 5. 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.

`MMWAVE_SECDEV_INSTALL_PATH`

# 5. MMWAVE SDK deep dive

## 5. 1. System Deployment

### 5. 1. 1. xWR14xx

A typical mmWave application using xWR14xx would perform these operations:

- Control and monitoring of RF front-end through mmwaveLink
- External communications through standard peripherals
- Some radar data processing using FFT HW accelerator

Typical xWR14xx system deployments could be envisioned as follows:

1. **Autonomous xWR14xx sensor (aka Standalone mode)**
    a. xWR14xx program code is downloaded from the serial flash memory attached to xWR14xx (via QSPI)
    b. Optional high level control from remote entity
    c. Sends low speed data output (objects detected) to remote entity
2. **Hybrid xWR14xx sensor + Controller**
    a. Serial flash is attached/in-built to external controller and SPI interface exists between xWR14xx and controller
    b. High level control from controller (code download, GPIO toggling, etc)
    c. Sends low speed data output (objects detected) to controller.
3. **Satellite xWR14xx sensor + DSP**
    a. Program code is either in serial flash memory attached to xWR14xx (via QSPI) or downloaded via the control interface between xWR14xx and DSP (ex: via SPI)
    b. High level control from DSP
    c. Sends high speed data output (1D/2D FFT output) to DSP

These deployments are depicted in the Figure 4 and Figure 5.

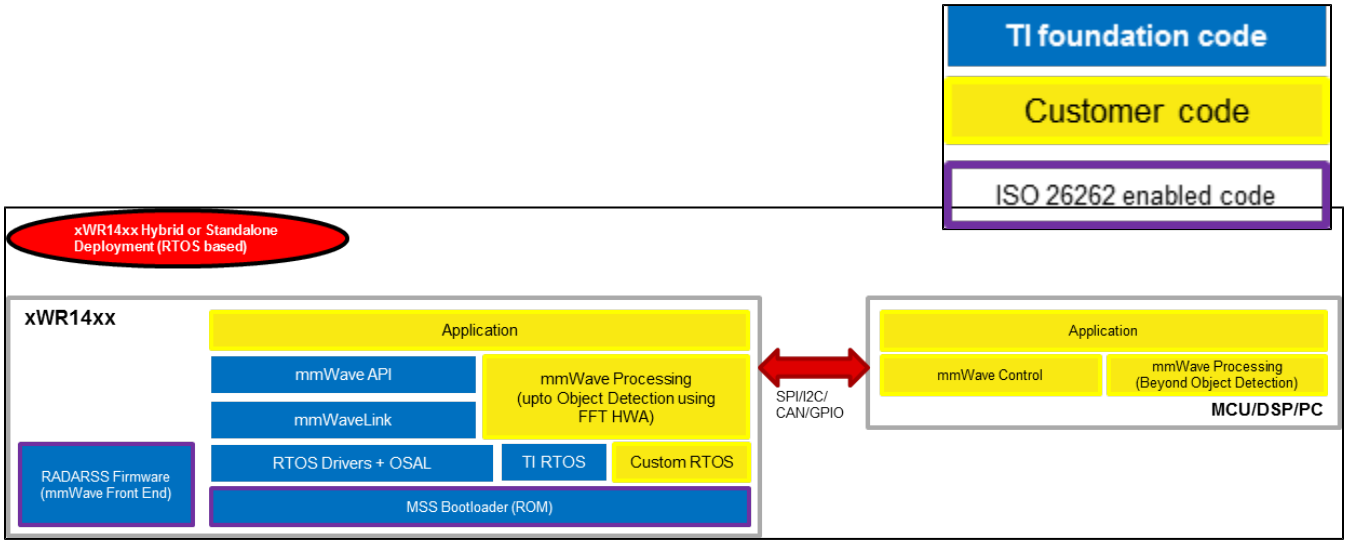



*Figure 4: xWR14xx Deployment in Hybrid or Standalone mode*

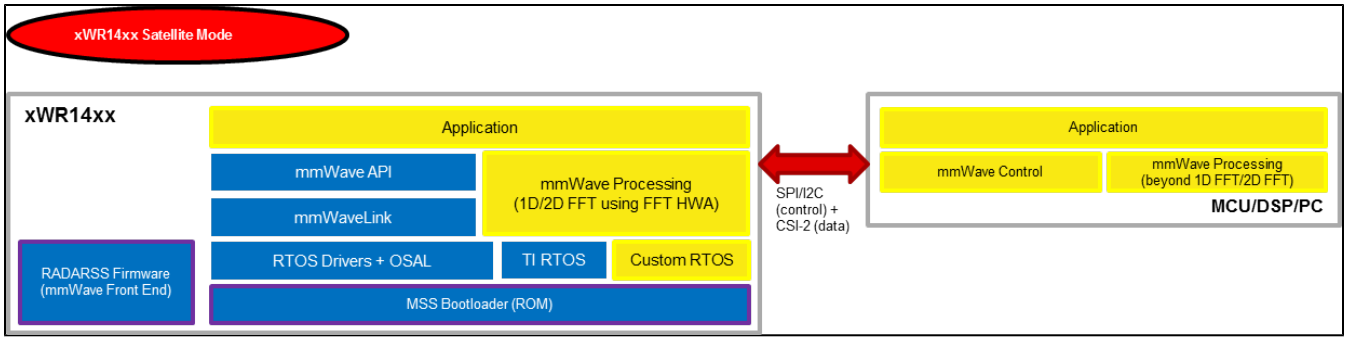



*Figure 5: xWR14xx Deployment in Satellite mode*

Note that the software architecture presented above demonstrates only the mmWave SDK components running on the external devices – MCU, DSP, PC. There are other software components running on those external devices which are part of the ecosystem of those devices and out of scope for this document. The mmWave SDK package would provide, in future, sample code for the mmWave API running on these external devices but the porting of this layer onto these external device ecosystem is the responsibility of system integrator/application code provider.

### 5. 1. 2. xWR16xx

A typical xWR16xx 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 xWR16xx customer deployment is shown in Figure 6:

a. xWR16xx program code for MSS and DSP-SS is downloaded from the serial flash memory **attached** to xWR16xx (via QSPI)
b. **Optional** high level control from remote entity
c. Sends **low speed data** output (objects detected) to remote entity
d. **Optional** high speed data (debug) sent out of device over LVDS

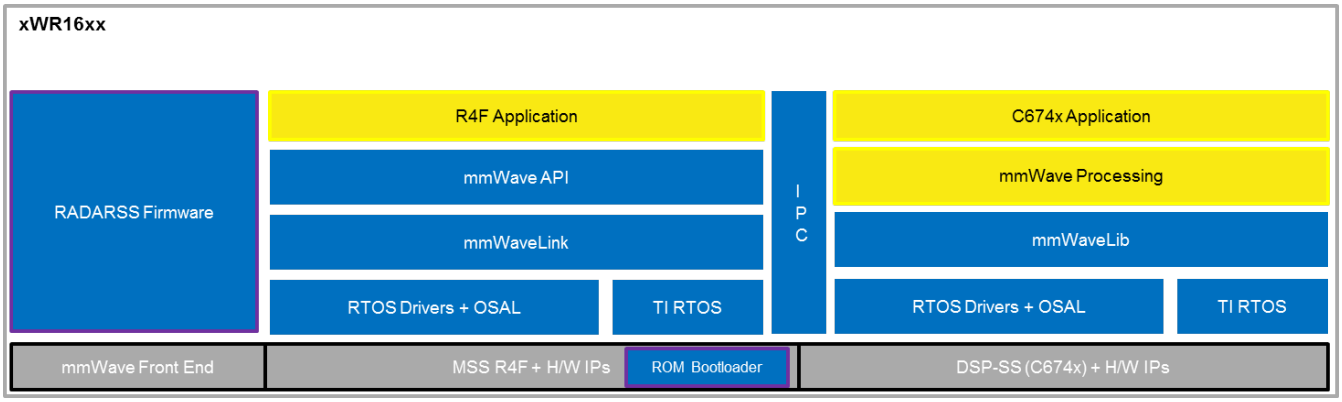



*Figure 6: Autonomous xWR16xx sensor (Standalone mode)*

## 5. 2. Typical mmWave Radar Processing Chain

Following figure shows a typical mmWave Radar processing chain:

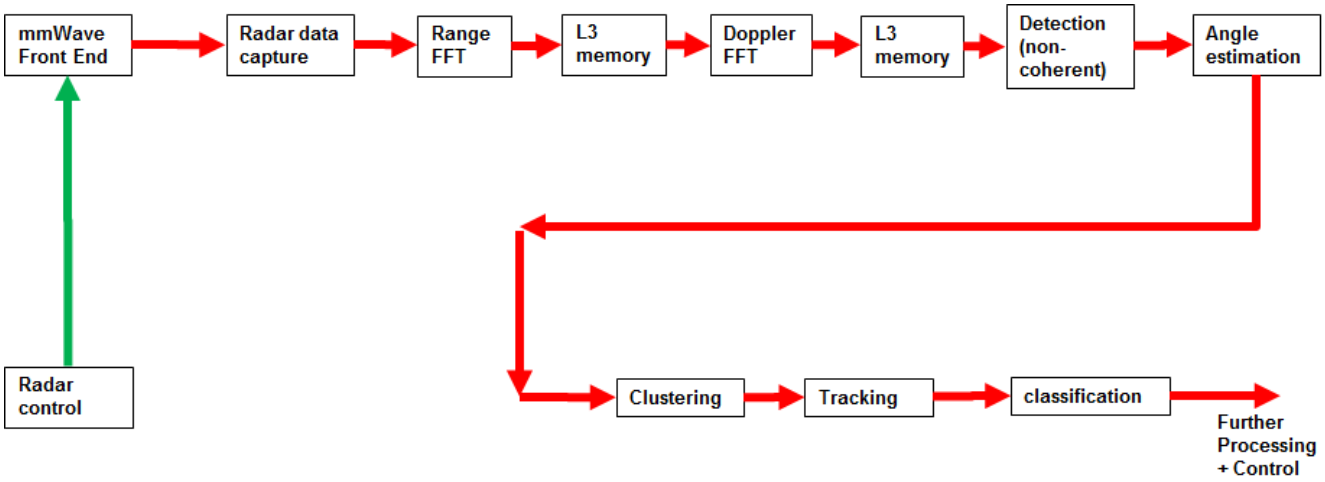



*Figure 7: Typical mmWave radar processing chain*

Using mmWave SDK the above chain could be realized as shown in the following figure for xWR14xx and xWR16xx. 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 AR device or would run entirely on the external processor.

