MMWAVE SDK User Guide

Product Release 0.8.0.0 Beta

Release Date: Jan 10, 2017

Document Version: 1.0

COPYRIGHT

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. System Overview

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

1. 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
- BSS Firmware
- Board Setup and Flash Utilities

1. 2. mmWave Demos

SDK provides a suite of 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

- mmWave Processing Demo
- ADC Data Capture demo
- ADC Data Streaming demo

1. 3. External Dependencies

The SDK depends on the following external components which are not part of the SDK package but will be needed to integrate the mmWave SDK.

- TI RTOS (or Custom RTOS)
- XDC Tools (if building for TI RTOS)
- CCS (for debugging)
- TI ARM and C674X compiler
- DSPLib and Mathlib

Please refer to the mmWave SDK Release Notes for detailed information on these external dependencies.

2. System Deployment

2. 1. AR14xx

A typical mmWave application using AR14xx 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 AR14xx system deployments could be envisioned as follows:

1. **Autonomous AR14xx sensor (aka Standalone mode)**
    a. AR14xx program code is downloaded from the serial flash memory attached to AR14xx (via QSPI)
    b. Optional high level control from remote entity
    c. Sends low speed data output (objects detected) to remote entity
2. **Hybrid AR14xx sensor + Controller (Industrial Applications)**
    a. Serial flash is attached/in-built to external controller and SPI interface exists between AR14xx 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 AR14xx sensor + DSP**
    a. Program code is either in serial flash memory attached to AR14xx (via QSPI) or downloaded via the control interface between AR14xx 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 1 and Figure 2.

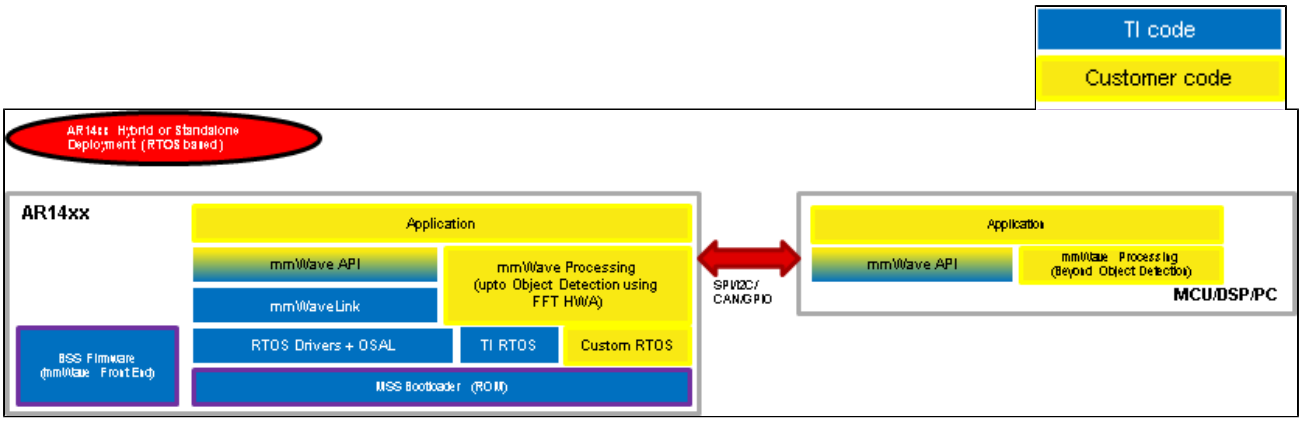


Figure 1: AR14xx Deployment in Hybrid or Standalone mode

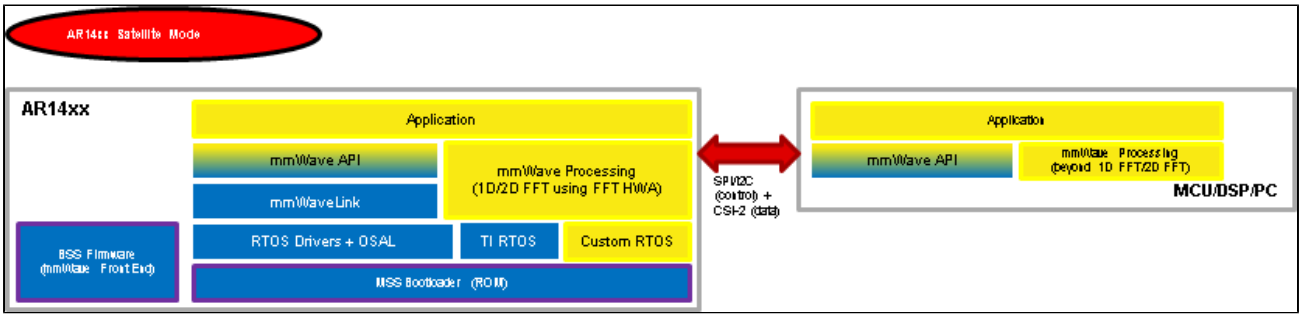


Figure 2: AR14xx Deployment in Satellite mode

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.

2. 2. AR16xx

A typical AR16xx 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 AR16xx customer deployment is shown in Figure 3:

a. AR16xx program code for MSS and DSP-SS is downloaded from the serial flash memory **attached** to AR16xx (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

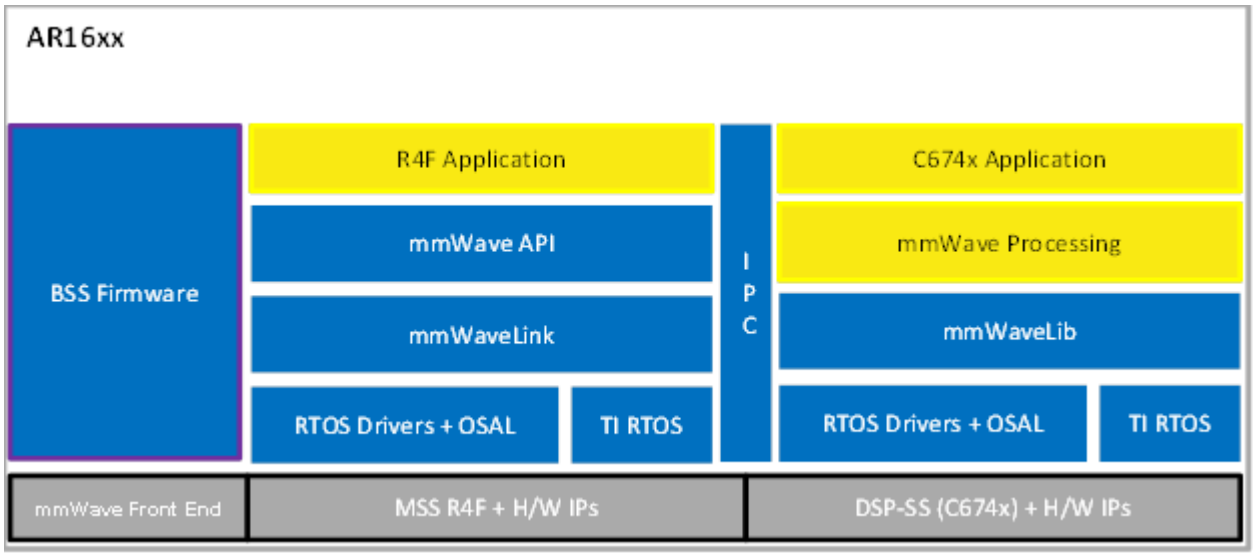


Figure 3: Autonomous AR16xx sensor (Standalone mode)

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. The demos are placed at <mmwave_sdk_package>/packages/ti/demo/<platform> folder. Currently following demos are supported within the SDK:

1. **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 AR14XX/AR16XX 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
2. **Capture Demo**: This demo is located at mmwave_sdk_<ver>/packages/ti/demo/<platform>/capture folder. The capture demo shows some of the radar sensing capabilities of the AR14XX/AR16XX SoC using the drivers in the mmWave SDK (Software Development Kit). It allows user to specify the chirping profile and on successful execution captures ADC data in the device's L3 memory. The data captured depends on the frame configuration provided and the amount of L3 memory that is available on the device. In AR14xx, the data buffer in L3 is implemented as a circular buffer and memory will be overwritten if the frame configuration produces more data than that can fit in L3 memory. In AR16xx, the data buffer in L3 is a linear buffer and capture stops when the buffer is full. Typically this demo should be used to capture only one frame worth of data by specifying number of frame parameter to be 1 in the cli's frameCfg command. Another constraint is that the interchirp time should be more than 10 usec to allow for DMA to copy the data from ADC buffer to L3 at every chirp available interrupt.
   In AR16xx, this demo has additional commands to enable the streaming of ADC data over LVDS lanes.
   A detailed explanation of this demo is available in the demo's docs folder: mmwave_sdk_<ver>\packages\ti\demo\<platform>\capture\docs\doxygen\html\index.html
3. **Streaming Demo**: This demo applies for AR14xx only and is located at mmwave_sdk_<ver>/packages/ti/demo/ar14xx/stream folder. The streaming demo shows some of the radar sensing capabilities of the AR14XX SoC using the drivers in the mmWave SDK (Software Development Kit) and how the raw sensor data can be shipped out over LVDS/CSI2 interface. It allows user to specify the chirping profile. There is no processing done on the AR14xx SOC. Internally, the data is also copied (via DMA) into the L3 memory to verify that the data transferred on the LVDS/CSI2 interface matches the ADC samples generated on the AR14xx. Note that this demo needs AR14xx to be interfaced with another device that has a compatible LVDS/CSI2 interface.

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

3. 1. Connecting the AR14xx/AR16xx EVM to PC

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 EVM User Guide for details on the COM port. 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 TI code composer studio is not installed on that PC, then XDS110 drivers need to be installed.
2. If TI code composer studio is installed, then version of CCS and emulation package need to be checked 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 COM7 and COM8 in the Figure below

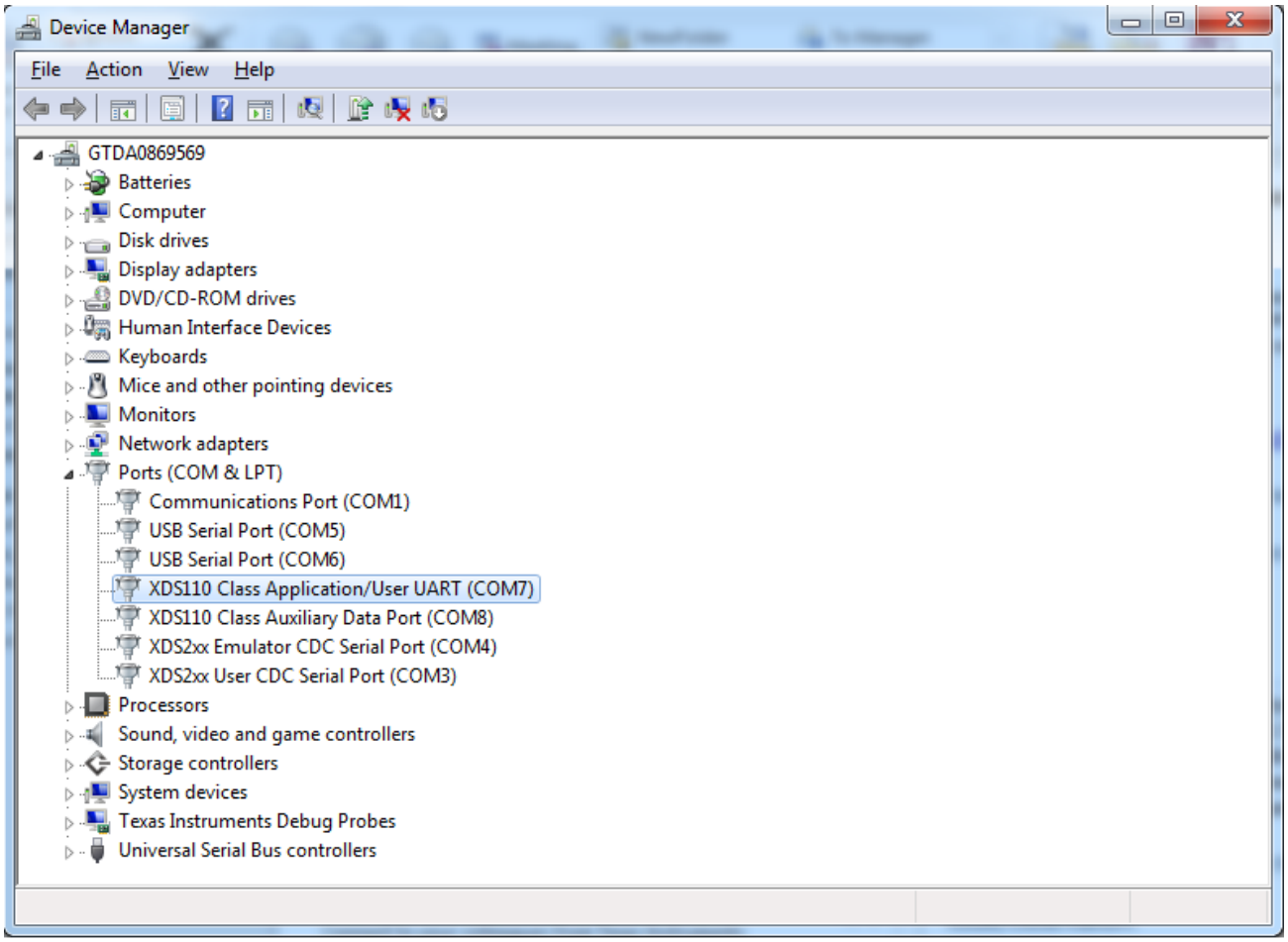


Figure 4: AR14xx/AR16xx PC Connectivity - Device Manager - COM Ports

> 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.

3. 2. Programming AR14xx/AR16xx

AR14xx

AR14xx 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.

AR16xx

AR16xx 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 execuable can be loaded and run on MSS R4F without downloading any code on DSS. Similarly, DSS unit test executable 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"

3. 3. Loading images onto AR14xx/AR16xx EVM

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

3. 3. 1. Demonstration Mode

This mode should be used when either experimenting with the prebuilt 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 AR14xx/AR16xx EVM) .
    a. For AR16xx: use the mmwave_sdk_<ver>/packages/ti/demo/ar16xx/<demo>/ar16xx_<demo>.bin as the METAIMAGE1

file name.
b. For AR14xx: use the mmwave_sdk_<ver>/packages/ti/demo/ar14xx/<demo>/ar14xx_<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. 3. 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

> A little background: When the AR1xxx 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 BSS firmware and/or DSS application) to be present on the flash. During AR1xxx 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 (AR16xx only) directly via CCS in the AR14xx/AR16xx 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 AR14xx/AR16xx EVM .
      i. For AR16xx: use mmwave_sdk_<ver>/packages/ti/utils/ccsdebug /ar16xx_ccsdebug.bin as the METAIMAGE1 filename for the one-time flash.
      ii. For AR14xx: use mmwave_sdk_<ver>/packages/ti/utils/ccsdebug /ar14xx_ccsdebug_mss.bin as the MSS_BUILD filename for the one-time flash.
   b. Follow the steps in How to connect AR14xx/AR16xx 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 R4_0
      ii. Load the MSS program. (for example: ar16xx_<module>_mss.xer4f prebuilt executables provided in the SDK release package)
   d. If the test requires an application to run on DSP (AR16xx only)
      i. Connect CCS to C674X_0
      ii. Load the DSS program. (for example: ar16xx_<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. 4. Running the Demos

Assuming that you have loaded the right demo binary using the section above, set the EVM to functional mode and power up the device. Connect the EVM to the PC using its XDS110 micro-USB port/cable. When the USB is connected to the PC, the device manager should recognize the following COM ports as shown in the Figure above:

1. **XDS110 Class Auxiliary Data port** -> 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), call this **visualization port**.
2. **XDS110 Class Application/User UART** -> This is the port where **CLI (command line interface)** runs for all the various demos.