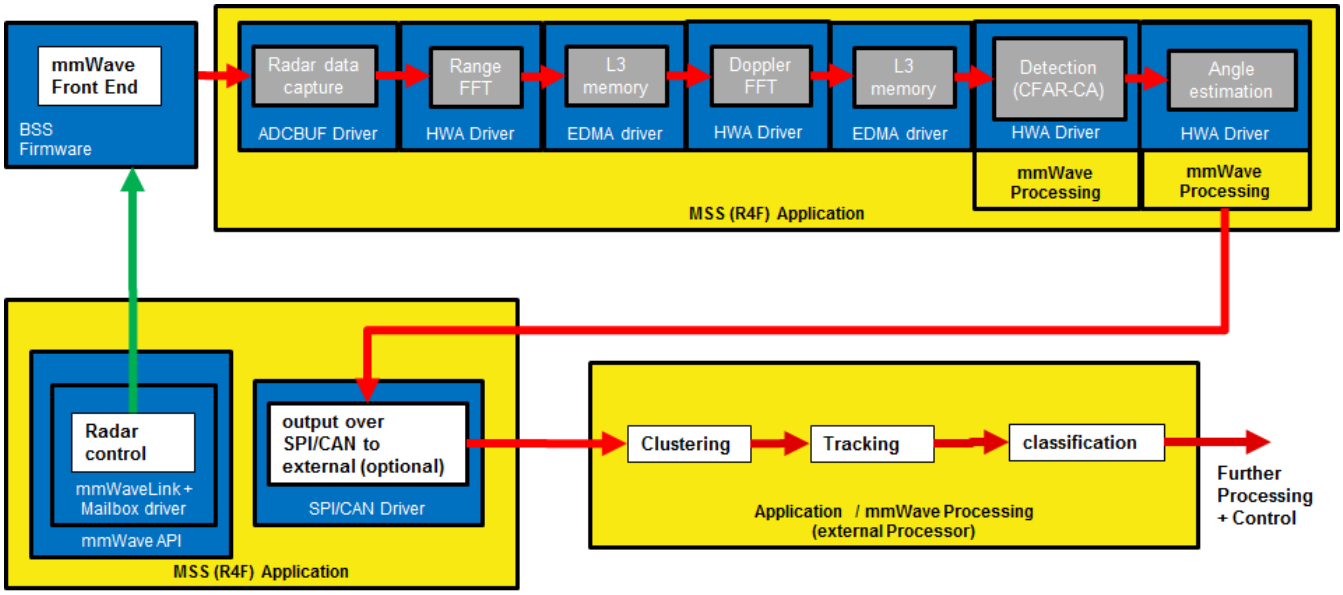



*Figure 8: Typical mmWave radar processing chain using xWR14xx mmWave SDK*

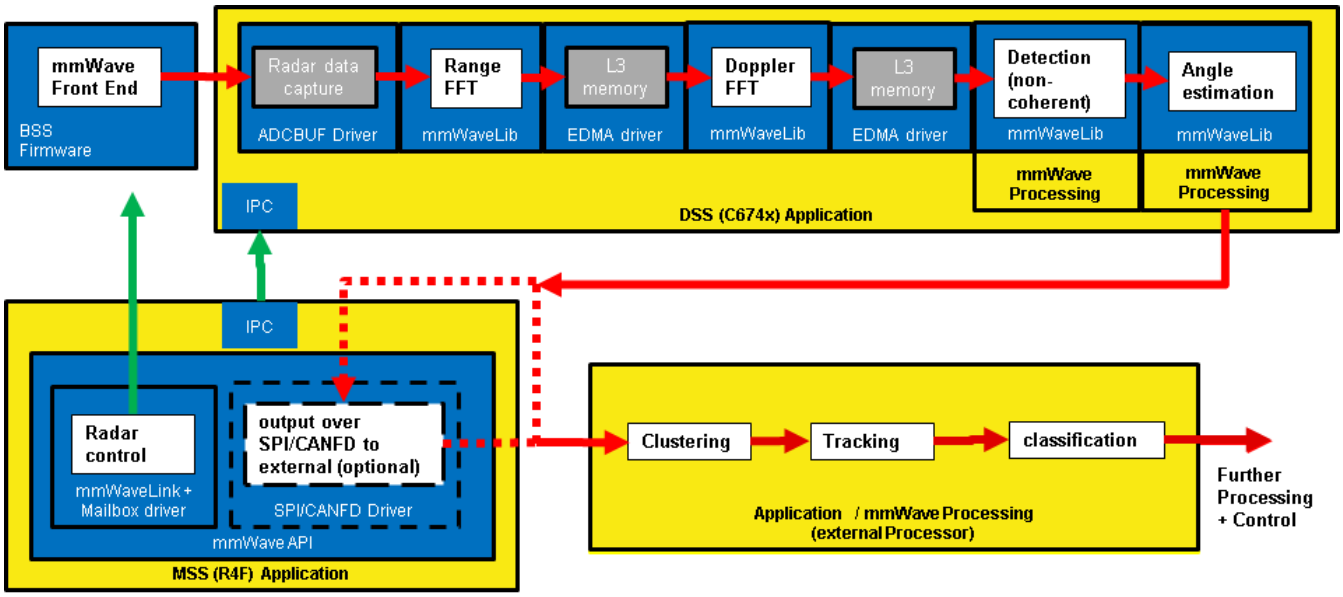



*Figure 9: Typical mmWave radar processing chain using xWR16xx mmWave SDK*

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

## 5. 3. Typical Programming Sequence

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

### 5. 3. 1. Control Path

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

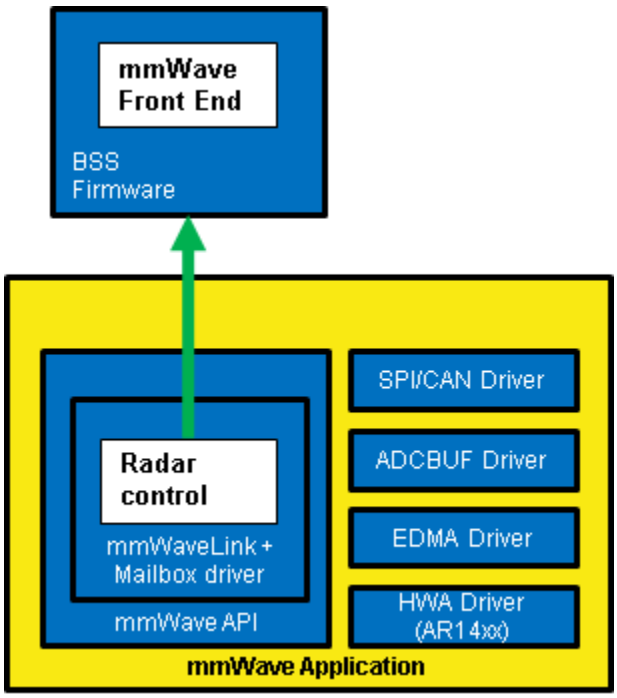


*Figure 10: Typical mmWave radar control flow*

Following set of figures shows how an application programming sequence would be for setting up the typical control path - init, config, start. This is a high level diagram simplified to highlight the main sofwtare 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_sdk_<ver>\packages\ti\demo\<platform>\mmw folder.

#### 5. 3. 1. 1. xWR14xx (MSS<->RADARSS)

On xWR14xx, the control path runs on the Master subsystem (Cortex-R4F) and the application can simply call the mmwave APIs in the SDK to realize most of the functionality.