3. 4. 1. mmWave Demo for AR14xx/AR16xx

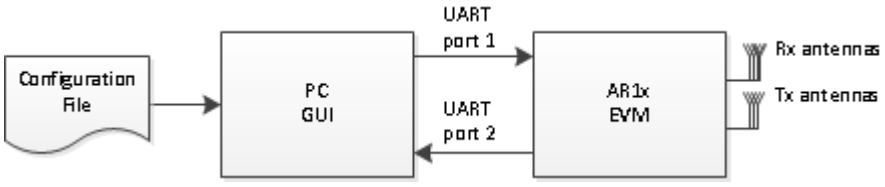


Figure 5: mmWave Demo PC-AR14xx Connectivity

1. Power on the EVM in functional mode, connect it to the PC as shown above with the USB cable and run the GUI executable located in the SDK release at mmwave_sdk_<ver>\packages\ti\demo\<platform>\mmw\gui\mmw_demo.exe on the Windows PC

with the following arguments. A few profile configuration files (*.cfg) that have been tested are provided in the same directory as the GUI executable. See CFG section below to understand the various commands and the parameters that are present in this cfg file.

Command line

```
# Use the correct COM ports from device manager; See section above on how
to match the Device manager ports to this command line
mmw_demo   8   3.5   2   7   profile_2d.cfg  1
```

Follow the instructions in mmwave_sdk_<ver>\packages\ti\demo\<platform>\mmw\gui\mmw_demo_preRequisites.txt when running this executable for the first time.

To know the definition of arguments to the executable, type only mmw_demo without any arguments. See one such output below for ar16xx reference:

sample help output for mmw_demo.exe on ar16xx

```
> mmw_demo
Starting UI for mmWave Demo ....
!!ERROR:Missing arguments!!
Specify arguments in this order: <comportSnum> <range_depth> <range_width>
<comportCliNum> <cliCfgFileName> <loadCfg>
comportSnum             : comport on which visualization data is being sent

range_depth (in meters) : determines the y-axis of the plot
range_width (in meters) : determines the x-axis (one sided) of the plot
comportCliNum           : comport over which the cli configuration is sent
to AR16xx
cliCfgFileName          : Input cli configuration file
loadCfg                 : loadCfg=1: cli configuration is sent to AR16xx
                          loadCfg=0: it is assumed that configuration is
already sent to AR16xx
                                     and it is already running, the
configuration is not sent to AR16xx,
                                     but it will keep receiving and
displaying incomming data.
```

The GUI exe performs the following steps:

a. Reads the configuration file to be able to make sense of the incoming data that it will display.
b. If the last argument is 1:
    i. Opens the CLI UART port.
    ii. Transmits the .cfg file to the CLI.
    iii. Closes the CLI port.
c. Receives the data generated by the demo on the visualization COM port and processes it to create various displays based on the GUI configuration in the cfg file.

The CLI port can also be operated outside the context of the Matlab GUI executable. The user can enter the commands from the cfg file manually on the COM port and after sensorStart is given, run the GUI executable with the last argument as 0 and using the same parameters provided on the CLI as input cfg file to the executable. In this case, the user may have to run the executable more than once if you see a message "Serial port sync loss! ...".

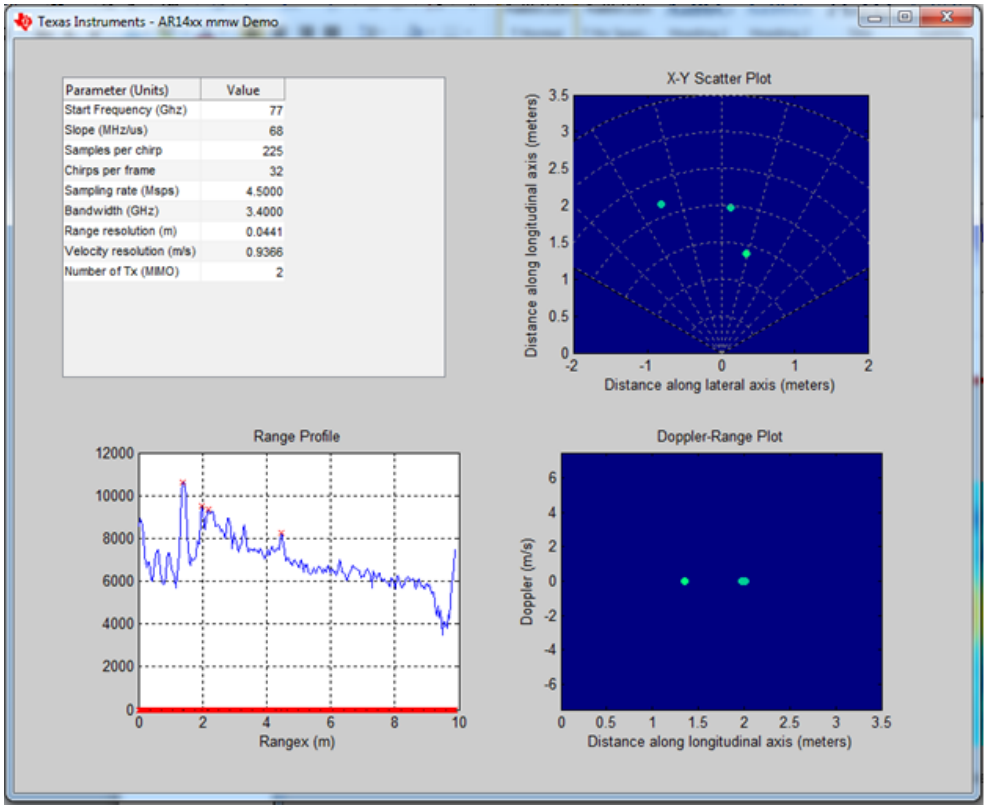
2. The format of the data streamed out of the demo is currently as follows (for latest, the user is advised to refer the doxygen documentation of the mmw demo):

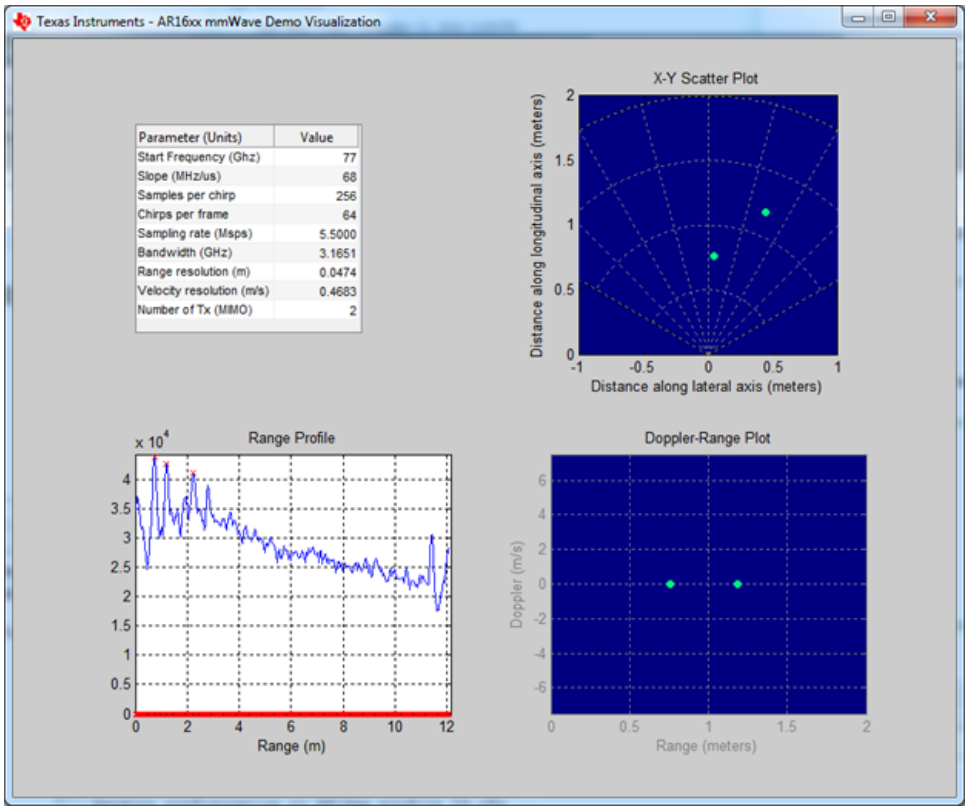
UART Data ouput from AR14xx/AR16xx mmWave Demo

```
[Note: the flags mentioned in 5, 6 and 7, 8 below are related to the GUI
configuration in the cfg file]
1. Magic word, size = (8 bytes) [0x0102, 0x0304, 0x0506, 0x0708]
2. RF4 CPU interframe processing cycles, size = sizeof(uint32_t),
3. Noise Energy, size = sizeof(uint32_t),
4. Number of objects detected, size = sizeof(uint8_t),
5. If detectedObjects flag is set, pbjOut structure containing range,
   doppler, and X,Y,Z location for detected objects,
   size = sizeof(objOut_t) * MMW_MAX_OBJ_OUT
6. If logMagRange flag is set,  rangeProfile,
   size = number of range bins * sizeof(uint16_t)
7. If rangeAzimuthHeatMap flag is set, the zero Doppler column of the
   range cubed matrix, size = number of Rx Azimuth virtual antennas *
   number of chirps per frame * sizeof(uint32_t)
8. If rangeDopplerHeatMap flag is set, the log magnitude range-Doppler
matrix,
   size = number of range bins * number of Doppler bins * sizeof(uint16_t)
```

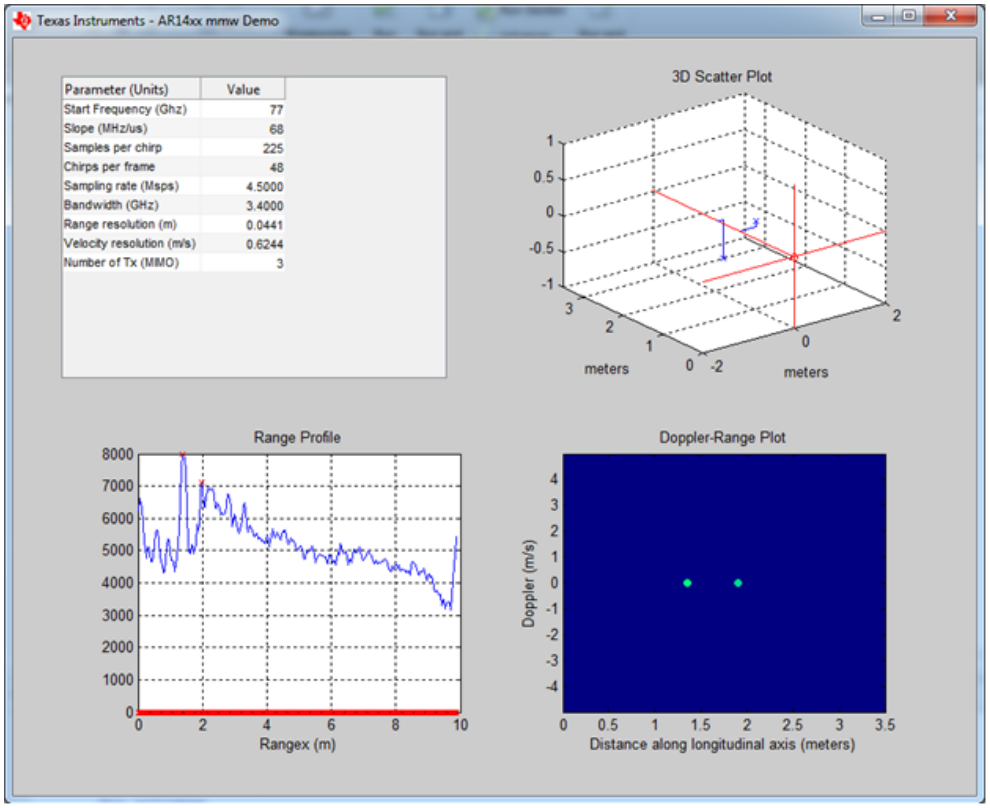
3. Here are some examples of GUI appearances that mmw_demo.exe produces based on the profile config that is passed to the application

Snapshot: Based on profile_2D.cfg

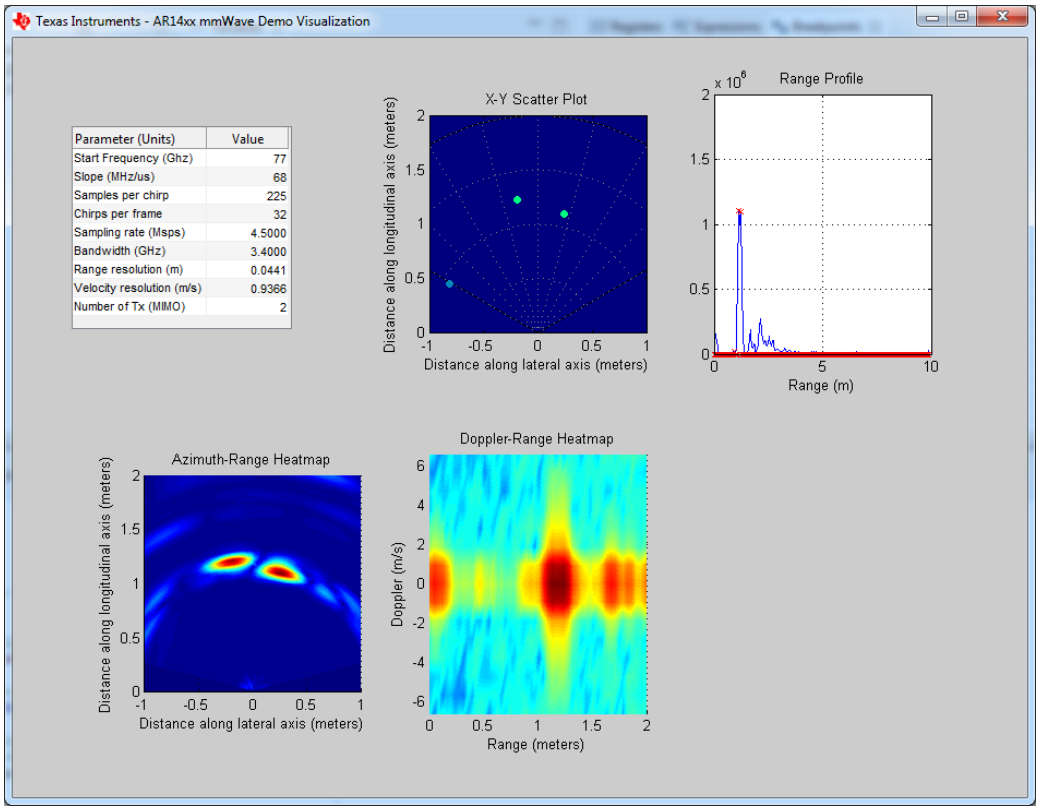


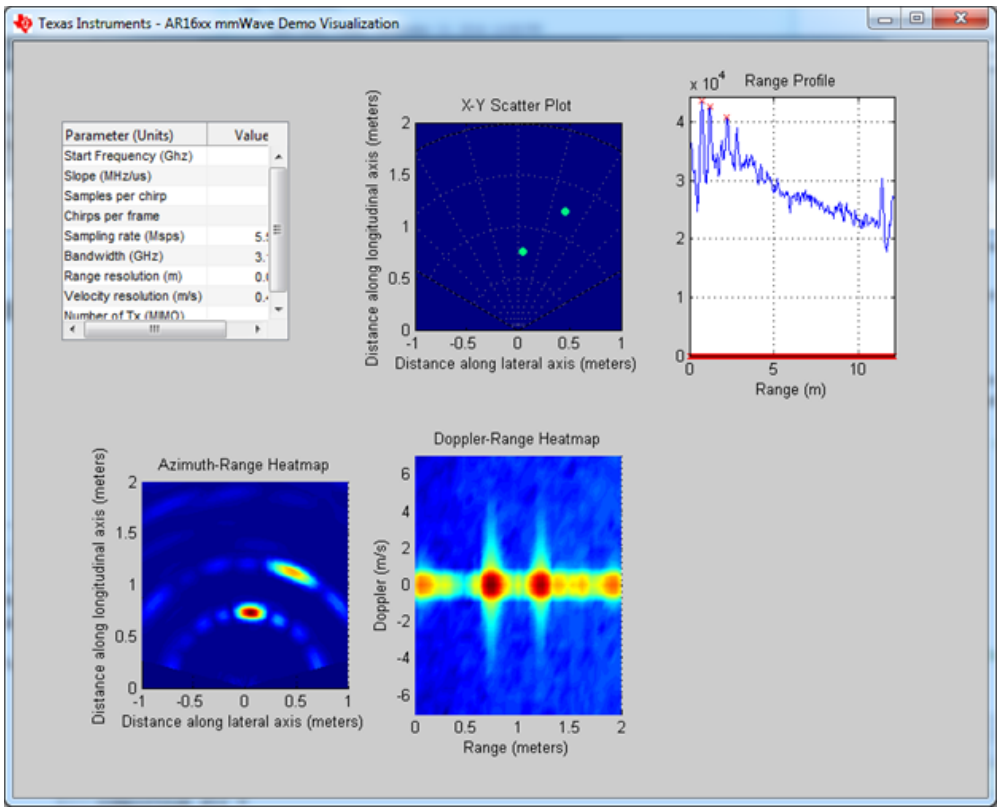


Snapshot: Based on profile_3D.cfg (AR14xx EVM only)



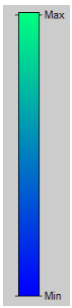
Snapshot: Based on profile_heat_map.cfg



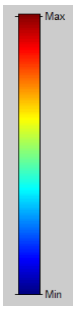


Color map legend:

The detected points plots uses "winter" color map



The heat map plots used "jet" color map



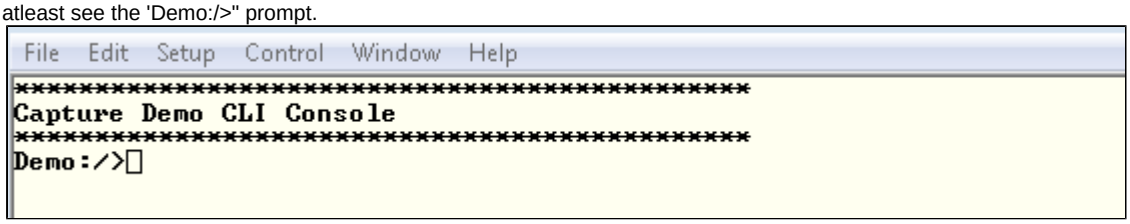
4. If board is rebooted, follow the steps starting from 1 above.

> Note that if you used the CLI COM port directly to send the commands (instead of mmw_demo.exe) you will have to close the CLI teraterm window and open a new one on every reboot. If you are using mmw_demo.exe to send the commands to the EVM, then just stop the application by setting the window focus on the graph plots and pressing the letter 'q'.

3. 4. 2. Capture Demo for AR14xx

> Note this demo requires connecting TI Code Composer studio to the AR14xx EVM

1. Power on the AR14xx EVM in functional mode, connect it to the PC with the USB cable. Open a teraterm or hyperterminal to connect to "User UART" COM port (see figure above).
2. The User UART COM port shows the demo cli as follows on power up. If you don't see this banner, hit 'enter' and you should atleast see the 'Demo:/>" prompt.

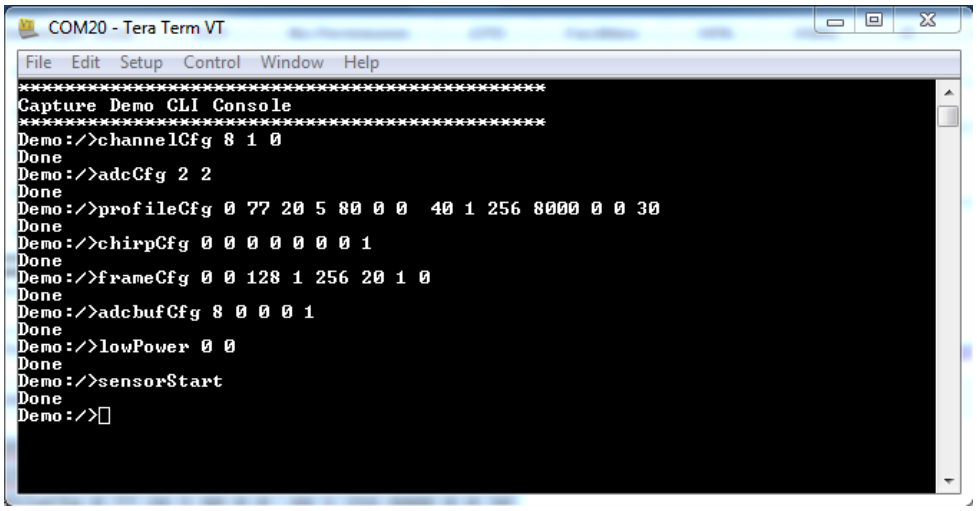


3. Send the commands provided in file mmwave_sdk_<ver>\packages\ti\demo\ar14xx\capture\ capture_demo_script.txt to the CLI window. Refer to CFG Section for details on parameters used in this demo.

Capture demo configuration for AR14xx

```
channelCfg 8 1 0
adcCfg 2 2
profileCfg 0 77 20 5 80 0 0  40 1 256 8000 0 0 30
chirpCfg 0 0 0 0 0 0 0 1
frameCfg 0 0 128 1 256 20 1 0
adcbufCfg 8 0 0 0 1
lowPower 0 0
sensorStart
```

Teraterm window would like as follows:



4. Without powering off the board, connect the board to CCS. (See section How to connect AR14xx/AR16xx EVM to CCS using JTAG for details on how to connect to CCS).
5. In CCS Expressions window, view the global variable gCaptureMCB and check DMA interrupt counter(dmaIntCounter) and chirp interrupt counter(chirpIntCounter) after sensorStart command. The chirpIntCounter should match the configuration. For the above configuration, the counter should be 0x80. The dmaIntCounter will be the actual number of DMA transfers triggered.

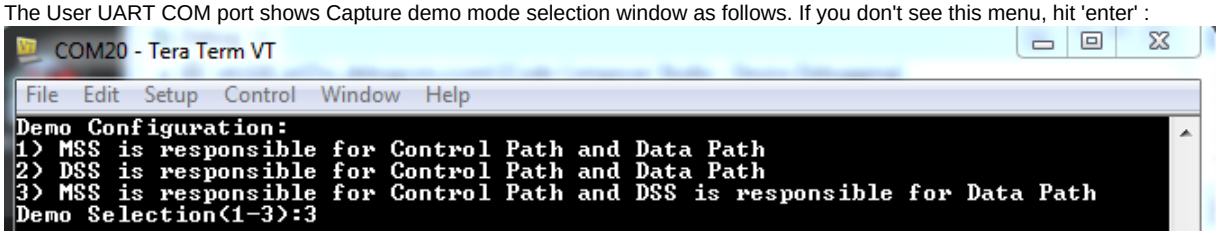
gCaptureDemoMCB	struct Capture_MCB	{...}
cfg	struct Capture_Cfg	{...}
bssMailbox	void *	0x08001A18
linkSemaphore	struct ti_sysbios_knl_Semaphore_Object *	0x080019A0
startSemaphore	struct ti_sysbios_knl_Semaphore_Object *	0x08001AC8
stopSemaphore	struct ti_sysbios_knl_Semaphore_Object *	0x00000000
loggingUartHandle	struct UART_Config *	0x0800B680
commandUartHandle	struct UART_Config *	0x0800B674
socHandle	void *	0x08000038
crcHandle	void *	0x08000C70
spawnFxn	void (*)(unknown*)	0x00000000
bssStatus	unsigned int	0x00000001
adcbufHandle	struct ADCBuf_Config_t *	0x0800B71C
dmaHandle	void *	0x08000CB0
chirpSema	struct ti_sysbios_knl_Semaphore_Object *	0x08000C90
chirpIntHandle	void *	0x08000074
arDeviceConfig	struct Capture_ARDeviceConfig	{...}
stats	struct Capture_Stats_t	{...}
chirpIntCounter	unsigned int	0x00000080
dmaIntCounter	unsigned int	0x00000080
spawnOverrun	unsigned int	0x00000000

6. Data generated can be saved from CCS and analyzed offline.
   a. Data is saved from L3 memory, base address(DEMO_L3RAM_DATA_MEM_ADDRESS) is defined in mmwave_sdk_<ver>\packages\ti\demo\ar14xx\capture\ capture.h.
   Select Tools -> save memory in file ccs_data.dat with format 16-bit Hex TI style, select memory location and size according to capture.h definition.
   b. Run mmwave_sdk_<ver>\packages\ti\demo\ar14xx\capture\ capture_demo.m in Matlab. The script is just an example for offline analyzing. The users are encouraged to re-use or create their own processing algorithms for this data.

3. 4. 3. Capture demo for AR16xx

Note this demo requires connecting TI Code Composer studio to the AR16xx EVM. If the demo enables streaming then the DevPack is required to be connected to AR16xx EVM to access the LVDS interface. See AR16xx EVM User guide for details.

1. Power on the AR16xx EVM in functional mode, connect it to the PC with the USB cable. Open a teraterm or Hyperterminal to connect to "User UART" COM port (see figure above).
2. The User UART COM port shows Capture demo mode selection window as follows. If you don't see this menu, hit 'enter' :



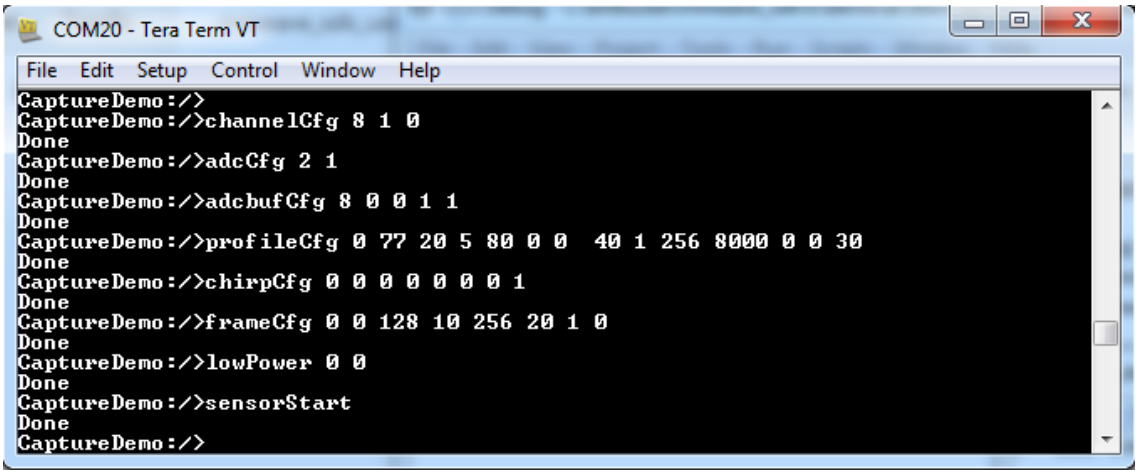
3. If mode 1 (MSS only) and 3 (MSS and DSS running in cooperative mode) are selected, the USER UART COM port shows demo configuration CLI as follows. For mode 2(DSS only mode), there is no configuration CLI. All configuration parameters are hard-coded in the code and are same as seen in the capture_demo_script.txt.

4. Next choose whether you want to run this demo in "**memory capture**" or "**streaming over LVDS**" mode and follow either step 5 or step 6 below. Note that only one of this mode can be selected per reboot. If streaming mode is selected, make sure a compatible device is connected to the other end of LVDS cable for capturing the stream of data.
5. **Memory capture procedure:**
   a. Send the commands provided in file(capture_demo_script.txt) to the CLI window. Refer to Configuration table for details on parameters used in this demo.

Capture demo configuration

```
channelCfg 8 1 0
adcCfg 2 1
adcbufCfg 8 0 0 1 1
profileCfg 0 77 20 5 80 0 0  40 1 256 8000 0 0 30
chirpCfg 0 0 0 0 0 0 0 1
frameCfg 0 0 128 1 256 20 1 0
lowPower 0 0
sensorStart
```

Teraterm window would like as follows:



   b. Without powering off the board, connect the board to CCS. (See section How to connect AR14xx/AR16xx EVM to CCS using JTAG for details on how to connect to CCS).
   c. In CCS Expressions window, view the global variable gCaptureMCB and check DMA interrupt counter(dmaIntCounter) and chirp interrupt counter(chirpIntCounter) after sensorStart command. The chirpIntCounter should match the configuration. For the above configuration, the counter should be 0x80. The dmaIntCounter will be the actual number of DMA transfers triggered.