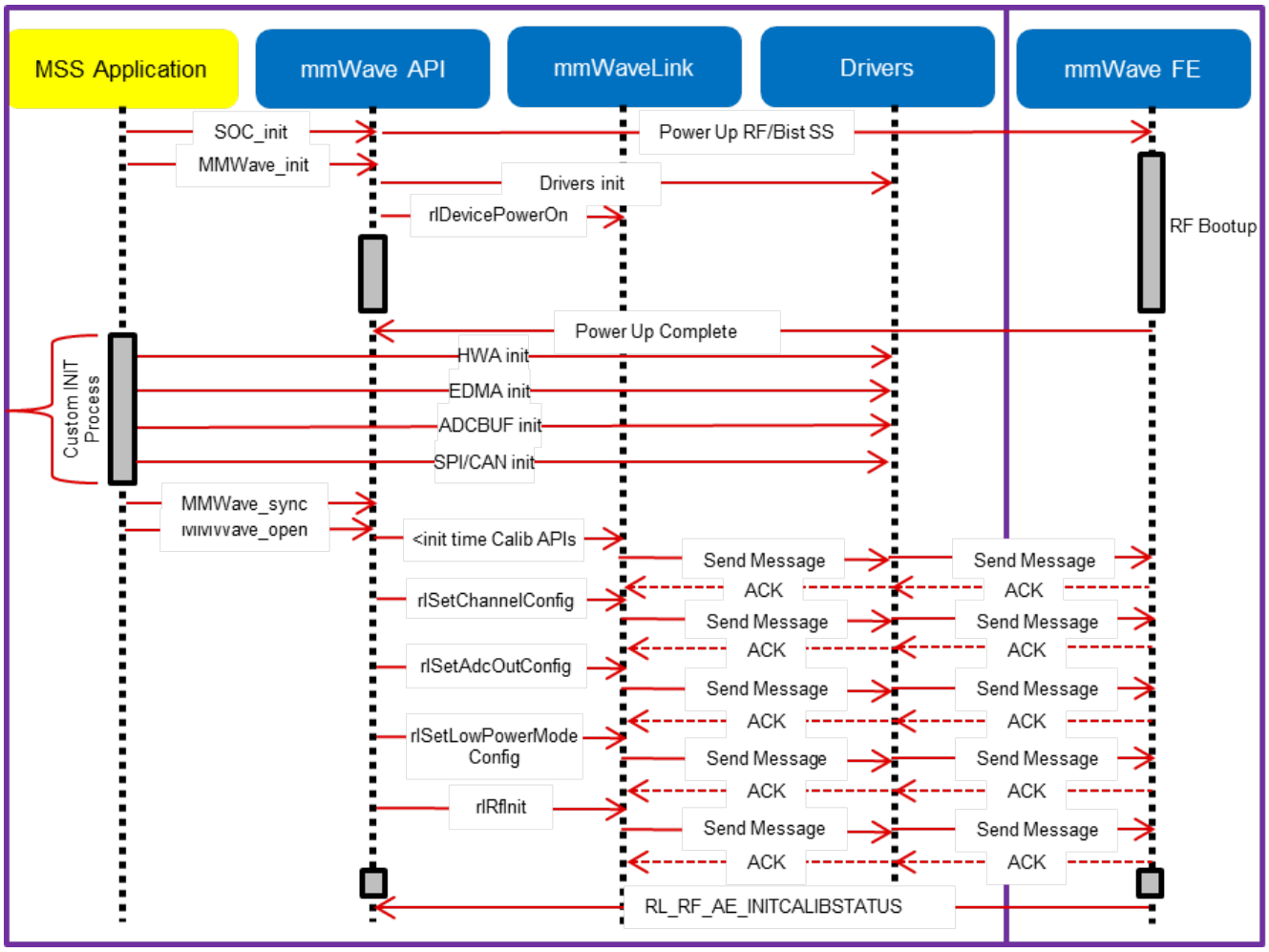



*Figure 11: xWR14xx: Detailed Control Flow (Init sequence)*

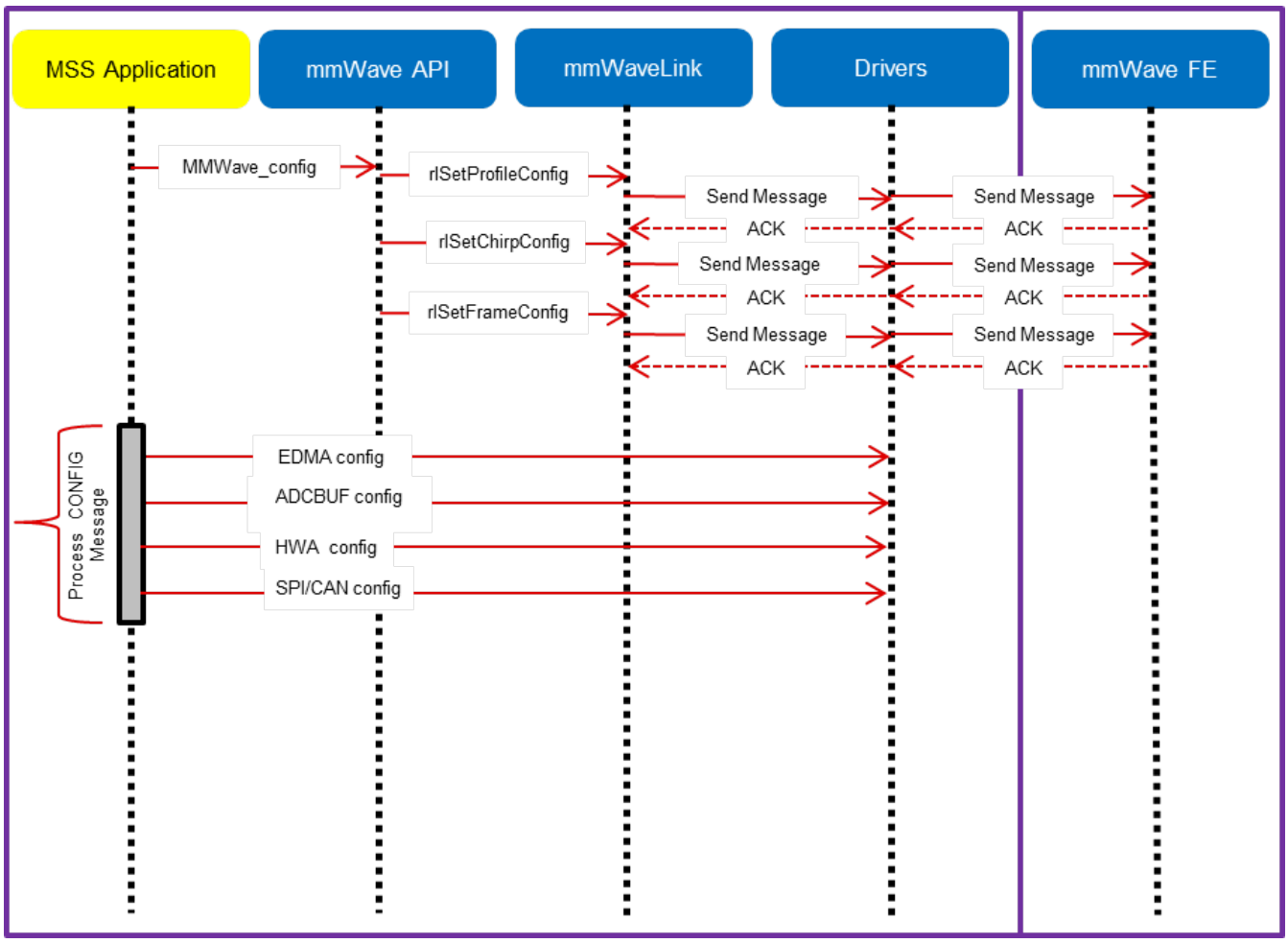



*Figure 12: xWR14xx: Detailed Control Flow (Config sequence)*

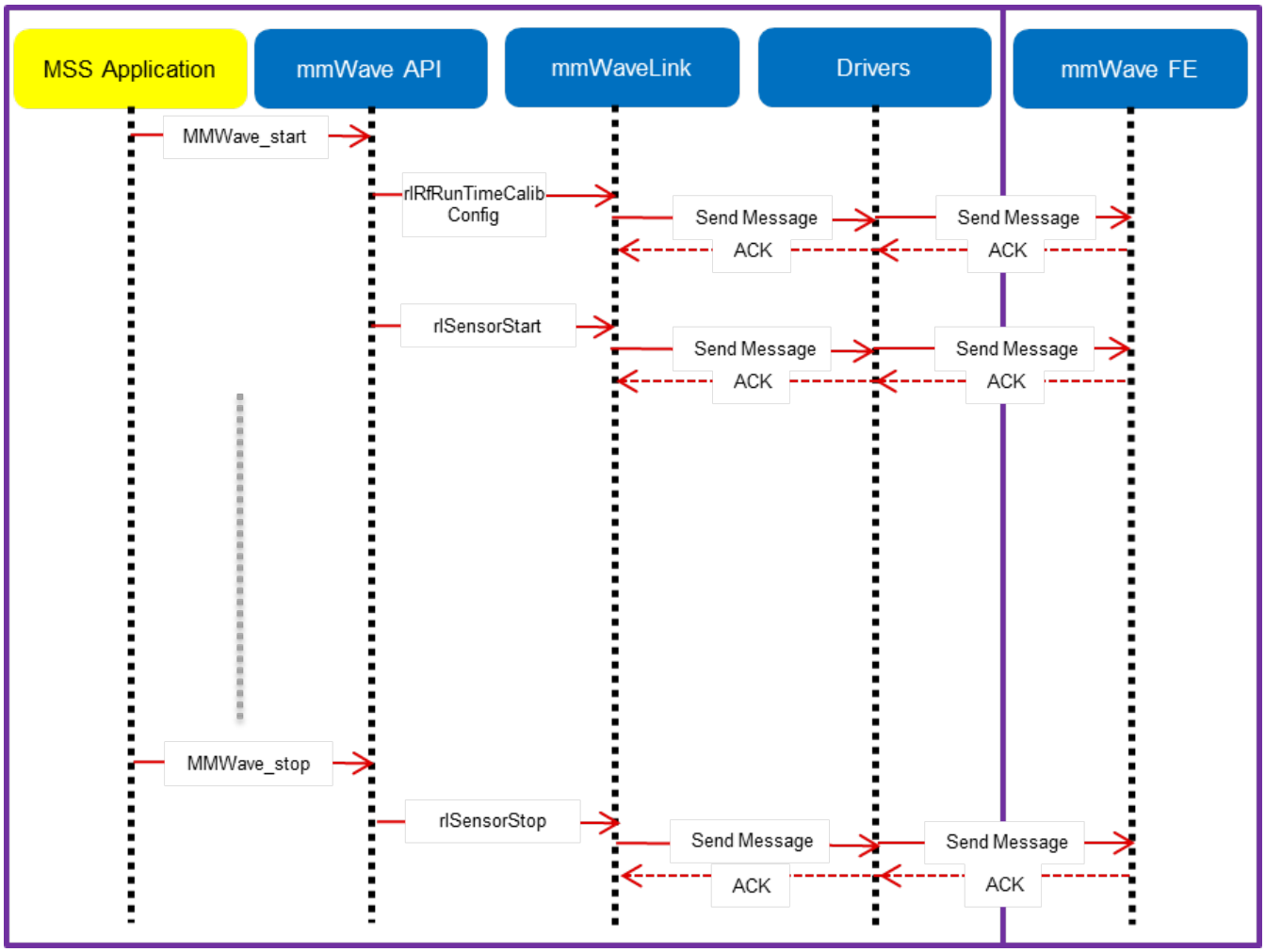



*Figure 13: xWR14xx: Detailed Control Flow (start sequence)*

#### 5. 3. 1. 2. xWR16xx

On xWR16xx, the control path can run on MSS only, DSS only or in "co-operative" mode where the init and config are 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 co-operative mode can be seen in the MMW demo. The mmWave unit tests provide a sample implementation of all 3 modes.

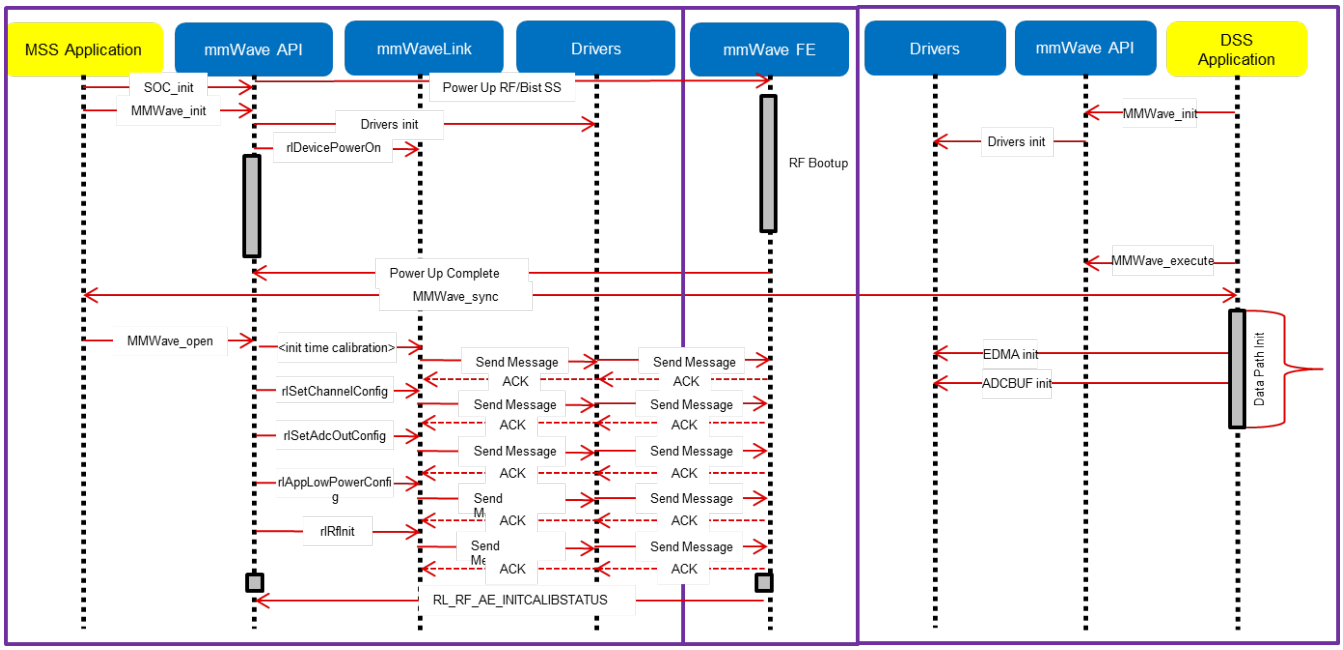



*Figure 14: xWR16xx: Detailed Control Flow (Init sequence)*

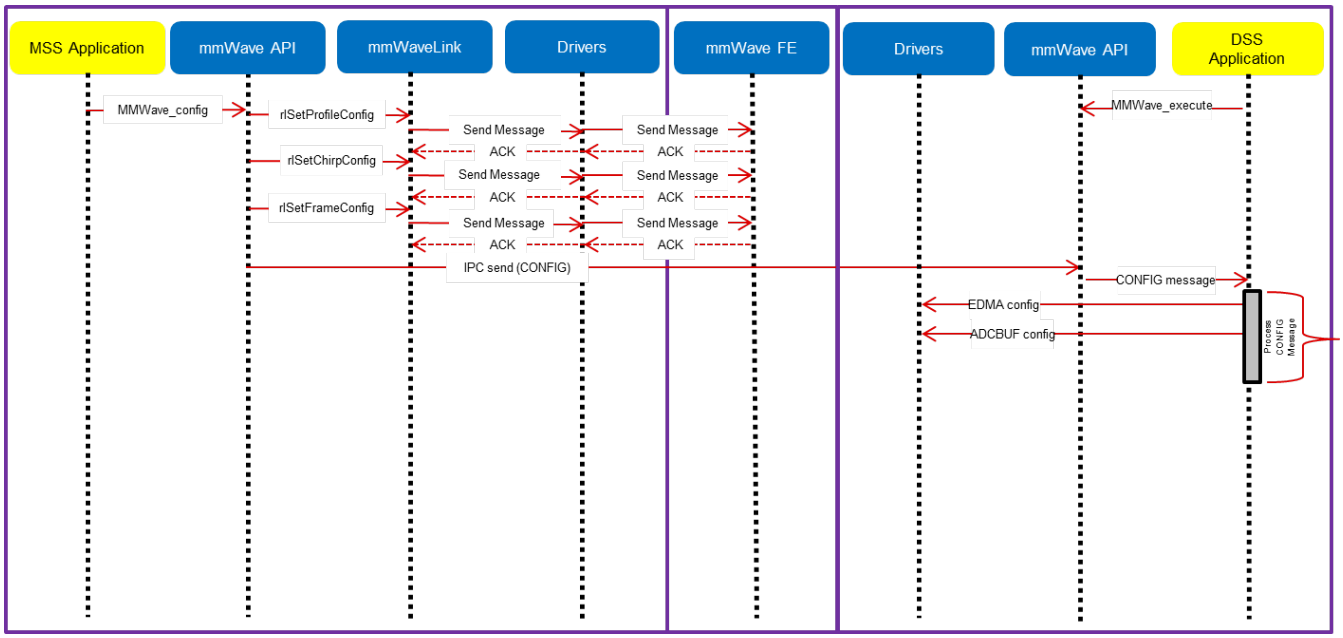



*Figure 15: xWR16xx: Detailed Control Flow (Config sequence)*

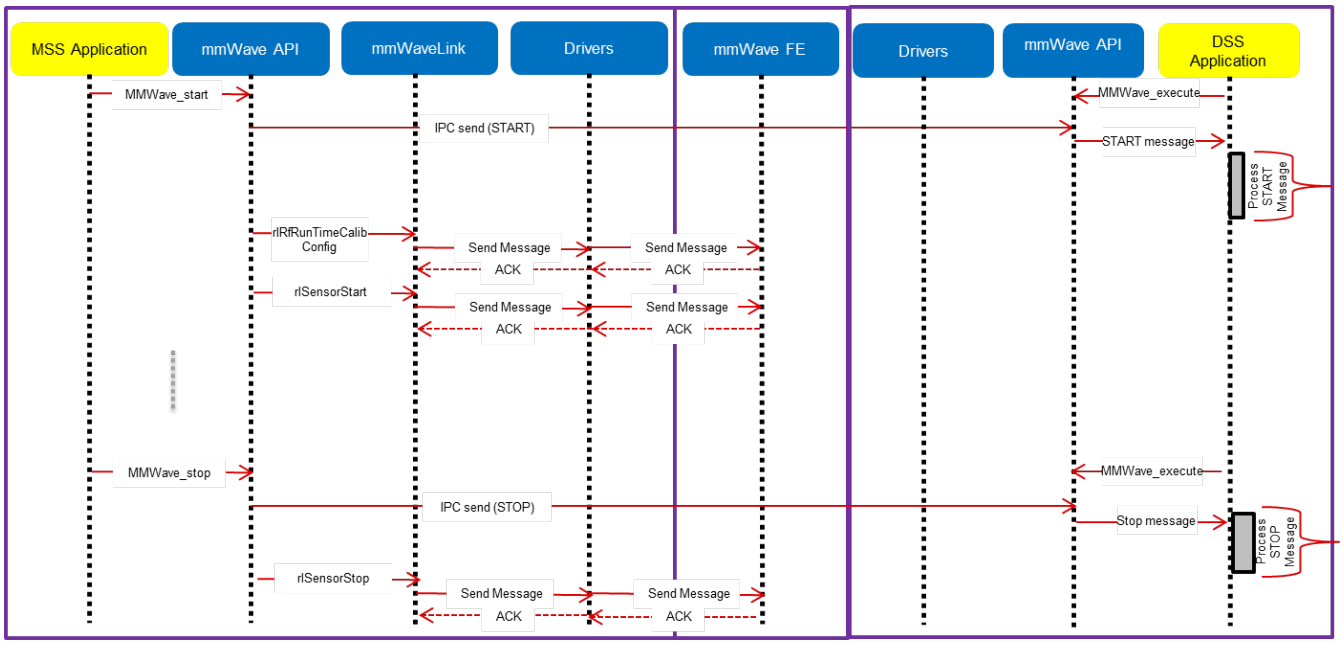



*Figure 16: **xWR16xx: Detailed Control Flow (Start sequence)***

## 5. 3. 2. Data Path

### 5. 3. 2. 1. xWR14xx

In xWR14xx, the data path in the above processing chain is depicted by the following blocks running on the MSS (Cortex-R4F).