> The stats(counters) should be checked on the Core that runs "data path".
>
> Mode 1: check on R4F
>
> Mode 2 and Mode 3: check on DSP

gCaptureMCB	struct Capture_MCB	{...}
cfg	struct Capture_Cfg	{...}
ctrlHandle	void *	0x0800A2C0
eventHandle	struct ti_sysbios_knl_Event_Object *	0x08000C70
peerMailbox	void *	0x00000000
mboxSemHandle	struct ti_sysbios_knl_Semaphore_Object *	0x00000000
loggingUartHandle	struct UART_Config *	0x0800C134
commandUartHandle	struct UART_Config *	0x0800C128
socHandle	void *	0x08000038
dataPathObj	struct Capture_DataPathObj_t	{...}
state	enum Capture_STATE_e	Capture_STATE_STARTED
stats	struct Capture_STATS_t	{...}
configEvt	unsigned char	0x00
startEvt	unsigned char	0x00
stopEvt	unsigned char	0x00
cliSensorStartEvt	unsigned char	0x01
cliSensorStopEvt	unsigned char	0x00
chirpIntCounter	unsigned int	0x00000080
frameStartIntCounter	unsigned int	0x00000001
dmaIntCounter	unsigned int	0x00000080
chirpEvt	unsigned int	0x00000000
frameStartEvt	unsigned int	0x00000000
frameTrigEvt	unsigned int	0x00000001

d. Data generated can be saved from CCS and analyzed offline.
   i. Data is saved from L3 memory, base address(gFrameAddress[0]) .
      Select Tools -> save memory in file ccs_data.dat with format 16-bit Hex TI style, select memory location and size according to test configuration.
      For the configuration from sample script, address is gFrameAddress[0], data size is 0x20000.
   ii. Run mmwave_sdk_<ver>\packages\ti\demo\ar16xx\capture\gui\capture_demo.m in Matlab. The script is just an example for offline analyzing.
      The users are encouraged to re-use or create their own processing algorithms for this data.

6. **Streaming over LVDS** :
   a. For mode 1 (MSS only) and 3 (MSS and DSS running in cooperative mode): by default the Capture demo does not stream data. In order for the demo to stream out the data via the LVDS High speed interface enter the additional CLI "`enableStream`" command. A configuration file capture_demo_script_lvds.txt is provided for reference.

> Please ensure that the "`enableStream`" command is entered before the **sensorStart** command.

Capture Demo CLI LVDS Enable Streaming

(capture_demo_script_lvds.txt)

```
channelCfg 8 1 0
adcCfg 2 1
adcbufCfg 8 0 0 1 1
lowPower 0 0
profileCfg 0 77 100 5 80 0 0  40 1 256 8000 0 0 30
chirpCfg 0 0 0 0 0 0 0 1
frameCfg 0 0 128 1024 256 100 1 0
enableStream
sensorStart
```

The LVDS configuration used in the test is as follows:- [**NOTE:** This can be modified by changing the parameters in the Demo `CBUFF_open` invocation]
   i. LVDS lanes is 2
   ii. 16bit Output format
   iii. DDR Clock Mode

   b. For mode 2(DSS only mode), as there is no configuration CLI: by default streaming is not enabled from the DSS; but users can modify the source file and set the global flag to enable streaming.

DSS only mode: Enable LVDS streaming

```
gCaptureMCB.cfg.enableHighSpeedInterface = 1U;
```

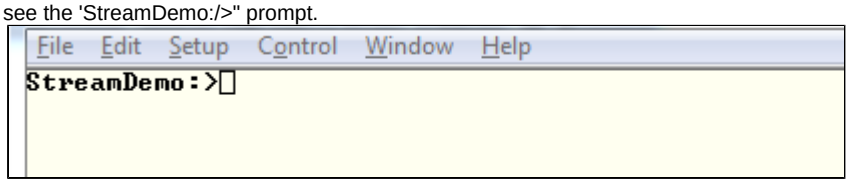
   c. If a compatible device is connected to the other end of the LVDS, data will be streamed out over LVDS lanes and can be collected on that compatible device.

d. You can follow the CCS procedure from the "Memory capture" mode to look at internal state/variables.

3. 4. 4. Streaming Demo for AR14xx

Note this demo requires connecting TI Code Composer studio to the AR14xx EVM. Also, DevPack is required to be connected to AR14xx EVM to access the CSI-2/LVDS interface. See AR14xx EVM User guide for details.

1. Connect the CSI-2/LVDS Interface of AR14xx EVM/DevPack to an external device that has a compatible CSI-2/LVDS. The details of the external device is out-of-scope for this document.
2. Toggle the SOP mode pins exposed on the DevPack (refer to DevPack Hardware documentation for more details) to set the functional mode for AR14xx EVM.
3. Power on the AR14xx EVM in functional mode (see step 2), connect it to the PC with the USB cable. Open a teraterm or hyperterminal to connect to "User UART" COM port (see figure above).
4. The User UART COM port shows the demo cli as follows on power up. If you don't see this banner, hit 'enter' and you should see the 'StreamDemo:/>" prompt.



5. Send the commands provided in file mmwave_sdk_<ver>\packages\ti\demo\ar14xx\stream\ lvds_stream _demo_script.txt or mmwave_sdk_<ver>\packages\ti\demo\ar14xx\stream\ csi_stream _demo_script.txt to the CLI window (command used for capture demo and stream demo are the same). Refer to CFG Section for details on parameters used in this demo.
   a. See sample commands below:

stream_demo_script.txt

```
channelCfg 8 1 0
adcCfg 2 2
adcbufCfg 8 0 0 0 1
lowPower 0 0
profileCfg 0 77 20 5 80 0 0  40 1 256 8000 0 0 30
chirpCfg 0 0 0 0 0 0 0 1
frameCfg 0 0 128 1024 256 20 1 0
setHSI   <LVDS|CSI>
sensorStart
```

   b. By default the demo does **"not stream"** data. Since the AR14xx supports both LVDS and CSI-2 the CLI command "`setHSI`" needs to be specified.

      Use the following command to select the LVDS as the High speed interface

Select LVDS AR14xx CLI command

```
setHSI   LVDS
```

      Use the following command to select CSI as the High speed interface

Select CSI AR14xx CLI command

```
setHSI   CSI
```

   c. After "sensorStart" command is issued, the mmWave ADC samples are streamed out over the LVDS/CSI-2 interface onto the external device.
6. Connect the board to CCS. (See section "How to connect AR14xx/AR16xx EVM to CCS using JTAG" for details on how to connect to CCS). The streaming demo maintains a global variable '`gStreamMCB`' which has all the configuration and run time information stored for the demo. The structure has a stats field which holds the Chirp Interrupt & DMA Interrupt counter. This should match the configuration which as per the above sample configuration should be set to 128. In addition to streaming it out, the demo copies all the ADC data to the L3 memory as well to enable matching the data generated on the AR14xx with the streamed data captured on the external device. See table below for location of each chirp buffer in AR14xx L3 memory.

Chirps	AR14xx L3 buffer location
1	0x51030000
2	0x51030400
3	0x51030800
4	0x51030C00
5	0x51031000
...	

On CSI: The demo has only been verified in Raw12 and Raw14 mode with the Data format set to CBUFF_DataFmt_ADC_DATA.

On LVDS: The demo has only been verified in the 16bit output format in DDR mode with the Data format set to CBUFF_DataFmt_ADC_DATA.

3. 5. 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) and shown with the profile_2d.cfg configuration as an example below. Note that some of the commands (ex: guiMonitor) are available for mmWave Demo only.

profile cfg file (profile_2d.cfg for AR16xx)

```
channelCfg 15 3 0
adcCfg 2 1
adcbufCfg 15 0 0 1 1
profileCfg 0 77 7 7 58 0 0 68 1 256 5500 0 0 30
chirpCfg 0 0 0 0 0 0 0 1
chirpCfg 1 1 0 0 0 0 0 2
frameCfg 0 1 32 0 256 100 1 0
lowPower 0 0
guiMonitor 1 1 0 0
cfarRangeCfg 0 8 4 4 0 1 5000
cfarDopplerCfg 0 8 4 4 0 1 5000
sensorStart
```

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.

Configuration	Parameters	Values	Comments	Values	Comments
		mmWave Demo: Profile for 2D detection		Capture/Stream Demo Profile	
Channel Cfg					
	Receive antenna mask e.g for 4 antennas, it is 0x1111b = 15	15	rx1, rx2, rx3, rx4	8	rx4
	Transmit antenna mask e.g for tx1,tx3 antennas out of the 2 physical, it is 0x11 = 3	3	tx1, tx2	1	tx1
	SoC cascading, not applicable for AR16XX, set to 0	0		0	
ADC Cfg					
	Number of ADC bits (0 for 12-bits, 1 for 14-bits and 2 for 16-bits)	2	16-bit	2	16-bit

	Output format : 0 - real 1 - complex 1x (image band filtered output) 2 - complex 2x (image band visible))	1	complex 1x - image band filtered	1	complex 1x - image band filtered
ADC buf Cfg					
	Receive channel bitmap e.g for 4 antennas, it is 0x1111b = 15	15	rx1,rx2,rx3,rx4	8	rx4
	Input sample format: 0 - complex 1 - real	0		0	
	IQ swap selection: 0 - I in LSB, Q in MSB 1 - Q in LSB, I in MSB	0		0	
	Input sample format with respect to multiple antennas: 0 - interleaved (not supported on AR16xx) 1 - non-interleaved	1	Must be 1 for AR16xx	1	Must be 1 for AR16xx
	Chirp threshold, number of chirps for ping/pong buffer to trigger ping/pong buffer switch.	1		1	
Profile Cfg					
	profile Identifier	0		0	
	start frequency in GHz	77	77 GHz	77	77 GHz
	idle time in u-sec	7	7 usec	20	20 usec
	ADC start time in u-sec	7	7 usec	5	5 usec
	Ramp end time in u-sec	58	58 usec	80	80 usec
	Tx output power back-off code for tx antennas (see mmwavelink doxgen for details)	0	zero back-off for all transmit antennas	0	zero back-off for all transmit antennas
	tx phase shifter for tx antennas (see mmwavelink doxgen for details)	0	no additional phase shift for all transmit antennas	0	no additional phase shift for all transmit antennas
	frequency slope constant (see mmwavelink doxgen for details)	68 MHz/u-sec	total bandwidth = 68 * 58 = 3.944 GHz	40	total bandwidth = 40 * 80 = 3.200 GHz
	tx start time in u-sec	1	1 u-sec	1	1 u-sec
	number of ADC samples	256	46.54 usec worth of samples (256/5500 ksps (next parameter)); sampled bandwidth = 68 * 46.54 = 3.16 GHz	256	46.54 usec worth of samples (256/5500 ksps (next parameter)); sampled bandwidth = 40 * 46.54 = 1.86 GHz
	ADC sampling frequency in ksps	5500	5500 ksps	8000	8000 ksps

	HPF1 (High Pass Filter 1) corner frequency 0: 175 KHz 1: 235 KHz 2: 350 KHz 3: 700 KHz	0	175 KHz	0	175 KHz
	HPF2 (High Pass Filter 2) corner frequency 0: 350 KHz 1: 700 KHz 2: 1.4 MHz 3: 2.8 MHz	0	350 KHz	0	350 KHz
	rx gain in dB (valid values 24 to 48)	30 dB		30 dB	
chirp Cfg #0					
	chirp start index	0		0	
	chirp end index	0		0	
	profile identifier	0	use profile 0	0	use profile 0
	start frequency variation in Hz	0	0 Hz	0	0 Hz
	frequency slope variation in Hz	0	0 KHz/u-sec	0	0 KHz/u-sec
	idle time variation in u-sec	0	0 u-sec	0	0 u-sec
	ADC start time variation in u-sec	0	0 u-sec	0	0 u-sec
	tx antenna enable mask (Tx2,Tx1) e.g (10)b = Tx2 enabled, Tx1 disabled	1	enable Tx1 only	1	enable Tx1 only
chirp Cfg #1				NA	
	chirp start index	1			
	chirp end index	1			
	profile identifier	0	use profile 0		
	start frequency variation	0			
	frequency slope variation	0			
	idle time variation	0			
	ADC start time variation	0			
	tx antenna enable mask	2	enable TX2 only		
FrameCfg					
	chirp start index (0-511)	0	Alternating chirp #0 and chirp #1	0	chirp #0
	chirp end index (chirp start index-511)	1	Alternating chirp #0 and chirp #1	0	
	number of loops (1 to 2^16-1)	32	32 times: 64 chirps in total	128	128 times (128 chirps)
	number of frames (valid range is 0 to 65535, 0 means infinite)	0	infinite	1	one frame
	Number of ADC samples to capture	256	256 samples	256	256 samples
	frame periodicity in ms	100	100 ms	20	20 ms

	trigger select (see mmwavelink doxgen for details) 1: Software trigger 2: Hardware trigger.	1	Software trigger	1	Software trigger
	Frame trigger delay in ms	0	0 ms	0	0 ms
guiMonitor				Not Applicable	
	All parameters below are flags (1 to enable and 0 to disable)				
	detected objects	1	Send detected objects		
	log magnitude range	1	Send log-magnitude of range		
	range-azimuth heat map related information (all virtual antenna data of 0th (doppler column) of the range-doppler matrix, the Matlab computes the FFT of this to show heat map)	0	Do not send range-azimuth heat map		
	range-doppler heat map	0	Do not send range-doppler heat map		
cfarRangeCfg (AR16xx) **cfarDopplerCfg (AR16xx)** **cfarCfg (AR14xx)**					
	CFAR averagining mode: 0 - CFAR_CA (Cell Averaging) 1 - CFAR_CAGO (Cell Averaging Greatest Of) 2 - CFAR_CASO (Cell Averaging Smallest Of)	0	CFAR-CA Note: on AR16xx, only CFAR-CA mode is supported. All modes are supported for AR14xx demo.		
	noise averaging window length	8	units: samples		
	guard length	4	units: samples		
	Cumulative noise sum divisor expressed as a shift. If this is x, noise sum is divided by 2^x	4	In this example, 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. The value to be used here should match the "CFAR averaging mode" and the "noise averaging window length" that is selected above. The actual value that is used for division (2^x) is a power of 2, even though the "noise averaging window length" samples may not have that restriction.		

	cyclic mode: 0 - disabled 1 - enabled	0	AR14xx: used for programming the CFAR engine inside HWA AR16xx: dont care in this release;The parameter cyclic mode is not used (the implementation assumes cyclic mode is enabled).		
	peak grouping: 0 - disabled 1 - enabled	1	Peak grouping enabled		
	Threshold scale. This is used in conjuntion with the noise sum divisor (say x). the CUT comparison for log input is: CUT > Threshold scale + (noise sum / 2^x) Note: log input is used for both AR14xx and AR16xx mmw demo	5000			
		Not Applicable			
setHSI **(Only on the AR14xx Stream Demo)**	High speed Interface name over which the data is streamed out			LVDS CSI	Stream out the data over the selected High Speed Interface i.e. LVDS or CSI (AR14xx only)
enableStream **(Only on the AR16xx Capture Demo)**	Enables the streaming of data over the LVDS High speed interface				Stream out the data over the LVDS High speed interface. (AR16xx only)
sensorStart	Starts the sensor. This function triggers the transmission of the frames as per the frame and chirp configuration.				
sensorStop	Presently not functional, must reboot board to give new chirp/frame configuration.				