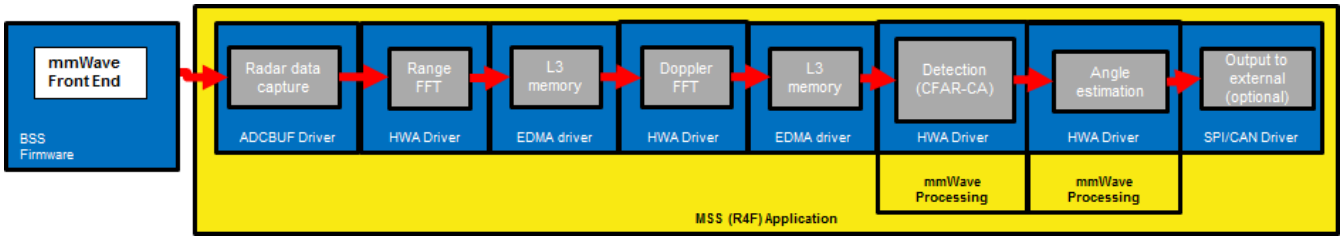



*Figure 17: Typical mmWave radar data flow in xWR14xx*

Please refer to the documentation provided here mmwave_sdk_<ver>\packages\ti\demo\xwr14xx\mmw\docs\doxygen\html\index.html for more details on each of the individual blocks of the data path.

### 5. 3. 2. 2. xWR16xx

In xWR16xx, the data path in the above processing chain is depicted by the following blocks running primarlity on the DSS (C674x).

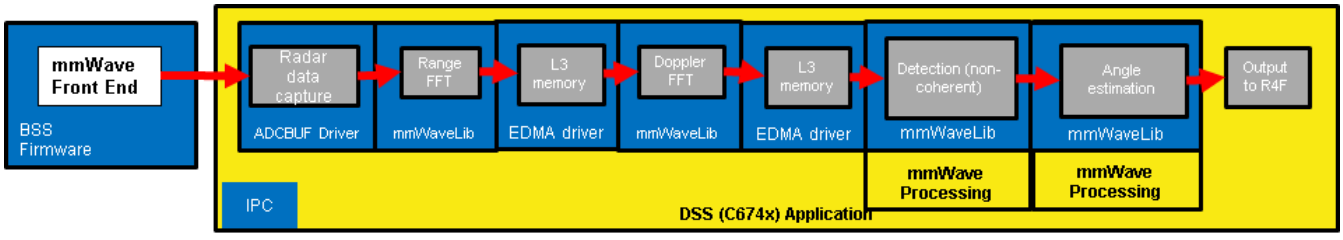



*Figure 18: Typical mmWave radar data flow in xWR16xx*

Please refer to the documentation provided here mmwave_sdk_<ver>\packages\ti\demo\xwr16xx\mmw\docs\doxygen\html\index.html for more details on each of the individual blocks of the data path.

## 5. 4. mmWave SDK - TI components

The mmWave SDK functionality broken down into components are explained in next few subsections.

### 5. 4. 1. Drivers

Drivers encapsulate the functionality of the various hardware IPs in the system and provide a well defined API to the higher layers. The drivers are designed to be OS-agnostic via the use of OSAL layer. Following figure shows typical internal software blocks present in the SDK drivers. The source code for the SDK drivers are present in the mmwave_sdk_<ver>\packages\ti\drivers\<ip> folder. Documentation of the API is available via doxygen and placed at mmwave_sdk_<ver>\packages\ti\drivers\<ip>\docs\doxygen\html\index.html . The driver's unit test code, running on top of SYSBIOS is also provided as part of the package mmwave_sdk_<ver>\packages\ti\drivers\<ip>\test\. The library for the drivers are placed in the mmwave_sdk_<ver>\packages\ti\drivers\<ip>\lib directory and the file is named lib<ip>_<platform>.aer4f for MSS and lib<ip>_<platform>.ae 674 for DSP.

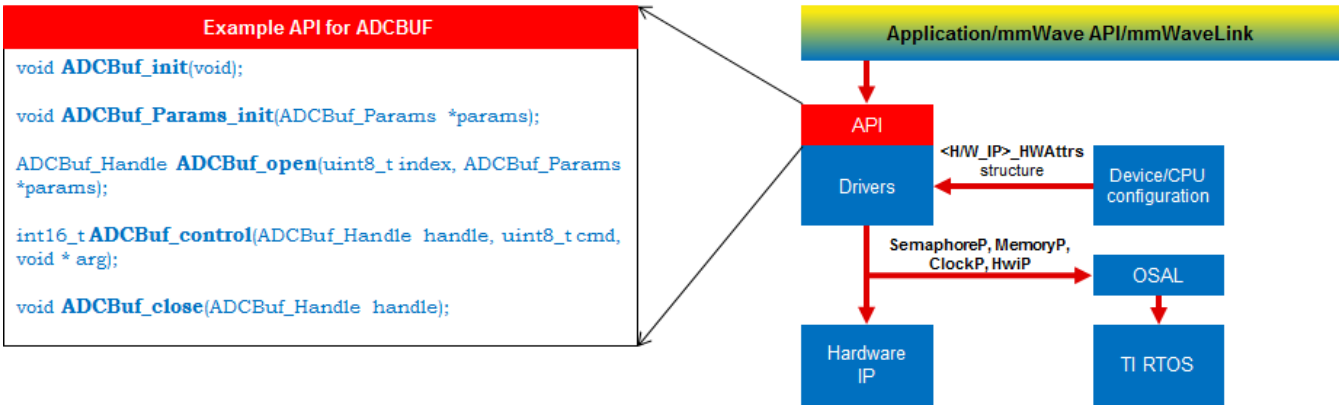



*Figure 19: mmWave SDK Drivers - Internal software design*

| Drivers/Hardware IP | Platform supported | Driver Functionality Implemented in mmWave SDK |
|---|---|---|
| ADCBUF | xWR14xx R4F<br><br>xWR16xx R4F<br>xWR16xx DSP | All features of IP (ADCBUF, CQ) are implemented in the driver |
| CAN | xWR14xx R4F<br><br>xWR16xx R4F | Following features of IP are implemented in the driver:<br>Configure Rx and Tx I/O Control registers<br>Configure DCAN mode of operation<br>Configure DCAN controller, interrupts, ECC, parity<br>Set bit time parameters<br>Configure Rx and Tx message objects<br>Receive and Transmit a CAN message<br>Retrieve Tx message object transmission status and Rx message object reception status<br>Check the validity of the received message |

| | | |
|---|---|---|
| CANFD | AWR16xx R4F | Following features of IP are implemented in the driver:<br>• Reset MCAN driver<br>• Initialize MCAN clock stop controls, auto wakeup, MCAN mode - classic versus FD mode, transceiver delay conpensation<br>• Configure MCAN controller and global filters<br>• Configure MCAN mode of operation<br>• Set bit time parameters<br>• Configure message filters, Rx/Tx FIFOs<br>• Add and cancel Tx requests<br>• Transmits a CAN message<br>• Receive a CAN message<br>• Check the validity of the received message<br>• Configure interrupt multiplexer to service message objects<br>• Retrieve interrupt line status, interrupt pending status, parity error status, bit error status, ECC diagnostics status, ECC error status and MCAN error status<br>• Clear interrupt pending status, ECC diagnostics error status, ECC error status and MCAN error status<br>• Configure MCAN parity function, self test mode, ECC Diagnostic mode |
| CBUFF | xWR14xx R4F<br><br>xWR16xx R4F<br>xWR16xx DSP | Following features of IP are implemented in the driver:<br>Supports Platform defined HSI: LVDS or CSI (IWR14xx only).<br>Initialize and setup the CBUFF Driver<br>Configure the Linked List and EDMA Channels to support the data transfer<br>Supports Interleaved and Non-Interleaved data mode<br>Supports the data formats: ADC, ADC_CP, CP_ADC, CP_ADC_CQ<br>Supports CRC |
| CBUFF (LVDS) | xWR14xx R4F<br><br>xWR16xx R4F<br>xWR16xx DSP | Following features of IP are implemented in the driver:<br>LVDS driver supports the chirp and continuous mode of data transmission.<br>Supports only 2 and 4 lane configuration in Format0.<br>Supports transfer of S/W triggered user data over CBUFF/LVDS interface |
| CRC | xWR14xx R4F<br><br>xWR16xx R4F<br>xWR16xx DSP | All features of IP are implemented in the driver including:<br>CRC-16<br>CRC-32<br>CRC-64 |
| CRYPTO | xWR16xx (HS) R4F | The driver supports following AES mode of encryption:<br>▪ Electronic codebook mode (ECB)<br>▪ Cipher-block chaining mode (CBC)<br>▪ Cipher feedback mode (CFB)<br>▪ Counter mode (CTR)<br>▪ Integer counter mode (ICM)<br>▪ Galios/counter mode (GCM)<br>▪ Counter with CBC-MAC mode (CCM)<br><br>The driver supports following HMAC modes:<br>▪ MD5<br>▪ SHA-1<br>▪ SHA-224<br>▪ SHA-256 |
| CSI-2 | IWR14xx R4F | Following features of IP are implemented in the driver:<br>Initialization and Setup of the Protocol Engine<br>Initialization and configuration of the DSI PHY<br>DSI Phy Parameters can be customized by the application |

| | | |
|---|---|---|
| DMA | xWR14xx R4F<br>xWR16xx R4F | Following features of IP are implemented in the driver:<br>software and hardware triggered transfer<br>frame based transfer<br>block based transfer<br>Addressing mode (Constant, Indexed, Post Increment)<br>FTC, BTC, LFS, HBC interrupts<br>channel chaining<br>auto-initiation mode<br>interrupt based and polling based channel completion APIs |
| EDMA | xWR14xx R4F<br>xWR16xx R4F<br>xWR16xx DSP | All features of IP are implemented in the driver except "privilege" feature |
| ESM | xWR14xx R4F<br>xWR16xx R4F<br>xWR16xx DSP | Default ESM FIQ Interrupt handler for R4F and hook function for DSP's NMI<br>Provide application to register callback functions on specific ESM errors. |
| GPIO | xWR14xx R4F<br>xWR16xx R4F | All features of IP are implemented in the driver |
| HWA | xWR14xx R4F | All features of IP are implemented in the driver |
| I2C | xWR14xx R4F<br>xWR16xx R4F | All features of IP are implemented in the driver |
| MAILBOX | xWR14xx R4F<br>xWR16xx R4F<br>xWR16xx DSP | All features of IP are implemented in the driver. |
| OSAL | xWR14xx R4F<br>xWR16xx R4F<br>xWR16xx DSP | Provides an abstraction layer for some of the common OS services:<br>Semaphore<br>Mutex<br>Debug<br>Interrupts<br>Clock<br>Memory |
| PINMUX | xWR14xx R4F<br>xWR16xx R4F | All Pinmux fields can be set and all device pad definitions are available |
| QSPI | xWR14xx R4F<br>xWR16xx R4F | All features of IP are implemented in the driver. |

| | | |
|---|---|---|
| QSPIFLASH | xWR14xx R4F<br>xWR16xx R4F | All features of IP are implemented in the driver. |
| RTI | xWR14xx R4F<br>xWR16xx R4F | Part of TI RTOS offering |
| SOC | xWR14xx R4F<br>xWR16xx R4F<br>xWR16xx DSP | Provides abstracted APIs for system-level initialization. See section "mmWave SDK - System Initialization" for more details. |
| SPI (MIBSPI) | xWR14xx R4F<br>xWR16xx R4F | All features of IP are implemented in the driver including:<br>4-wire Slave and master mode<br>3-wire Slave and Master mode<br>both polling mode and DMA mode for read/write<br>char length 8-bit and 16-bit. |
| VIM | xWR14xx R4F<br>xWR16xx R4F | Part of TI RTOS offering |
| UART | xWR14xx R4F<br>xWR16xx R4F<br>xWR16xx DSP | All features of IP are implemented in the driver including:<br>Standard Baud Rates: 9600, 14400, 19200 till 921600<br>Data Bits: 7 and 8 Bits<br>Parity: None, Odd and Even<br>Stop Bits: 1 and 2 bits<br>Blocking and Polling API for reading and writing to the UART instance<br>DMA support for read/write APIs |
| WATCHDOG | xWR14xx R4F<br>xWR16xx R4F<br>xWR16xx DSP | All features of IP are implemented in the driver. |

***Table 2: Supported drivers and their functionality***

### 5. 4. 2. 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. The source code for the OSAL layer is present in the mmwave_sdk_<ver>\packages\ti\drivers\osal folder. Documentation of the APIs are available via doxygen and placed at mmwave_sdk_<ver>\packages\ti\drivers\osal\docs\doxygen\html\index.html. A sample porting of this OSAL for TI RTOS is provided as part of the mmWave SDK. System integrators could port the OSAL for their custom OS or customize the same TI RTOS port for their custom application, as per their requirements.

Examples of what integrators may want to customize:

- MemoryP module - for example, choosing from among a variety of heaps available in TI RTOS (SYSBIOS), or use own allocator.
- Hardware interrupt mappings. This case is more pronounced for the C674 DSP on xWR16xx which has only 16 interrupts (of which 12 are available under user control) whereas the events in the SOC are much more than 16. These events go to the C674 through an interrupt controller (INTC) and Event Combiner (for more information see the C674x megamodule user guide at http://www.ti.com/lit/ug/sprufk5a/sprufk5a.pdf). The default OSAL implementation provided in the release routes all events used by the drivers through the event combiner. If a user chooses to route differently (e.g for performance reasons), they may add conditional code in OSAL implementation to route specific events through the INTC and event combiner blocks. User can conveniently use event defines in ti/common/sys_common_*.h to acheive this.

### 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 R4F) and/or DSP subsystem (DSS C674x, xWR16xx only). 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.

- Link between application and Radar SS
- Handles communication errors, Notifies exceptions
- Platform and OS independent
- Can work in single threaded (non OS) environment

Following figure shows the various interfaces/APIs of the mmWaveLink component. The source code for mmWaveLink is present in the mmwave_sdk_<ver>\packages\ti\control\mmwavelink folder. Documentation of the API is available via doxygen and placed at mmwave_sdk_<ver>\packages\ti\control\mmwavelink\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_sdk_<ver>\packages\ti\control\mmwavelink\test\ .

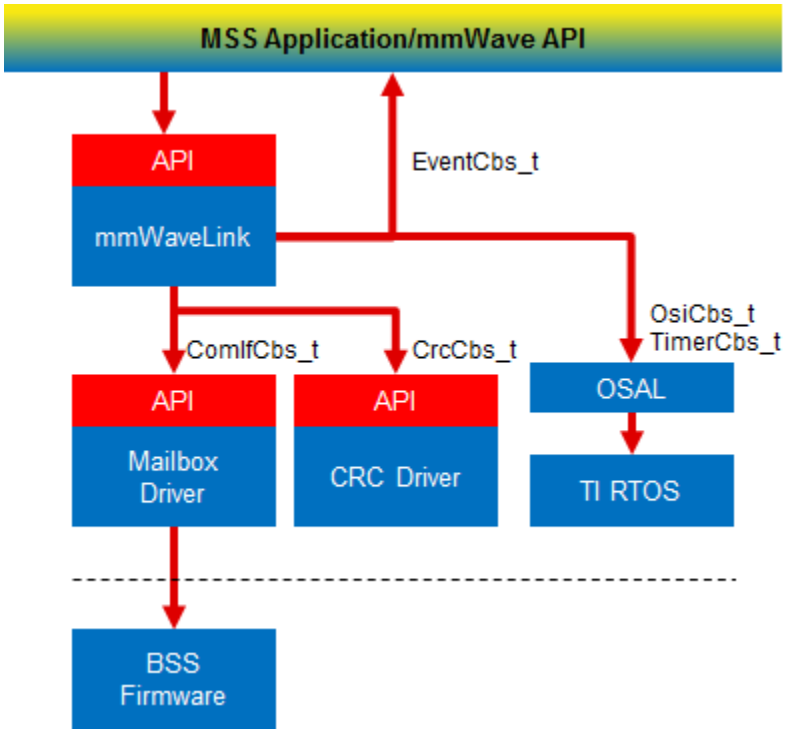



*Figure 20: mmWaveLink - Internal software design*

### 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 xWR16xx, 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_sdk_<ver>\packages\ti\control\mmwave folder. Documentation of the API is available via doxygen and placed at mmwave_sdk_<ver>\packages\ti\control\mmwave\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_sdk_<ver>\packages\ti\control\mmwave\test\.

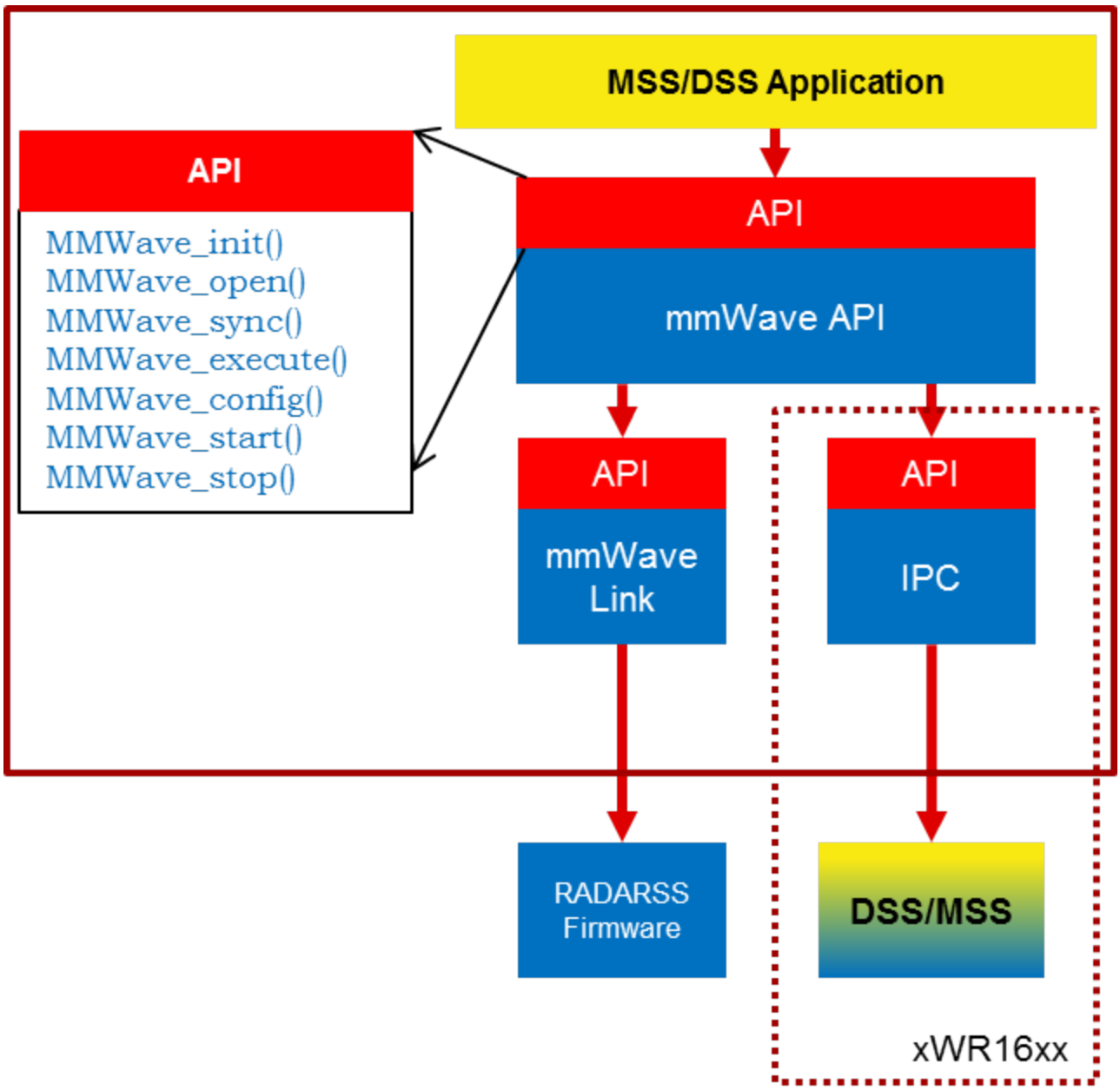



*Figure 21: mmWave API - Internal software design*

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

#### 5. 4. 4. 1. 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 on the xwr16xx
- 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

Currently API rlSetBpmChirpConfig and rlSetBpmCommonConfig are not handled by the mmWave API layer as part of the full configuration. Users desiring this functionality can call these APIs between MMWave_config and MMWave_start.

#### 5. 4. 4. 2. Minimal configuration

For advanced users, that either need to use advanced frame config of mmWave Front End or need to perform extra sequence of commands in the CONFIG routine, the minimal mode is recommended. In this mode the application has access to only a subset of services provided by the mmWave control module. These features includes:-

- Initialization of the mmWave Link
- Synchronization services between the MSS and DSS on the xr16xx
- Asynchronous Event Management
- Start & Stop services

In this mode the application is responsible for directly invoking the mmWave Link API in the correct order as per their configuration requirements. The configuration services are not available to the application; so in xWR16xx, the application is responsible for passing the configuration between the MSS and DSS if required.

See sample call flow below:

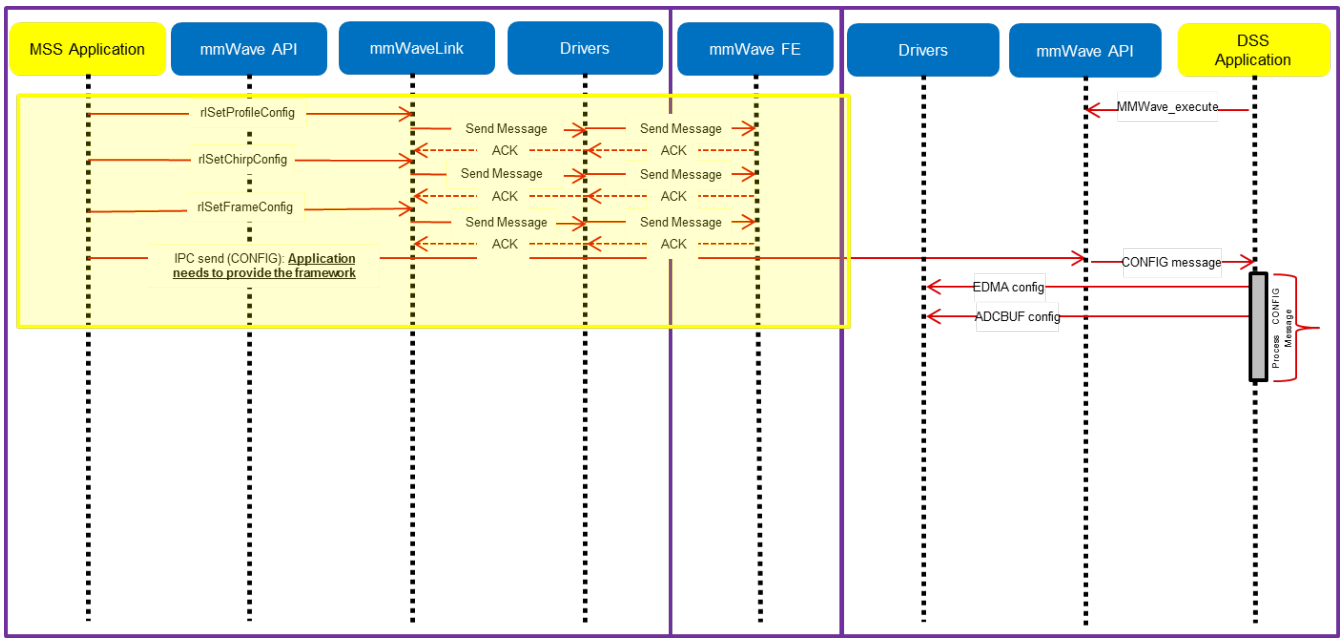



*Figure 22: mmWave API - 'Minimal' Config - Sample flow (xWR16xx)*

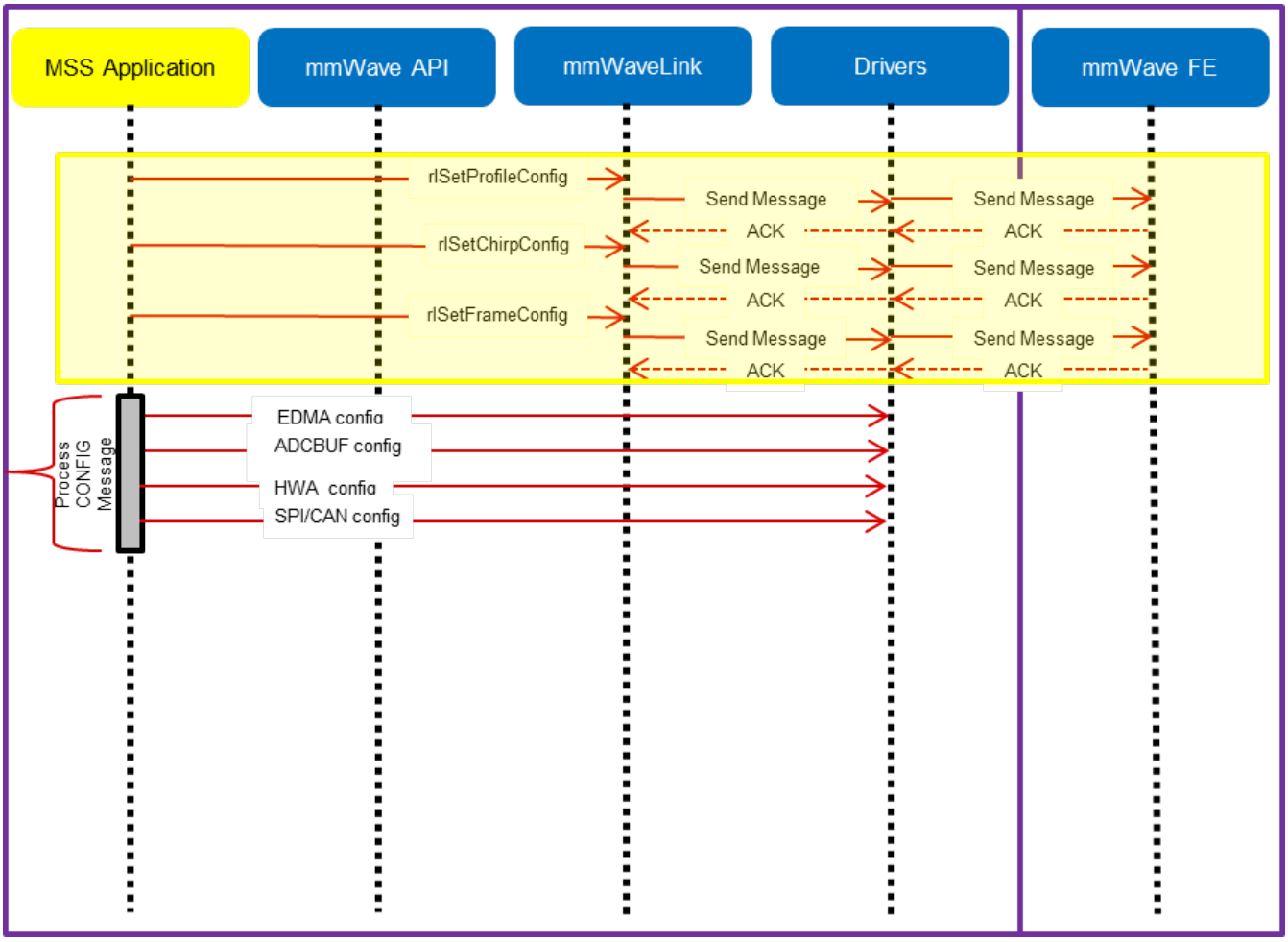



*Figure 23: mmWave API - 'Minimal' Config - Sample flow (xWR14xx)*

**mmWave Front End Calibrations**
mmWave API, by default, enables all init/boot time time calibrations for mmWave Front End. Moreover, 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_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. mmWaveLib

mmWaveLib is a collection of algorithms that provide basic functionality needed for FMCW radar-cube processing. This component is available for xWR16xx only and contains optimized library routines for C674 DSP architecture only. This component is not available for cortex R4F (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_sdk_<ver>\packages\ti\alg\mmwavelib folder. Documentation of the API is available via doxygen and placed at mmwave_sdk_<ver>\packages\ti\alg\mmwavelib\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_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
- Optimized and available for xWR16xx C674x DSP only
- 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.
- 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 power accumulation: accumulates signal powers in floating point version.
    - Histogram: mmwavelib_histogram right-shifts unsigned 16-bit vector and calculates histogram.
    - Right shift operation on signed16-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.
    - 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: multiplies a 16-bit complex vector and a 32-bit complex vector element by element, and outputs to the 32-bit complex output vector.
    - 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
- FFT utility: mmwavelib_dftSingleBinWithWindow calculates single bin DFT with windowing.

#### 5. 4. 6.

### 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 informations, 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 provided for xWR16xx device only but is supported for both R4F and C674x. The source code for gtrack is present in the mmwave_sdk_<ver>\packages\ti\alg\gtrack folder. Documentation of the API is available via doxygen and placed at mmwave_sdk_<ver>\packages\ti\alg\gtrack\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_sdk_<ver>\packages\ti\alg\gtrack\test\ .

### 5. 4. 7. RADARSS Firmware

This is a binary (mmwave_sdk_<ver>\firmware\radarss) that runs on Radar subsystem of the xWR14xx/xWR16xx and realizes the mmWave front end. It exposes configurability via a set of messages over mailbox which is understood by the mmWaveLink component running on the MSS. RADARSS firmware is responsible for configuring RF/analog and digital front-end in real-time, as well as to periodically schedule calibration and functional safety monitoring. This enables the mmWave front-end to be self-contained and capable of adapting itself to handle temperature and ageing effects, and to enable significant ease-of-use from an external host perspective. Features/enhancements information can be found under firmare/radarss/mmwave_radarss_release_notes.pdf

### 5. 4. 8. 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. For xWR14xx, this binary is for R4F (MSS) and for xWR16xx, there is an executable for both R4F (MSS) and C674 (DSS) and is combined into one metaImage for flashing along with RADARSS firmware. Note that the CCS debug application for C674 (DSS) has the L1 and L2 cache turned off so that new application that gets downloaded via CCS can enable it as needed, without any need for cache flush operations, etc during switching of applications. CCS debug for MSS (R4F) has the while loop implemented using ARM instruction set since its purpose is to allow users to load another application using CCS and the first instruction that the application would run will be _c_int00 which is compiled only in ARM mode.

### 5. 4. 9. 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_sdk_<ver>\packages\ti\utils\hsiheader folder. Documentation of the API is available via doxygen and placed at mmwave_sdk_<ver>\packages\ti\utils\hsiheader\docs\doxygen\html\index.html.

### 5. 4. 10. mmWave SDK - System Initialization

Application should call init APIs for the following system modules (ESM, SOC, Pinmux) to enable correct operation of the device

#### 5. 4. 10. 1. ESM

ESM_init should be the first function that is called by the application in its main(). Refer to the doxygen for this function at mmwave_sdk_<ver>\packages\ti\drivers\esm \docs\doxygen\html\index.html to understand the API specification.

#### 5. 4. 10. 2. SOC

SOC_init should be the next function that should be called after ESM_init. Refer to the doxygen for this function at mmwave_sdk_<ver>\packages\ti\drivers\soc \docs\doxygen\html\index.html to understand the API specification. It primarily takes care of following things:

DSP un-halt

This applies for xWR16xx only. Bootloader loads the DSP application from the flash onto DSP's L2/L3 memory but doesnt un-halt the C674x core. It is the responsibility of the MSS application to un-halt the DSP. SOC_init for xWR16xx MSS provides this functionality under its hood.

RADARSS un-halt/System Clock

To enable selection of system frequency to use "closed loop APLL", the SOC_init function unhalts the RADARSS and then spins around waiting for acknowledgement from the RADARSS that the APLL clock close loop calibration is completed successfully.

> Note that this function assumes that the crystal frequency is 40MHz.

MPU (Cortex R4F)

MPU or Memory Protection Unit needs to be configured on the Cortex R4F of xWR14xx and xWR16xx for the following purposes:

- Protection of memories and peripheral (I/O) space e.g not allowing execution in I/O space or writes to program (.text) space.
- Controlling properties like cacheability, buferability and orderability for correctness and performance (execution time, memory bandwidth). Note that since there is no cache on R4F, cacheability is not enabled for any region.

MPU has been implemented in the SOC module as a private function SOC_mpu_config() that is called by public API SOC_init(). Doxygen of SOC (mmwave_sdk_<ver>\packages\ti\drivers\soc \docs\doxygen\html\index.html) has SOC_mpu_config() documented with details of choice of memory regions etc. When MPU violation happens, BIOS will automatically trap and produce a dump of registers that indicate which address access caused violation (e.g DFAR which indicates what data address access caused violation). Note: The SOC function uses as many MPU regions as possible to cover all the memory space available on the respective device. There may be some free MPU regions available for certain devcies (ex: xWR14xx) for the application to use and program as per their requirement. See the function implementation/doxygen for more details on the usage and availability of the MPU regions.

> A build time option called DOWNLOAD_FROM_CCS has been added which when set to yes prevents program space from being protected. This option should be set to yes when debugging using CCS because CCS, by default, attempts to put software break-point at main() on program load which requires it to change (temporarily) the instruction at beginning main to software breakpoint and this will fail if program space is read-only. Hence the benefit of code space protection is not there when using CCS for download. It is however recommended to set this option to no when building the application for production so that program space is protected.

MARs (xWR16xx C674)

The cacheability property of the various regions as seen by the DSP (C674x in xWR16xx) is controlled by the MAR registers. These registers are programmed as per driver needs in in the SOC module as a private function SOC_configMARs() that is called by public API SOC_init(). See the doxygen documentation of this function to get more details. Note that the drivers do not operate on L3 RAM and HS-RAM, hence L3/HS-RAM cacheability is left to the application/demo code writers to set and do appropriate cache (writeback/invalidate etc) operations from the application as necessary, depending on the use cases. The L3 MAR is MAR32 -> 2000_0000h - 20FF_FFFFh and HS-RAM MAR is MAR33 -> 2100_0000h - 21FF_FFFFh.

#### 5. 4. 10. 3. Pinmux

Pinmux module is provided under mmwave_sdk_<ver>\packages\ti\drivers\pinmux with API documentation and available device pads located at mmwave_sdk_<ver>\packages\ti\drivers\pinmux\docs\doxygen\html\index.html. Application should call these pinmux APIs in the main() to correctly configure the device pads as per their hardware design.

> **TI Pinmux Utility**
> TI Pinmux Tool available at https://dev.ti.com/pinmux supports mmWave devices and can be used for designing the pinmux configuration for custom board. It also generates code that can be included by the application and compiled on top of mmWave SDK and its Pinmux driver.

### 5. 4. 11. Usecases

#### 5. 4. 11. 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 millimeter wave demo. The demo runs in real-time and has all the associated framework for RADARSS control etc with it.

The “HWA_EDMA” for xwr14xx and "DSP_EDMA" for xwr16xx tests (located at mmwave_sdk_<ver>\packages\ti\drivers \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 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 mmw demo documentation.

The “HWA_EDMA” and "DSP_EDMA" 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 “HWA_EDMA” and "DSP_EDMA" 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_sdk_<ver>\packages\ti\drivers\test\<test_dir>\docs\doxygen\html\index.html.

#### 5. 4. 11. 2. CSI-2 based streaming of ADC data

IWR14xx device has a high speed CSI-2 transmit interface that can be used to ship ADC data or 1D/2D processed data out of the device. An example usecase on how to program the front end to generate the ADC samples and tie it up to CBUFF/CSI-2 interface for data shipment is provided under mmwave_sdk_<ver>\packages\ti\drivers\test\csi_stream. Refer to the doxygen documentation located at mmwave_sdk_<ver>\packages\ti\drivers\test\csi_stream\docs\doxygen\html\index.html for more details.

#### 5. 4. 11. 3. Basic configuration of Front end and capturing ADC data in L3 memory

To access ADC data from mmWave sensors, user need to program various basic components within the device in a given sequence. In order to help user understand the programming model needed to configure the device and generate ADC data in device's L3 memory, an example usecase is provided under mmwave_sdk_<ver>\packages\ti\drivers\test\mem_capture. Refer to the doxygen documentation located at mmwave_sdk_<ver>\packages\ti\drivers\test\mem_capture\docs\doxygen\html\index.html for more details.

# 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 .xer4f/.xe674 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_sdk_<ver>\packages\ti\demo\<platform>\mmw\docs\doxygen\html\index.html documentation.

## 6. 2. Register layout

The register layout of the device is available inside each hardware IP's driver source code. See mmwave_sdk_<ver>\packages\ti\drivers\<ip>\include\reg_*.h. The system level registers (RCM, TOPRCM, etc) are available under the SOC module (mmwave_sdk_<ver>\packages\ti\drivers\soc\include\reg_*.h).

## 6. 3. Enable DebugP logs

The DebugP_log OSAL APIs in ti/drivers/osal/DebugP.h are used in the drivers and test/app code for debug streaming. These are tied to BIOS's Log_* APIs and are well documented in SYSBIOS documentation. The logs generated by these APIs can be directed to be stored in a circular buffer and observed using ROV in CCS (http://rtsc.eclipse.org/docs-tip/Runtime_Object_Viewer).