Table 1: mmWave SDK Demos - CLI commands and parameters

3. 6. 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/ti_rtos. 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 flash an image onto AR14xx/AR16xx EVM

You will need the AR16xx EVM, USB cable and a Windows 7 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. To put the EVM in flashing mode, power off the board and place jumpers on pins marked as SOP2 and SOP0 .

SOP2	SOP1	SOP0	Bootloader mode & operation
0	0	1	**Functional Mode** Device Bootloader loads user application from QSPI Serial Flash to internal RAM and switches the control to it
1	0	1	**Flashing Mode** Device Bootloader spins in loop to allow flashing of user application to the serial flash.

2. **Procure the Images**
   For flashing AR1xxx devices, a command line utility is provided as part of the mmWave SDK package: mmwave_sdk_<ver>\tools\FlashProgrammer\arprog_cmdline.exe.
   a. **AR16xx:**
      For the SDK packaged AR16xx demos and ccsdebug utility, there is a bin file provided in their respective folder: ar16xx_<demo|ccsdebug>.bin which is the metaImage to be used for flashing. The metaImage already has the MSS, BSS and DSS application combined into one file. Users can use this for flashing their own metaImage as well.
      i. For demo mode, either mmwave_sdk_<ver>\ti\demo\ar16xx\mmw\ar16xx_mmw_demo.bin or mmwave_sdk_<ver>\ti\demo\ar16xx\capture\ar16xx_capture_demo.bin should be selected.
      ii. For CCS development mode, mmwave_sdk_<ver>\ti\ utils\ccsdebug\ar16xx_ccsdebug.bin should be selected.
   b. **AR14xx**:
      For correct operation of mmWave SDK demos, this utility needs 3 binary images:
      i. BSS **:** mmwave_sdk_<ver>\firmware\bss\ar12xx_bss.bin
      ii. MSS **:** For the SDK packaged AR14xx demos and ccsdebug utility, there is a bin file provided in their respective folder
      For demo mode, choose from **one** of these binary files from the SDK:
         1. mmwave_sdk_<ver>\packages\ti\demo\ar14xx\mmw\ar14xx_mmw_demo_mss.bin
         2. mmwave_sdk_<ver>\packages\ti\demo\ar14xx\capture\ar14xx_capture_demo_mss.bin
         3. mmwave_sdk_<ver>\packages\ti\demo\ar14xx\strea\ar14xx_stream_demo_mss.bin
      For CCS development mode, mmwave_sdk_<ver>\ti\ utils\ccsdebug\ar14xx_ccsdebug_mss.bin should be selected.
      iii. Config File **:** mmwave_sdk_<ver>\firmware\ar1xxx_config.bin

3. **Create the input file for the utility**

   You will need to create a simple text file with commands and filenames that the flashing utility understands and uses for programming the flash. A sample file for AR14xx and AR16xx is shown below (also included as part of the release package). **Modify the contents of this file** to reflect your image paths and save it in the same location as the flash programming utility (mmwave_sdk_<ver>\tools\FlashProgrammer\). Make sure each of these commands is on a separate line.
   a. **AR16xx**: In this context, it means there should be 2 lines in this text file.
      i. First line should say "FORMAT,SFLASH". This line results in erasing of the serial Flash on AR16xx EVM.
      ii. Second line should start with the path and filename of the metaimage binary (see step 2 above). After the path, following a comma separator, the keywords "METAIMAGE1,SFLASH" should be provided

sample_ar16xx_flash_mmw.txt

```
FORMAT,SFLASH
C:\ti\mmwave_sdk_<ver>\packages\ti\demo\ar16xx\mmw\ar16xx_mmw_demo.bin,META_IMAGE1,SFLASH
```

   b. **AR14xx**: In this context, it means there should be 4 lines in this text file.
      i. First line should say "FORMAT,SFLASH". This line results in erasing of the serial Flash on AR14xx EVM.
      ii. Second line should start with the path and filename of the BSS binary (See step 2 above). After the path, following a comma separator, the keywords "BSS_BUILD,SFLASH" should be provided
      iii. Third line should start with the path and filename of the MSS binary (See step 2 above). After the path, following a comma separator, the keywords "MSS_BUILD,SFLASH" should be provided
      iv. Fourth line should start with the path and filename of the config file (See step 2 above). After the path, following

a comma separator, the keywords "CONFIG_INFO,SFLASH" should be provided

sample_ar14xx_flash_mmw.txt

```
FORMAT,SFLASH
C:\ti\mmwave_sdk_<ver>\firmware\bss\ar12xx_bss.bin,BSS_BUILD,SFL
ASH
C:\ti\mmwave_sdk_<ver>\packages\ti\demo\ar14xx\mmw\ar14xx_mmw_de
mo_mss.bin,MSS_BUILD,SFLASH
C:\ti\mmwave_sdk_<ver>\firmware\ar1xxx_config.bin,CONFIG_INFO,SF
LASH
```

4. **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 COM8. Note the digit here: i.e. '8'. This number should be used for -p option to the flashing utility. Use the following command line with the options shown to start flashing the AR16xx EVM. Specify the filename that you created in step 3 for the option '-b'.

   Command line

   ```
   # Use the correct COM port from device manager and correct txt file name
   that you created in the step above
   arprog_cmdline.exe  -c  -p 8 -b sample_ar16xx_flash_mmw.txt
   ```

   Typical output of this command for AR16xx would look like this

   ```
   Connecting Com Port
   connect to device
   port opened
   set break signal
   --- please restart the device ---
   connection succeeded
   process batch
   file sample_ar16xx_flash_mmw.txt--
   Execute Line 1 - Format SFLASH
   erase storage [SFLASH]
   port opened
   erase storage ok
   file sample_ar16xx_flash_mmw.txt--
   Execute Line 2 - Download META_IMAGE1
   (C:\ti\mmwave_sdk_00_08_00_00\packages\ti\demo\ar16xx\mmw\ar16xx_mmw_demo.b
   in) Storage(SFLASH)
   downloading [META_IMAGE1] size [352580]
   port opened
   send start download command
   disconnecting device
   ```

5. **Switch back to Functional Mode**

   Refer to the EVM User Guide to understand the bootup modes of the EVM and the SOP jumpers. 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. 2. How to erase flash on AR14xx/AR16xx EVM

1. Setup the Booster Pack EVM for flashing as mentioned in step 1 of the section: How to flash an image onto AR14xx/AR16xx EVM
2. You will need to create a simple text file with just one line "FORMAT,SFLASH" . A sample sample_ar1xxx_flash_erase.txt is shown below (also included as part of the release package)

sample_ar1xxx_flash_erase.txt

```
FORMAT,SFLASH
```

3. Use the same flashing utility and com port as mentioned in the section: How to flash an image onto AR14xx/AR16xx EVM Use the above created text file as input to the flashing utility

Command line

```
# Use the correct COM port from device manager and correct txt file name
that you created
arprog_cmdline.exe  -c  -p 8 -b sample_ar1xxx_flash_erase.txt
```

Typical output of this command would look like this

```
Connecting Com Port
connect to device
port opened
set break signal
--- please restart the device ---
connection succeeded
process batch
file sample_ar1xxx_flash_erase.txt--
Execute Line 1 - Format SFLASH
erase storage [SFLASH]
port opened
erase storage ok
disconnecting device
```

4. 3. How to connect AR14xx/AR16xx 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. 3. 1. Emulation Pack Update

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

4. 3. 2. Target Configuration file for CCS (CCXML)

To create the ccxml file for connecting to the EVM, you will need to first install the device support package provided as part of the mmWave SDK release.

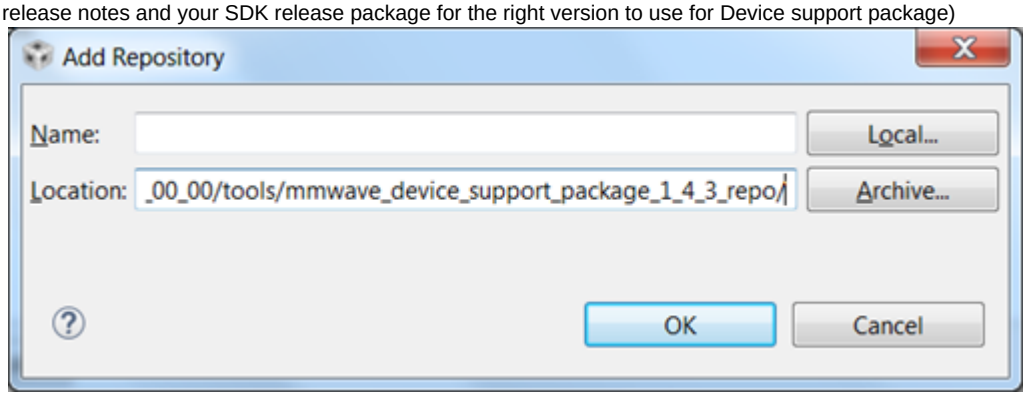
4. 3. 2. 1. Installing Device support package in CCS

Refer to your CCS installation help instructions on how to install a new software package and use that for installing the device support package. Following are sample instructions that are provided for your reference and assumes CCS v6.1.3 installation.

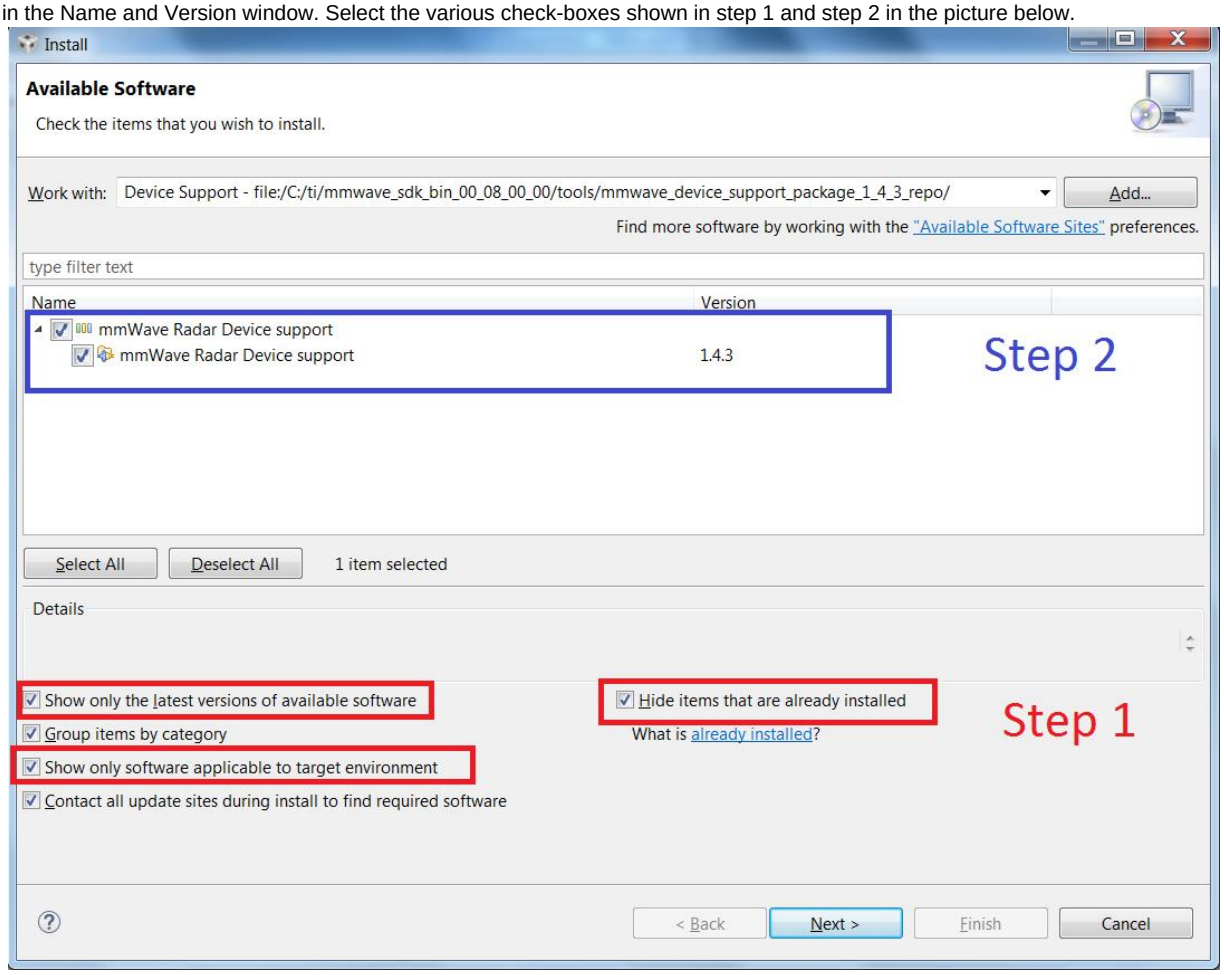
1. Unzip the zip file mmwave_sdk_<ver>\tools\mmwave_device_support_package_<ver>_repo.zip to some location, say C:\temp\mmwave_sdk_<ver>\tools\mmwave_device_support_package_<ver>\. This will have the directories binary, features

and some files underneath.

2. Start CCS and do Help->Install New Software
3. Select check-box "Show only software applicable to target environment" and press the Add button, this will pop up "Add Repository" window. In this window, press "Local" button, this will open "Browse for Folder" button. Navigate to the path where zip was extracted and say OK. See the snapshot below for reference (Note: 1_4_3 version is used for snapshot only. Refer to release notes and your SDK release package for the right version to use for Device support package)



4. The above step will fill the "Work with:" bar at the top with the path you have selected and show "mmWave Radar Device" entry in the Name and Version window. Select the various check-boxes shown in step 1 and step 2 in the picture below.

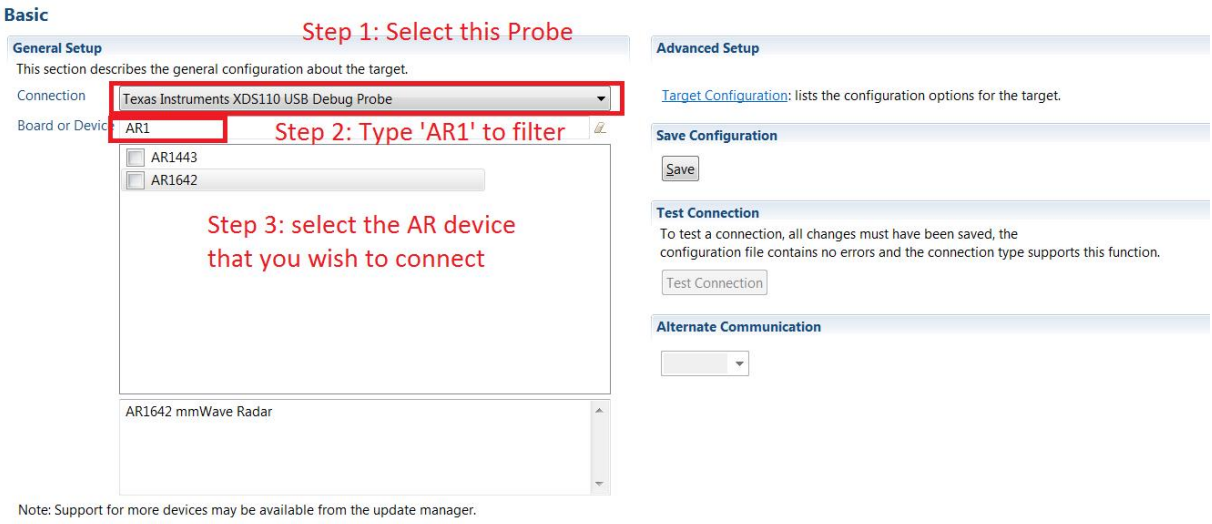


5. Press Next, this will start the installation and when done the screen will show "review the items to be installed".
6. Press Next and this will ask for acceptance of license agreement, select I accept after reading the agreement and press finish.
7. After installation is done, restart CCS.
8. Now you are ready to create a target configuration file for the mmWave device.