Following steps should be followed to enable these logs:

1. Enable the flag DebugP_LOG_ENABLED before the header inclusion as seen below.

```
#define DebugP_LOG_ENABLED 1
#include <ti/drivers/osal/DebugP.h>
```

2. Add the following lines in your SYSBIOS cfg file with appropriate setting of numEntries (number of messages) which will impact memory space:

**Application SYSBIOS cfg file**

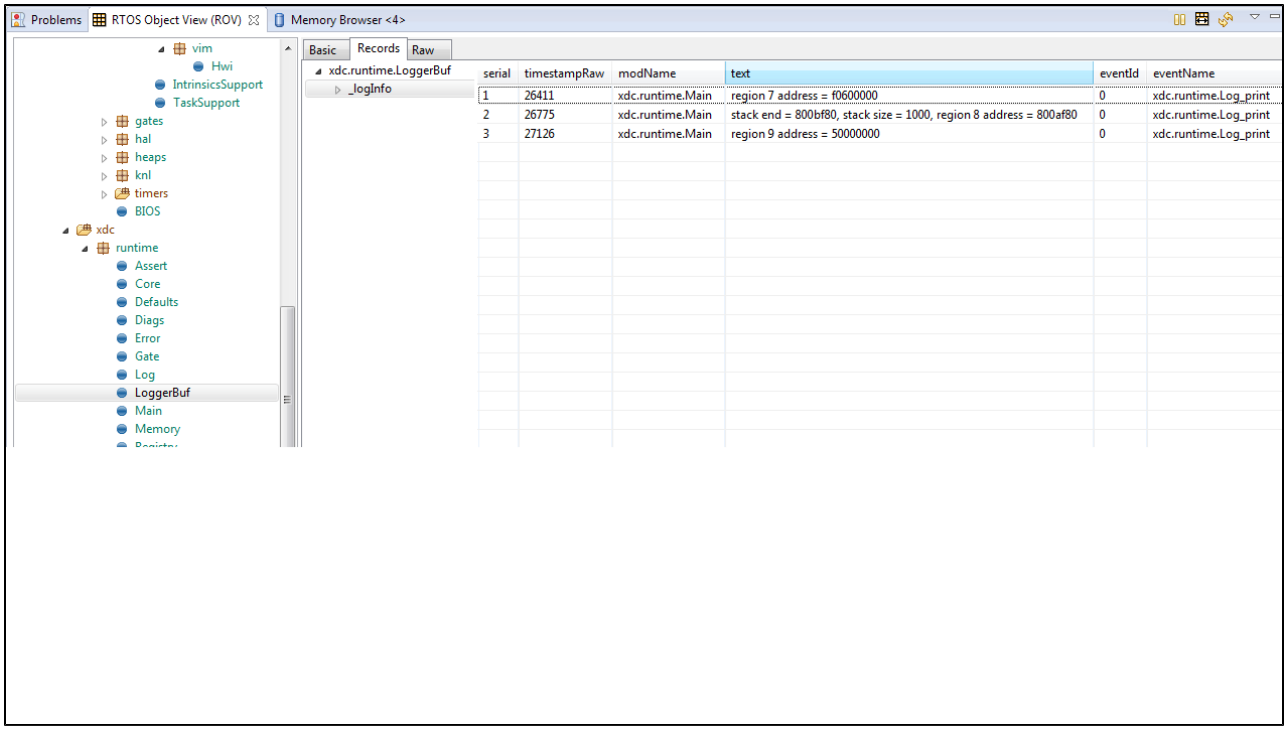
```
var Log       = xdc.useModule('xdc.runtime.Log');
var Main = xdc.useModule('xdc.runtime.Main');
var Diags = xdc.useModule('xdc.runtime.Diags');
var LoggerBuf                   = xdc.useModule('xdc.runtime.LoggerBuf');
LoggerBuf.TimestampProxy        = xdc.useModule('xdc.runtime.Timestamp');

/* Trace Log */
var loggerBufParams = new LoggerBuf.Params();
loggerBufParams.bufType = LoggerBuf.BufType_CIRCULAR; //BufType_FIXED
loggerBufParams.exitFlush = false;
loggerBufParams.instance.name = "_logInfo";
loggerBufParams.numEntries = 100; <-- number of messages this will affect memory consumption
// loggerBufParams.bufSection = ;
_logInfo = LoggerBuf.create(loggerBufParams);
Main.common$.logger = _logInfo;

/* Turn on USER1 logs in Main module (all non-module functions) */
Main.common$.diags_USER1 = Diags.RUNTIME_ON;

/* Turn on USER1 logs in Task module */
Task.common$.diags_USER1 = Diags.RUNTIME_ON;
```

A sample ROV log looks like below after code is re-build and run with above changes :




## 6. 4. Shared memory usage by SDK demos (xWR1642)

Existing SDK demos (mmw) for xWR1642 assigns all 6 banks of L3 memory to DSS (i.e. 768KB). No additional banks are added to MSS TCMA and TCMB; they remain at the default memory size. See TRM for more details on the L3 memory layout and "xWR1xxx Image Creator User Guide" in SDK for more details on shared memory allocation when creating flash images. Note that the image that is programmed into the flash of the xWR1642 device determines the shared memory allocation. So in CCS development mode, it is the allocation defined in ccsdebug metaImage that applies and not the application that you download via CCS.

In SDK code, you can relate to these settings in the following places:

- Environment variable MMWAVE_L3RAM_SIZE that takes L3 memory size in bytes (in hex format) and controls
  - the size of L3 memory section in linker command files (mmwave_sdk_<ver>\packages\ti\platform\xWR16xx)
  - the sys_common defines (SOC_XWR16XX_DSS_L3RAM_SIZE, SOC_XWR16XX_MSS_L3RAM_SIZE) for the application code to use and size the buffers, heaps, etc accordingly.
- Makefiles for the following components use the value 0x00000006 for SHMEM_ALLOC parameter (i.e. 6 banks for DSS) when invoking the generateMetaImage script. (See xWR1xxx Image Creator section). User should manually change these values for the metaImage script when using different value for the environment variable MMWAVE_L3RAM_SIZE.
  - mmwave_sdk_<ver>\packages\ti\utils\ccsdebug
  - mmwave_sdk_<ver>\packages\ti\demo\xwr16xx\mmw
- Since there is a chance for MMWAVE_L3RAM_SIZE and metaImage bank allocation to go out of sync (due to user error), SOC module init does a sanity check of the hardware programmed L3 bank allocations (that are fed via metaImage header) and the sys_common defines. If the sys_common defined L3 size is greater than hardware programmed bank allocations, the module throws an assert.

## 6. 5. xWR1xxx 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 xWR1xxx 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, follow the steps below to create the flash images based on the mmWave device that it is intended for.

### 6. 5. 1. xWR14xx

Use the generateBin script present under mmwave_sdk_<ver>\packages\ scripts\windows or mmwave_sdk_<ver>\packages\ scripts\linux for the conversion of out file to bin for the MSS R4F. Set **MMWAVE_SDK_DEVICE=awr14xx** or **MMWAVE_SDK_DEVICE=iwr14xx** and **MMWAVE_SDK_INSTALL_PATH=mmwave_sdk_<ver>\packages** in your environment before calling this script.This script needs 3 parameters:

- .out file : this is the input file and the parameter represents the file generated after the link step for the application (application executable). (this can be with file path)
- .bin file : this is the output file generated by the script and this parameter represents the filename for the flash binary (this can be with file path)
- offset : this is the offset for the MSS image section needed by the Bootloader and should be 0x200000

The .bin file generated by this script should be used for the MSS_BUILD during flashing step (How to flash an image onto xWR14xx/xWR16xx EVM)

**Unaligned sections**
MSS Bootloader for xWR14xx requires that the loadable sections be aligned to 16 bytes using ALIGN(16) in the linker command file. If this is not done, then out2rprc.exe (called from generateBin script) will throw following error and no bin file will be generated.

**Unaligned sections: File conversion failure**

```
Parsing the input object file, xwr14xx_mmw_demo_mss.xer4f.
Appending zeros 0
Appending zeros 256
Appending zeros 125192
Unaligned section 125192
File conversion failure!
```

### 6. 5. 2. xWR16xx

Application Image generation is two-step process for xWR16xx. Refer to "xWR1xxx Image Creator User Guide" in the SDK docs directory for details on the internal layout and format of the files generated in these steps.

1. RPRC format conversion:
   Firstly, application executable has to be converted from ELF/COFF format to custom TI RPRC image format.
   Use the generateBin script present under mmwave_sdk_<ver>\packages\ scripts\windows or mmwave_sdk_<ver>\packages\ scripts\ linux for the conversion of out file to bin for the MSS R4F and for the DSS C674 (need to run the script twice). Set **MMWAVE_SDK_ DEVICE=awr16xx** or **MMWAVE_SDK_DEVICE=iwr16xx** and **MMWAVE_SDK_INSTALL_PATH=mmwave_sdk_<ver>\packages** i n your environment before calling this script. This script needs 2 parameters:
   - executable (.xer4f or .xe674) file : this is the input file and the parameter represents the file generated after the link step for the application (application executable).
   - binary (.bin) file : this is the output file generated by the script and this parameter represents the filename for the flash binary
2. Multicore Image file generation:
   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_sdk_<ver>\packages\ scripts\windows or mmwave_sdk_<ver>\packages\ scripts\linux for merging the MSS, DSS and RADARSS binaries into one metaImage and appending correct CRC. Set **MMWAVE_S DK_INSTALL_PATH=mmwave_sdk_<ver>\packages** in your 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
   - SHMEM_ALLOC:[input] shared memory allocation in 32-bit hex format where each byte (left to right) is the number of banks needed for RADARSS (BSS),TCMB,TCMA and DSS. Refer to the the TRM on details on L3 shared memory layout and "xWR1xxx Image Creator User Guide" in the SDK.
   - MSS_IMAGE: [input] MSS input image in RPRC (,bin) format as generated by generateBin script in the step above, use keyword NULL if not needed
   - BSS_IMAGE: [input] RADARSS (BSS) input image in RPRC (,bin) format, use keyword NULL if not needed. Use mmwa ve_sdk_<ver>\firmware\radarss\xwr16xx_radarss_rprc.bin here.
   - DSS_IMAGE: [input] DSP input image in RPRC (,bin) format as generated by generateBin script in the step above, use keyword NULL if not needed

The FLASHIMAGE file generated by this script should be used for the METAIMAGE1 during flashing step (How to flash an image onto xWR14xx/xWR16xx EVM)

## 6. 6. xWR16xx mmw Demo: cryptic message seen on DebugP_assert

In mmw demo, the BIOS cfg file dss_mmw.cfg has below code at the end to optimize BIOS size. Because of some of these changes,

exceptions, such as those generated through DebugP_assert() calls may give a cryptic message instead of file name and line number that helps identify easily where the exception is located. To be able to restore this capability, the user can comment out the lines marked with the comment "" below. For more information, refer to the BIOS user guide.

```
/* Some options to reduce BIOS code and data size, see BIOS User Guide section
   "Minimizing the Application Footprint" */
System.maxAtexitHandlers = 0; /* COMMENT THIS FOR FIXING DebugP_Assert PRINTS */
BIOS.swiEnabled = false; /* We don't use SWIs */
BIOS.libType = BIOS.LibType_Custom;
Task.defaultStackSize = 1500;
Task.idleTaskStackSize = 800;
Program.stack = 1048; /* for isr context */
var Text = xdc.useModule('xdc.runtime.Text');
Text.isLoaded = false;
```

## 6. 7. How to execute Idle instruction in idle task when using SYSBIOS

The idle function hook provided by SYSBIOS can be used to install application specific function which in turn could call the "idle" asm instruction. See code snapshots below or refer to mmW demo for details.

**BIOS CFG file**

```
var Idle      = xdc.useModule('ti.sysbios.knl.Idle');
Idle.addFunc('&MmwDemo_sleep');
```

**WFI instruction for R4F**

```
void MmwDemo_sleep(void)
{
    /* issue WFI (Wait For Interrupt) instruction */
    asm(" WFI ");
}
```

**IDLE instruction for C674x**

```
void MmwDemo_sleep(void)
{
    /* issue IDLE instruction */
    asm(" IDLE ");
}
```

## 6. 8. Range Bias and Rx Channel Gain/Offset Measurement and Compensation

Refer to the section "Range Bias and Rx Channel Gain/Offset Measurement and Compensation" in the mmwave_sdk_<ver>\packages\ti\demo\xwr16xx\mmw\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_sdk_<ver>\packages\ti\demo\xwr16xx\mmw\profiles\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_sdk_<ver>\packages\ti\demo\xwr16xx\mmw\profiles\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 "Range/Angle Bias compensation config" textbox in the Visualizer and any new profile generated by the Visualizer will use these values. Alternatively, you can add this to your custom profile configs and use it via the "Load Config From PC and Send" button.