4. 3. 2. 2. Creating a CCXML file

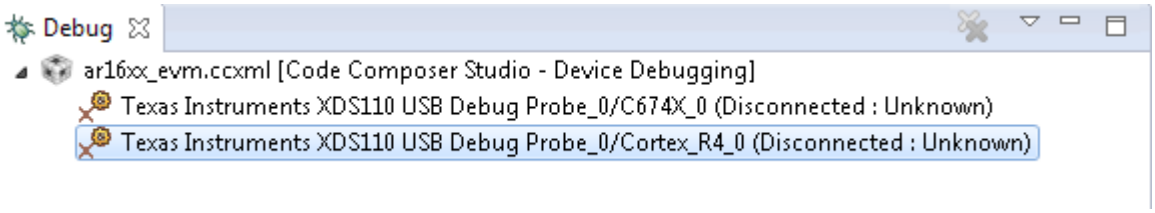
Assuming you have installed the device support package as mentioned in the section above, follow the steps mentioned below to create a target configuration file in CCS.

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 AR1642 or AR1443 and check the box. Click Save and the file will be created.



4. 3. 2. 3. Connecting to AR14xx/AR16xx 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. 1. Developing using SDK

4. 0. 1. Build Instructions

Follow the mmwave_sdk_release_notes instructions to install the mmwave_sdk in your development environment (windows or linux). Install all the tools versions mentioned in the mmwave_sdk_release_notes for the particular release. Most tools have their separate installer (windows or linux) and installation procedure should be straightforward. The exception is ARM compiler for which the installation needs to be done in CCS (Updating_CCSv6#Update_process)

4. 0. 2. Setting up build environment

4. 0. 2. 1. Windows

1. Create command prompt at mmwave_sdk_<ver>\packages\scripts\windows folder. Set the paths shown below as per your tools installation location. Place these commands in a batch file setenv_tools.bat and run **setenv_tools.bat**.

setenv_tools.bat

```
@REM Common settings for AR14xx and AR16xx
@REM TI ARM compiler
set
R4F_CODEGEN_INSTALL_PATH=C:/ti/ccsv6/tools/compiler/ti-cgt-arm_15.12.4.LTS
@REM Path to <mmwave_sdk installation path>/packages folder
set MMWAVE_SDK_INSTALL_PATH=C:/ti/mmwave_sdk_00_08_00_00/packages
@REM TI XDC
set XDC_INSTALL_PATH=c:/ti/xdctools_3_32_01_22_core
@REM TI BIOS
set BIOS_INSTALL_PATH=C:/ti/bios_6_46_04_53/packages
@REM perl
set PERL_INSTALL_PATH=C:/Strawberry/perl/bin
@REM if using CCS to download, set below define to yes else no
set DOWNLOAD_FROM_CCS=yes
@REM install from web (free s/w). skip if doxygen output is not needed
@REM set DOXYGEN_INSTALL_PATH=C:/ti/doxygen

@REM Following only needed for AR14xx
set MMWAVE_SDK_DEVICE=ar14xx

@REM Following only needed for AR16xx
@REM DEVICE
set MMWAVE_SDK_DEVICE=ar16xx
@REM TI DSP compiler
set C674_CODEGEN_INSTALL_PATH=C:/ti/ccsv6/tools/compiler/c6000_7.4.16
@REM DSPlib
set C64Px_DSPLIB_INSTALL_PATH=C:/ti/dsplib_c64Px_3_4_0_0
@REM MATHlib
set C674x_MATHLIB_INSTALL_PATH=C:/ti/mathlib_c674x_3_1_2_1
@REM AR16xx BSS firmware. Use the RPRC formatted binary file.
set AR16XX_BSS_IMAGE_BIN=<mmwave_sdk_<ver>
path>/firmware/bss/ar1xxx_bss_rprc.bin
```

Even though the build is in Windows environment the paths have to use forward slashes "/" in the paths shown above

Path setting for TI tools is done by mmwave_sdk_setupenv.bat below

2. Obtain CRC.pm (http://cpansearch.perl.org/src/OLIMAUL/Digest-CRC-0.21/lib/Digest/CRC.pm) and copy it to your Perl installation's lib\Digest path. ex: C:\Strawberry\perl\lib\Digest
3. Run **mmwave_sdk_setupenv.bat.** This should **not give errors** and should give the following output. The build environment is now setup

```
-----------------------------------------------
mmWave Build Environment Configured
-----------------------------------------------
```

Please note that the versions shown above are examples. The actual versions of the mmwave sdk and tools to be used are given in the Release notes of a particular release. Paths have to be updated with the correct location of the tools installed on the user's machine

4. 0. 2. 2. Linux

1. Open a terminal and cd to mmwave_sdk_<ver>/packages/scripts/unix. Set the paths shown below as per your tools installation location. Place these commands in a shell script setenv_tools.sh, enable execute permission and source it.

setenv_tools.sh

```
# Common settings for AR14xx and AR16xx
# TI ARM compiler
export R4F_CODEGEN_INSTALL_PATH=/opt/ti/ti-cgt-arm_15.12.4.LTS
# Path to <mmwave_sdk installation path>/packages folder
export MMWAVE_SDK_INSTALL_PATH=~/ti/mmwave_sdk_00_08_00_00/packages
# TI XDC
export XDC_INSTALL_PATH=/opt/ti/xdctools_3_32_01_22_core
# TI BIOS
export BIOS_INSTALL_PATH=/opt/ti/bios_6_46_04_53/packages
# perl
export PERL_INSTALL_PATH=/usr/bin
# if using CCS to download, set below define to yes else no
export DOWNLOAD_FROM_CCS=yes

# Following only needed for AR14xx
# Device
export MMWAVE_SDK_DEVICE=ar14xx

# Following only needed for AR16xx
# Device
export MMWAVE_SDK_DEVICE=ar16xx
# TI DSP compiler
export C674_CODEGEN_INSTALL_PATH=/opt/ti/c6000_7.4.16
# DSPlib
export C64Px_DSPLIB_INSTALL_PATH=/opt/ti/dsplib_c64Px_3_4_0_0
# MATHlib
export C674x_MATHLIB_INSTALL_PATH=/opt/ti/mathlib_c674x_3_1_2_1
# AR16xx BSS firmware. Use the RPRC formatted binary file.
export AR16XX_BSS_IMAGE_BIN=<mmwave_sdk_<ver>
path>/firmware/bss/ar1xxx_bss_rprc.bin
```

Run setenv_tools.sh

```
chmod +x setenv_tools.sh
source ./setenv_tools.sh
```

Path setting for TI tools is done by mmwave_sdk_setupenv.sh below

2. Install CRC for the perl installation:

Install other needed tools

```
sudo apt-get --assume-yes install libdigest-crc-perl

# Install Mono. This is needed to run windows executable to convert .out to
.bin (flashable application executable)
# The following command was tested for Ubuntu. For other Linux
distributions refer to http://www.mono-project.com/download/#download-lin
sudo apt-get --assume-yes install mono-complete
```

3. 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
```

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

Run mmwave_sdk_setupenv.sh

```
source ./mmwave_sdk_setupenv.sh
```

Please note that the versions shown above are examples. The actual versions of the mmwave sdk and tools to be used are given in the Release notes of a particular release. Paths have to be updated with the correct location of the tools installed on the user's machine

4. 0. 3. 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 ar14xx, the commands will create
REM     libadcbuf_ar14xx.aer4f library under
%MMWAVE_SDK_INSTALL_PATH%/ti/drivers/adcbuf/lib folder
REM     ar14xx_adcbuf_mss.xer4f unit test binary under
%MMWAVE_SDK_INSTALL_PATH%/ti/drivers/adcbuf/test/ar14xx folder
REM If MMWAVE_SDK_DEVICE is set to ar16xx, the commands will create
REM     libadcbuf_ar16xx.aer4f library for MSS under
%MMWAVE_SDK_INSTALL_PATH%/ti/drivers/adcbuf/lib folder
REM     ar16xx_adcbuf_mss.xer4f unit test binary for MSS under
%MMWAVE_SDK_INSTALL_PATH%/ti/drivers/adcbuf/test/ar16xx folder
REM     libadcbuf_ar16xx.ae674 library for DSS under
%MMWAVE_SDK_INSTALL_PATH%/ti/drivers/adcbuf/lib folder
REM     ar16xx_adcbuf_dss.xe674 unit test binary for DSS under
%MMWAVE_SDK_INSTALL_PATH%/ti/drivers/adcbuf/test/ar16xx folder

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 ar14xx, the commands will create
REM     libmmwavelink_ar14xx.aer4f library under
%MMWAVE_SDK_INSTALL_PATH%/ti/control/mmwavelink/lib folder
REM     ar14xx_link_mss.xer4f unit test binary under
%MMWAVE_SDK_INSTALL_PATH%/ti/drivers/control/mmwavelink/test/ar14xx folder
REM If MMWAVE_SDK_DEVICE is set to ar16xx, the commands will create
REM     libmmwavelink_ar16xx.aer4f library for MSS under
%MMWAVE_SDK_INSTALL_PATH%/ti/control/mmwavelink/lib folder
REM     ar16xx_link_mss.xer4f unit test binary for MSS under
%MMWAVE_SDK_INSTALL_PATH%/ti/control/mmwavelink/test/ar16xx folder
REM     libmmwavelink_ar16xx.ae674 library for DSS under
%MMWAVE_SDK_INSTALL_PATH%/ti/control/mmwavelink/lib folder
REM     ar16xx_link_dss.xe674 unit test binary for DSS under
%MMWAVE_SDK_INSTALL_PATH%/ti/control/mmwavelink/test/ar16xx folder

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 ar14xx, the commands will create
#     libadcbuf_ar14xx.aer4f library under
${MMWAVE_SDK_INSTALL_PATH}/ti/drivers/adcbuf/lib folder
#     ar14xx_adcbuf_mss.xer4f unit test binary under
${MMWAVE_SDK_INSTALL_PATH}/ti/drivers/adcbuf/test/ar14xx folder
# If MMWAVE_SDK_DEVICE is set to ar16xx, the commands will create
#     libadcbuf_ar16xx.aer4f library for MSS under
${MMWAVE_SDK_INSTALL_PATH}/ti/drivers/adcbuf/lib folder
#     ar16xx_adcbuf_mss.xer4f unit test binary for MSS under
${MMWAVE_SDK_INSTALL_PATH}/ti/drivers/adcbuf/test/ar16xx folder
#     libadcbuf_ar16xx.ae674 library for DSS under
${MMWAVE_SDK_INSTALL_PATH}/ti/drivers/adcbuf/lib folder
#     ar16xx_adcbuf_dss.xe674 unit test binary for DSS under
${MMWAVE_SDK_INSTALL_PATH}/ti/drivers/adcbuf/test/ar16xx folder


# 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 ar14xx, the commands will create
#     libmmwavelink_ar14xx.aer4f library under
${MMWAVE_SDK_INSTALL_PATH}/ti/control/mmwavelink/lib folder
#     ar14xx_link_mss.xer4f unit test binary under
${MMWAVE_SDK_INSTALL_PATH}/ti/drivers/control/mmwavelink/test/ar14xx folder
# If MMWAVE_SDK_DEVICE is set to ar16xx, the commands will create
#     libmmwavelink_ar16xx.aer4f library for MSS under
${MMWAVE_SDK_INSTALL_PATH}/ti/control/mmwavelink/lib folder
#     ar16xx_link_mss.xer4f unit test binary for MSS under
${MMWAVE_SDK_INSTALL_PATH}/ti/control/mmwavelink/test/ar16xx folder
#     libmmwavelink_ar16xx.ae674 library for DSS under
${MMWAVE_SDK_INSTALL_PATH}/ti/control/mmwavelink/lib folder
#     ar16xx_link_dss.xe674 unit test binary for DSS under
${MMWAVE_SDK_INSTALL_PATH}/ti/control/mmwavelink/test/ar16xx folder

# 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 Core Library only
drvClean             -> Clean the Core Library only
test                 -> Builds all the Unit Test
testClean            -> Cleans all the Unit Tests
**********************************************************************
```

example output of make help for mmwave control and alg component

```
**********************************************************************
* Makefile Targets for the mmWave Control
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. 0. 4. Building demo

To clean build a demo, first make sure that the environment is setup as detailed in earlier section. Then run the following command. On successful execution of the command, 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 AR14xx/AR16xx EVM".

Building demo in windows

```
REM Use ar14xx or ar16xx for <device> below. Use mmw or capture for <demo> below
cd %MMWAVE_SDK_INSTALL_PATH%/ti/demo/<device>/<demo>

REM Clean and build
gmake clean
gmake all

REM Incremental build
gmake all

REM For example to build the mmw demo for ar14xx
cd %MMWAVE_SDK_INSTALL_PATH%/ti/demo/ar14xx/mmw
gmake clean
gmake all
REM This will create ar14xx_mmw_demo_mss.xer4f & ar14xx_mmw_demo_mss.bin binaries
under %MMWAVE_SDK_INSTALL_PATH%/ti/demo/ar14xx/mmw folder


REM For example to build the mmw demo for ar16xx
cd %MMWAVE_SDK_INSTALL_PATH%/ti/demo/ar16xx/mmw
gmake clean
gmake all
REM This will create ar16xx_mmw_demo_mss.xer4f, ar16xx_mmw_demo_dss.xe674 &
ar14xx_mmw_demo.bin binaries under %MMWAVE_SDK_INSTALL_PATH%/ti/demo/ar16xx/mmw
folder
```

Building demo in linux

```
# Use ar14xx or ar16xx for <device> below. Use mmw or capture for <demo> below
cd ${MMWAVE_SDK_INSTALL_PATH}/ti/demo/<device>/<demo>

# Clean and build
make clean
make all

# Incremental build
make all

# For example to build the mmw demo for ar14xx
cd ${MMWAVE_SDK_INSTALL_PATH}/ti/demo/ar14xx/mmw
make clean
make all
# This will create ar14xx_mmw_demo_mss.xer4f & ar14xx_mmw_demo_mss.bin binaries
under ${MMWAVE_SDK_INSTALL_PATH}/ti/demo/ar14xx/mmw folder

# For example to build the mmw demo for ar16xx
cd ${MMWAVE_SDK_INSTALL_PATH}/ti/demo/ar16xx/mmw
gmake clean
gmake all
# This will create ar16xx_mmw_demo_mss.xer4f, ar16xx_mmw_demo_dss.xe674 &
ar14xx_mmw_demo.bin binaries under ${MMWAVE_SDK_INSTALL_PATH}/ti/demo/ar16xx/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.

5. MMWAVE SDK deep dive

5. 1. Typical mmWave Radar Processing Chain

Following figure shows a typical mmWave Radar processing chain:

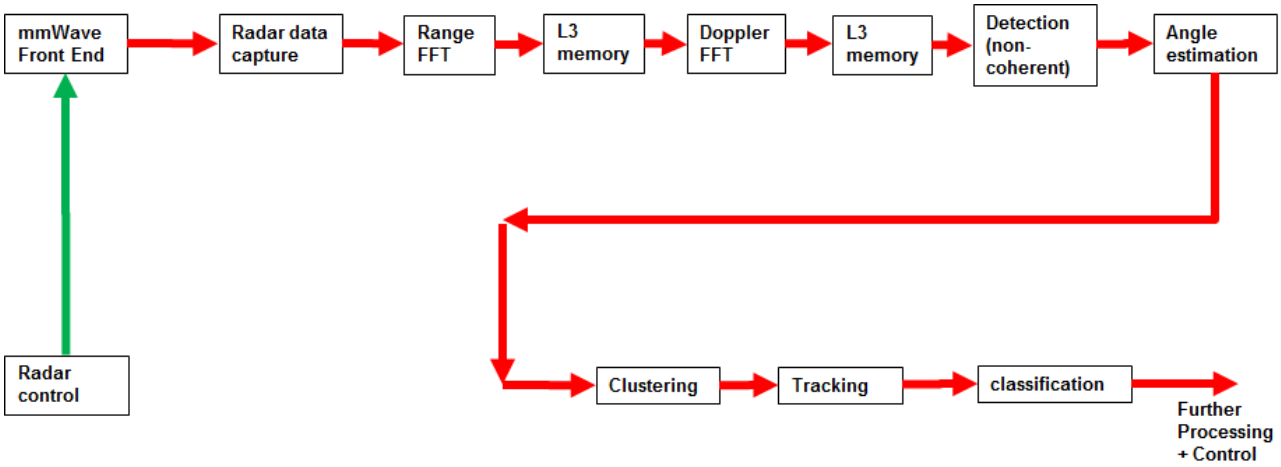


Figure 6: Typical mmWave radar processing chain

Using mmWave SDK the above chain could be realized as shown in the following figure for AR14xx and AR16xx. 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.

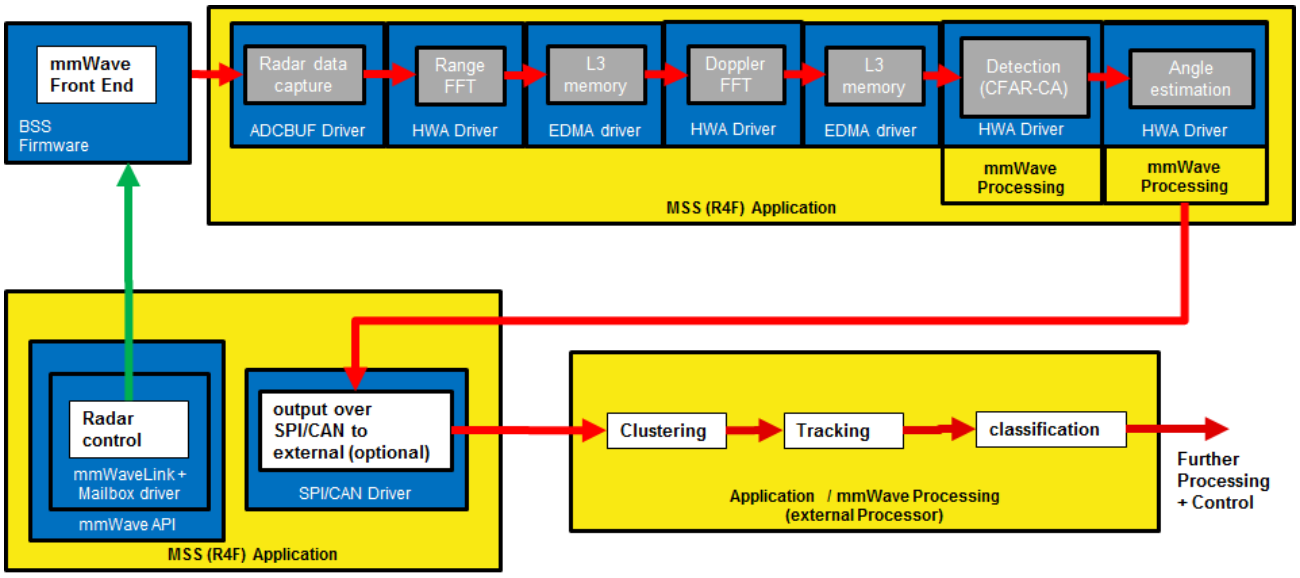


Figure 7: Typical mmWave radar processing chain using AR14xx mmWave SDK

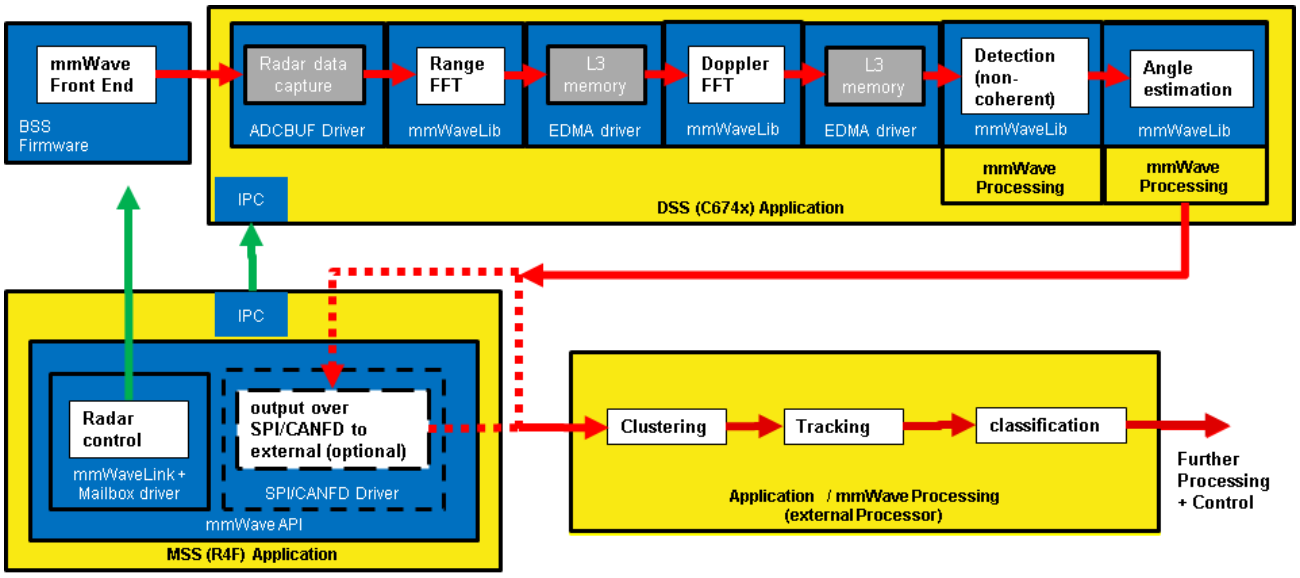


Figure 8: Typical mmWave radar processing chain using AR16xx mmWave SDK

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. 2. Typical Programming Sequence

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

5. 2. 1. Control Path

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

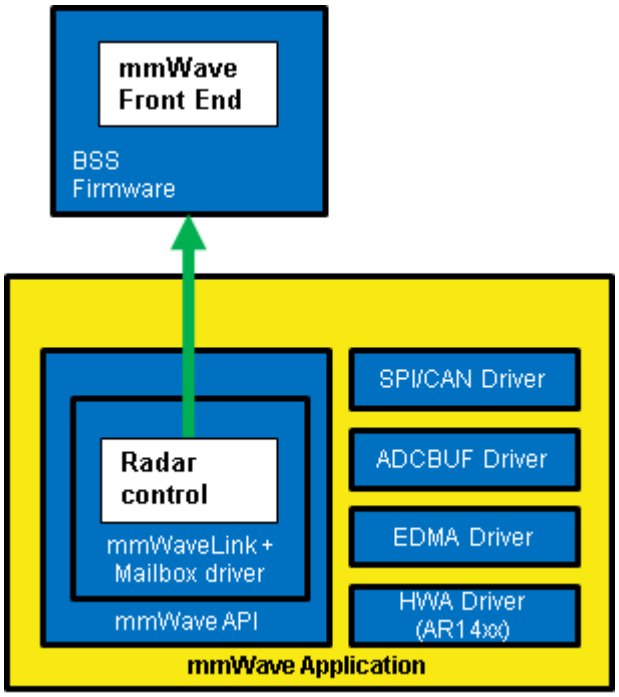


Figure 9: Typical mmWave radar control flow

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>\packag

es\ti\demo\<platform>\mmw folder.

5. 2. 1. 1. AR14xx (MSS<->BSS)

On AR14xx, 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.

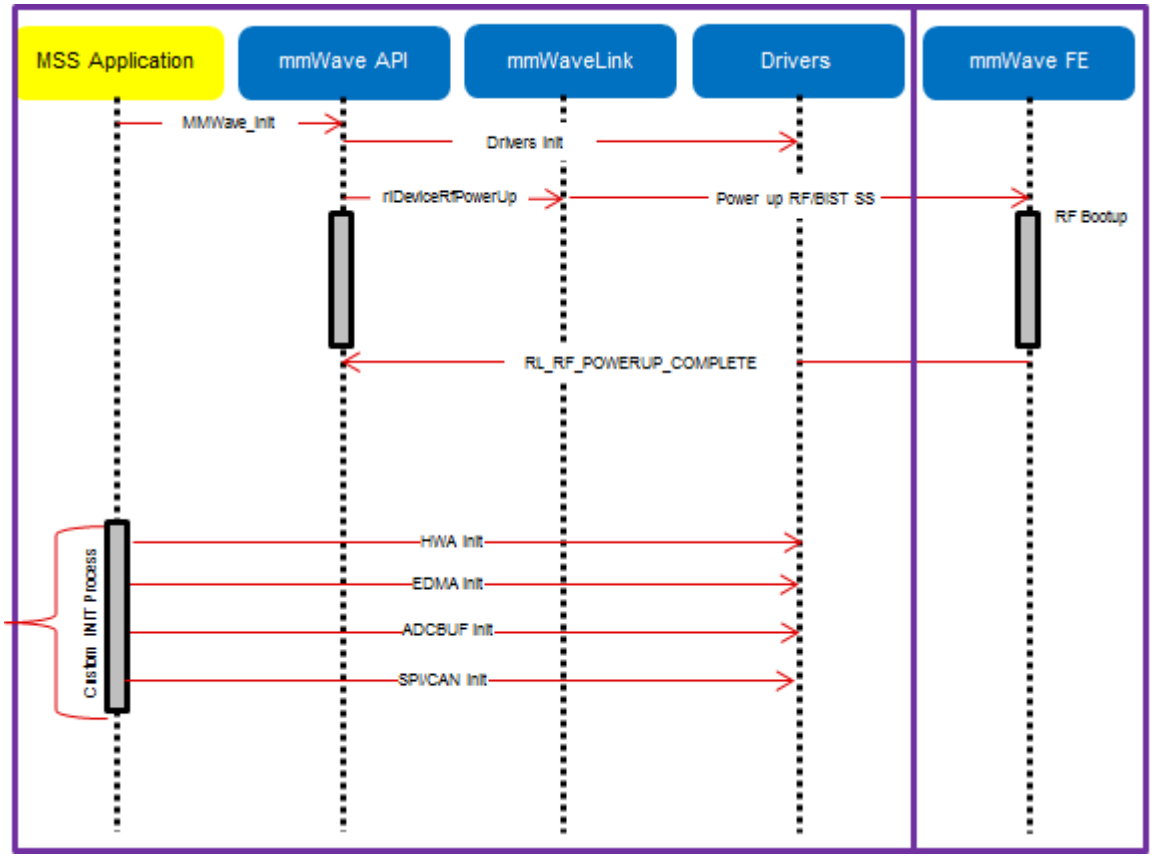


Figure 10: AR14xx: Detailed Control Flow (Init sequence)

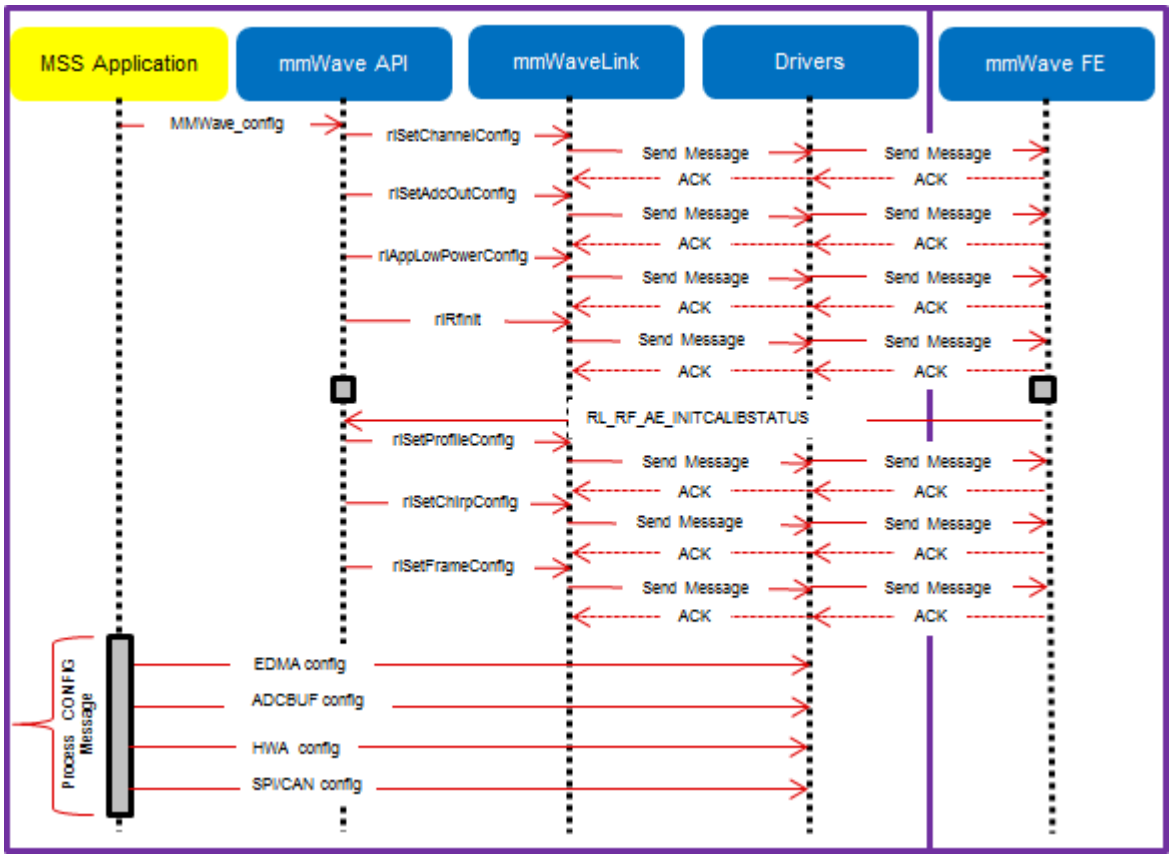


Figure 11: AR14xx: Detailed Control Flow (Config sequence)