## 6. 9. 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 R4F, 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.**R4Ft** target in the mmwave_sdk_<ver>\packages\ti\ common\mmwave_sdk.mak.
2. On C674X, compile portions of code that are not in compute critical path with appropriate -msX option. The -ms0 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 xWR16xx, using ms0 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 "ms" 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 C674X tends to have a bigger footprint of control code than the same C code compiled on R4F. So if feasible, partition the software to use C674X mainly for compute intensive signal-processing type code and keep more of the control code on the R4F. An example of this is in the mmw demo, where we show the usage of mmwave API to do configuration (of RADARSS) from R4F instead of the C674X (even though the API allows usage from either domain). In mmw demo, this prevents linking of `mmwave` (in mmwave_sdk_<ver>\packages\ ti\control ) and mmwavelink (in mmwave_sdk_<ver>\packages\ti\control ) code that is involved in configuration (profile config, chirp config etc) on the C674X side as seen from the .map files of mss and dss located at ti/demo/xwr16xx/mmw.
4. If using TI BIOS as the operating system, depending on the application needs for debug, safety etc, the BIOS footprint in the application may be reduced by using some of the techniques listed in the BIOS User Guide in the section "Minimizing the Application Footprint". Some of these are illustrated in the mmw demo on R4F and C674X. Some common ones are disabling system_printf (printf strings do contribute to significant code size), choosing sysmin and using ROV for debugging, disabling assert (although this should be done only when variability in driver configuration is not expected and existing configuration has been proven to function correctly). The savings from these features could be anywhere from 2KB to 10KB but user would lose some perks of debuggability.
5. 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. This is illustrated in the mmw demo on C674X code for xWR16xx and on R4F code for xWR14xx where such code is overlaid with (load time uninitialized) radar cube data in L3 RAM, refer to the file mmwave_sdk_<ver>\packages\ti\demo\xwr16xx\mmw\dss\dss_mmw_linker.cmd and mmwave_sdk_<ver>\packages\ ti\demo\xwr14xx\mmw\mss_mmw_linker.cmd (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 R4F due to bootloader limitation but can be used on C674X. The SDK uses ram model when building C674X executable images (unit tests and demos).
6. On C674X, 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. On the mmw demo, the L1D SRAM is used to allocate some buffers involved in data path processing whereas the L1P SRAM has 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.
7. 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 and L3 heaps on 'sensorStart'. These values can then be used to resize or re-allocate gMmwL1, gMmwL2 and gMmwL3 in mmwave_sdk_<ver>\packages\ti\demo\xwr16xx\mmw\dss\dss_data_path.c. The freed up space could be used as follows:
    a. Free heap space in L1D could be used to move some of the L2 buffers to L1D. Refer to section "EDMA versus Cache based Processing" in mmwave_sdk_<ver>\packages\ti\demo\xwr16xx\mmw\docs\doxygen\html\index.html documentation for overview of data buffer layout. The freed L2 space can be used for code/data.
    b. Free heap space in L2 could be trimmed by changing the 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 gMmwL3 definition and used for code/data space. One caveat is that L3 in mmW demo is un-cached and hence one could see some cycles impact but this is a decision each application developer can make depending on the memory and compute (MIPS) needs.

When using TI compilers for both R4F and C674x, 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
       ------                          ----     -------   -------


    obj_xwr14xx/
       main.oer4f                      5191     0         263980
       data_path.oer4f                 8441     0         65536
       config_hwa_util.oer4f           4049     0         0
       post_processing.oer4f           2480     0         0
       mmw_cli.oer4f                   2308     0         0
       config_edma_util.oer4f          1276     0         0
       sensor_mgmt.oer4f               1144     0         24
    +--+-------------------------------+--------+---------+---------+
       Total:                          24889    0         329540
```

## 6. 10. How to add a .const (table) beyond L3 heap in mmW demo where overlay is enabled

In mmW demo for xWR16xx, L3 heap is overlaid with the code to be copied into L1P at init time. To achieve this overlay, 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 dss_data_path.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[SOC_XWR16XX_DSS_L3RAM_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 dss_mmw_linker.cmd file:
    .l3data_garray:  load=L3SRAM PAGE 0 (HIGH)
```

## 6. 11. DSPlib integration in xWR16xx C674x application (Using 2 libraries simultaneously)

The TI C674X DSP is a merger of C64x+ (fixed point) and C67x+ (floating point) DSP architectures and DSPlib offers two different flavors of library for each of these DSP architectures. An application on C674X may need functions from both architectures. Normally this would be a straight-forward exercise like integrating other TI components/libraries. However there is a problem during integration of the two DSPLib libraries in the same application since the top level library API header `dsplib.h` has the same name and same relative path from the packages/ directory as seen below in the installation:

```
C:\ti\dsplib_c64Px_3_4_0_0\packages\ti\dsplib\dsplib.h
C:\ti\dsplib_c674x_3_4_0_0\packages\ti\dsplib\dsplib.h
```

Typically when integrating TI components, the build paths are specified up to `packages\` directory and headers are referred as below:

```
#include <ti/dsplib/dsplib.h>
```

However this will create an ambiguity when both libraries are to be integrated because the above path is same for both. There are a couple of ways to resolve this:

### 6. 11. 1. Integrating individual functions from each library

In this case, the headers individual functions are included in the application source file and the build infrastructure (makefiles for example) refers to the paths to the individual functions. This style of integration is illustrated in the mmw demo code as seen in the following code snippets: (Note: the mmw demo only uses one (C64P) dsplib so it could have been integrated in the straight-forward way but it is deliberately done this way to illustrate the method in question here and allows for future integration with C674x dsplib).

**Sample DSPLib integration using individual functions**

In file dss_mmw.mak:

**dss_mmw.mak**

```
dssDemo: C674_CFLAGS += --cmd_file=$(BUILD_CONFIGPKG)/compiler.opt \
      /* include path for DSP_fft16x16    */    \

-i$(C64Px_DSPLIB_INSTALL_PATH)/packages/ti/dsplib/src/DSP_fft16x16/c64P \
                        /* include path for DSP_fft32x32    */    \
                        -i$(C64Px_DSPLIB_INSTALL_PATH)/packages/ti/dsplib/src/DSP_fft32x32/c64
\
                        -i$(C674x_MATHLIB_INSTALL_PATH)/packages \
```

In dss_data_path.c:

**dss_data_path.c**

```
#include "DSP_fft32x32.h"
#include "DSP_fft16x16.h"
```

The C674P library can be integrated in the above code similar to the how the C64P has been done, this will not create any conflict.

A variant (not illustrated in mmw demo) of the above could be as follows where the paths are now in the .c and .mak only refers to the installation:

**dss_mmw.mak**

```
dssDemo: C674_CFLAGS += --cmd_file=$(BUILD_CONFIGPKG)/compiler.opt \
                        -i$(C64Px_DSPLIB_INSTALL_PATH)/packages \
                        -i$(C674x_MATHLIB_INSTALL_PATH)/packages \
```

**dss_data_path.c**

```
#include <ti/dsplib/src/DSP_fft16x16/c64P/DSP_fft32x32.h>
#include <ti/dsplib/src/DSP_fft16x16/c64P/DSP_fft16x16.h>
```

### 6. 11. 2. Patching the installation

The previous method can get cumbersome if there are many functions to be integrated from both libraries. Patching the installation to rename/duplicate the top level API header `dsplib.h` allows a straight-forward integration. This prevents the name conflict of the two headers. So the installation after patching would look like below for example:

```
C:\ti\dsplib_c64Px_3_4_0_0\packages\ti\dsplib\dsplib_c64P.h [one can retain the older dsplib.h if one wants
to]
C:\ti\dsplib_c674x_3_4_0_0\packages\ti\dsplib\dsplib_c674x.h [one can retain the older dsplib.h if one
wants to]
```

And the .mak and code will look like below:

**Sample DSPLib integration after renaming header files**

In file dss_mmw.mak:

**dss_mmw.mak**

```
dssDemo: C674_CFLAGS += --cmd_file=$(BUILD_CONFIGPKG)/compiler.opt \
                        -i$(C64Px_DSPLIB_INSTALL_PATH)/packages \ <-- C64P dsplib
                        -i$(C674x_DSPLIB_INSTALL_PATH)/packages \ <-- C674x dsplib
                        -i$(C674x_MATHLIB_INSTALL_PATH)/packages \
```

In dss_data_path.c:

**dss_data_path.c**

```
#include <ti/dsplib/dsplib_c64P.h>
#include <ti/dsplib/dsplib_c674x.h>
```

The present dsplibs do not have name conflicts among their functions so they can both be integrated in the above manner.

## 6. 12. SDK Demos: miscellaneous information

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

- In xWR14xx, when 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.
- Demo's rov.xs file is provided in the SDK package to facilitate the CCS debugging of pre-built binaries when demo is directly flashed onto the device (instead of loading via CCS).
- When using non-interleaved mode for ADCBuf, the ADCBuf offsets for every RX antenna/channel enabled need to be multiple of 16 bytes.
- 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. 13. Data size restriction for a given session when sending data over LVDS

For the current implementation of the CBUFF/LVDS driver and its intended usage, the CBUFF data size for a given session needs to be multiple of 8.

User should take care of this restriction when writing their custom application using the SDK LVDS driver. This alignment is taken care by the HSI header library if the application uses the headers for LVDS streaming. If no header are used while streaming data over LVDS lanes, user should calculate the total data size in bytes for the hardware triggered session (i.e. per chirp) and make sure it follows the rules mentioned above. Similar rules apply for the user data sent during the software triggered session.

## 6. 14. CCS Debugging of real time application

### 6. 14. 1. 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.

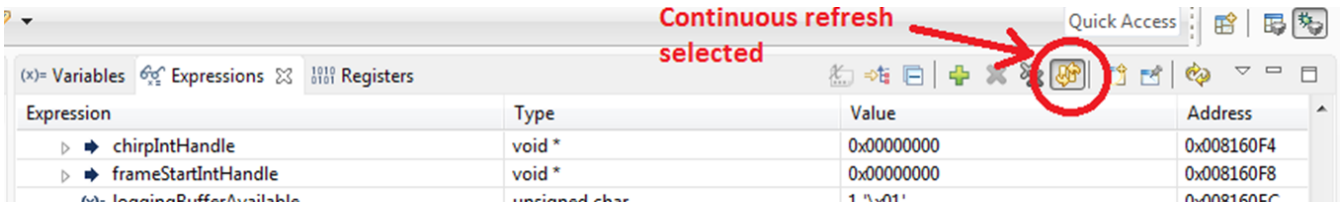
### 6. 14. 2. Using printfs in real time

This applies to SYSBIOS and debugging using CCS. Once the application starts real-time processing (i.e. once sensor start is issued), there should ideally be no prints on the console because CCS will halt the processor (unless CIO is disabled) on which such prints are issued for as long as it takes it to transfer the print string data from target to PC over JTAG and print the string on the PC (which can be of the order of seconds). This is true for any real-time application that uses SYSBIOS on any SoC (not just mmWave SDK/devices). For logging in real-time, SYSBIOS offers other options like LOG module, etc - although these will incur some memory overheads. For example, see "Enable DebugP logs" section. It is also possible in cfg file of SYSBIOS based application to direct System_printfs to an internal log buffer (circular or saturate) which will also prevent the hiccup by CCS (See 'xdc.runtime. SysMin' in SYSBIOS/XDC).

### 6. 14. 3. 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:**




**Crash in Console window:**

```
[C674X_0] Debug: Logging UART Instance @00815560 has been opened successfully
Debug: DSS Mailbox Handle @0080f550
Debug: MMWDemoDSS create event handle succeeded
Debug: MMWDemoDSS mmWave Control Initialization succeeded
Debug: MMWDemoDSS ADCBUF Instance(0) @00815530 has been opened successfully
Debug: MMWDemoDSS Data Path init succeeded
Debug: MMWDemoDSS initTask exit
[CortexR4_0] ******************************************
Debug: Launching the Millimeter Wave Demo
******************************************
Debug: MMWDemoMSS Launched the Initialization Task
Debug: MMWDemoMSS mmWave Control Initialization was successful
Debug: CLI is operational
Sensor has been stopped
Debug: MMWDemoMSS Received CLI sensorStart Event
[C674X_0] Heap L1 : size 16384 (0x4000), free 2816 (0xb00)
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=0xfffffdf5
A20=0xfffffff0 A21=0xffffffaa
A22=0xa6 A23=0xffffff60
A24=0xfffffea7 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=0xfffffecb
B8=0x0 B9=0xf00218
B10=0xffffffeee B11=0xffffffa7
```

C674x CPU Exception

**Enable "Silicone Real-time" mode:**