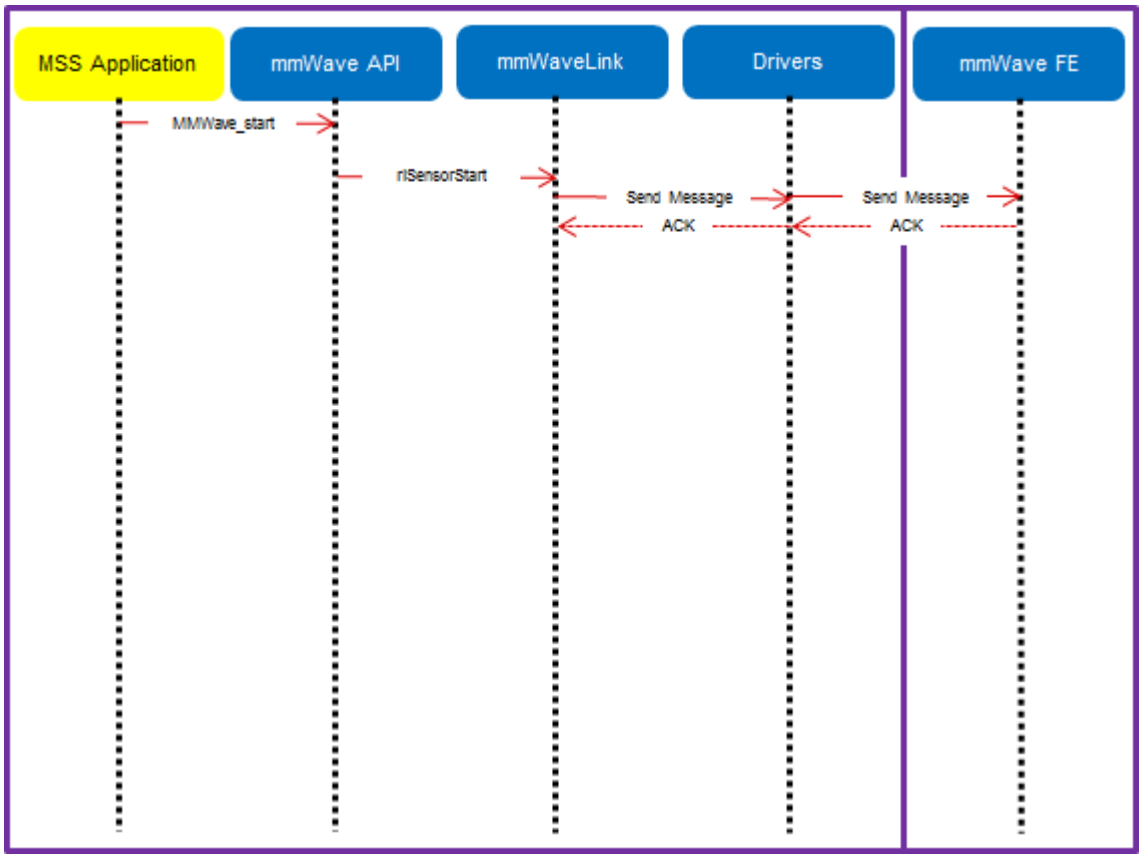


Figure 12: AR14xx: Detailed Control Flow (start sequence)

5. 2. 1. 2. AR16xx

On AR16xx, 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 BSS (RF) and DSS (data processing). The co-operative mode can be seen in the MMW demo. The "capture" demo provides a sample implementation of all 3 modes.

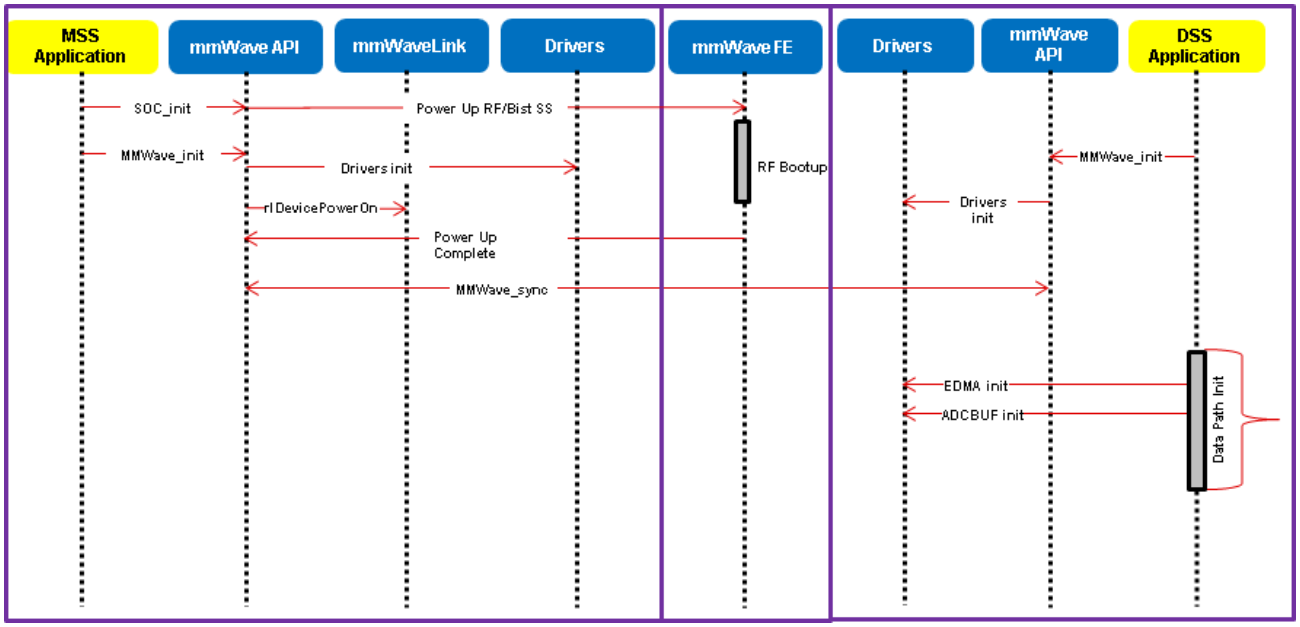


Figure 13: ***AR16xx: Detailed Control Flow (Init sequence)***

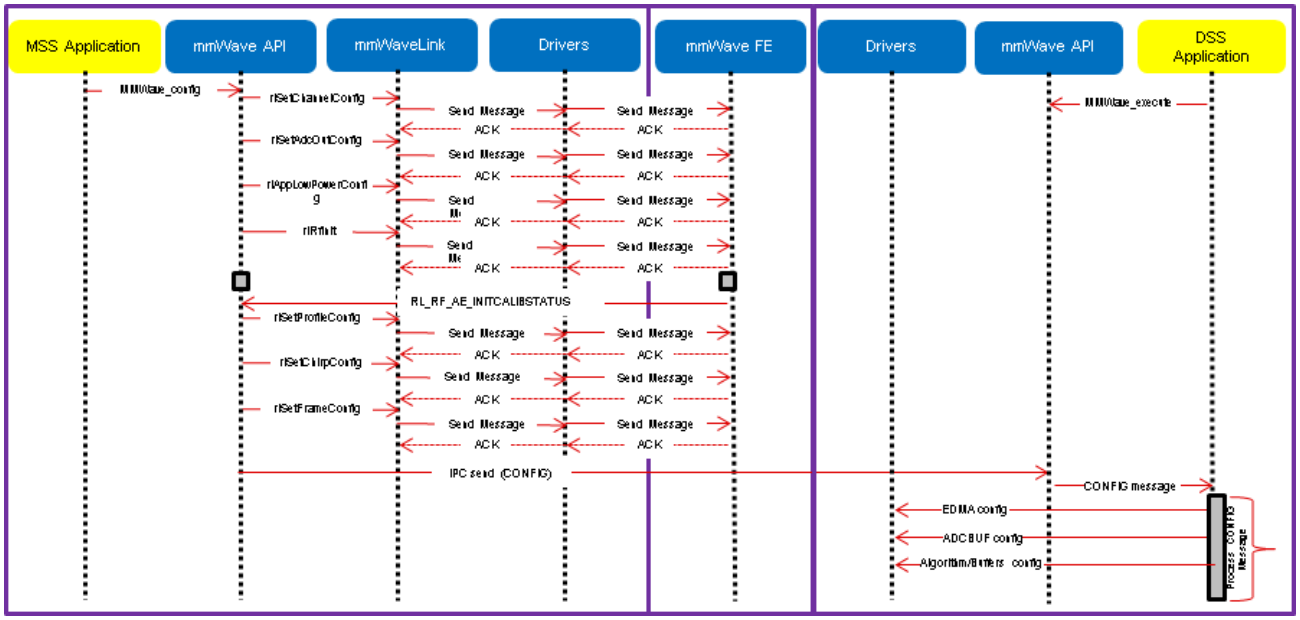


Figure 14: ***AR16xx: Detailed Control Flow (Config sequence)***

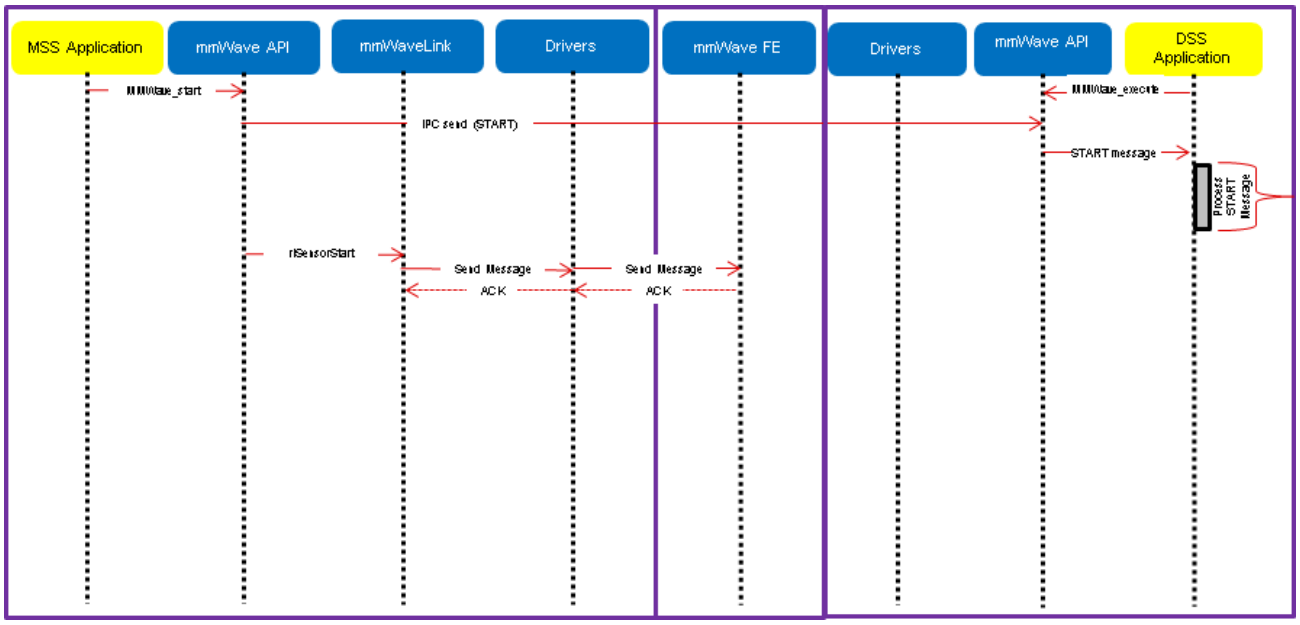


Figure 15: ***AR16xx: Detailed Control Flow (Start sequence)***

5. 2. 2. Data Path

5. 2. 2. 1. AR14xx

In AR14xx, the data path in the above processing chain is depicted by the following blocks running on the MSS (Cortex-R4F).

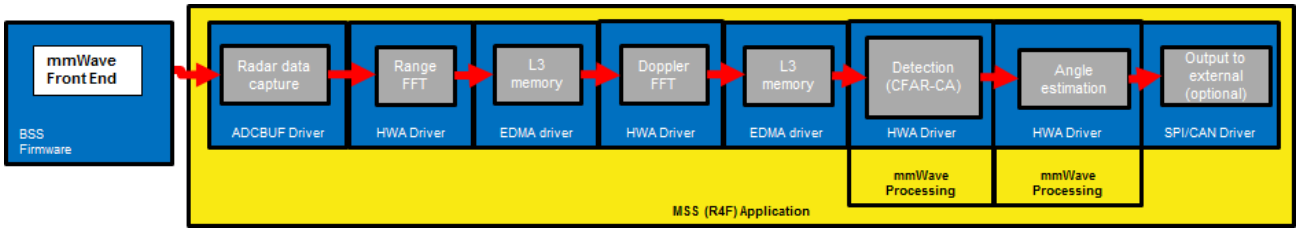


Figure 16: Typical mmWave radar data flow in AR14xx

Please refer to the documentation provided here mmwave_sdk_<ver>\packages\ti\demo\ar14xx\mmw\docs\doxygen\html\index.html for more details on each of the individual blocks of the data path.

5. 2. 2. 2. AR16xx

In AR16xx, the data path in the above processing chain is depicted by the following blocks running on primarlity on the DSS (C674x).

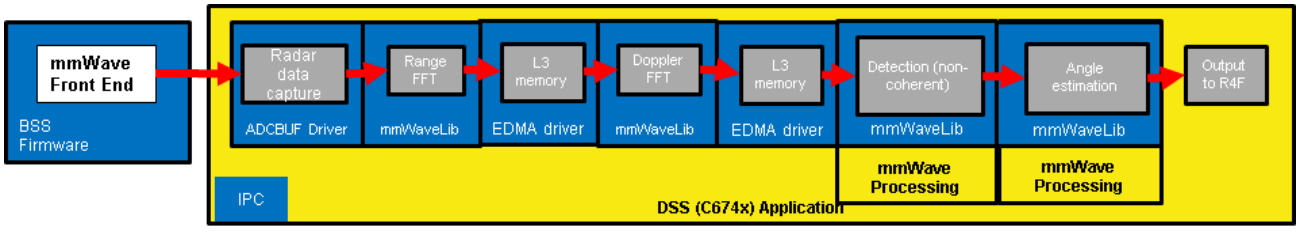


Figure 17: Typical mmWave radar data flow in AR16xx

Please refer to the documentation provided here mmwave_sdk_<ver>\packages\ti\demo\ar16xx\mmw\docs\doxygen\html\index.html for more details on each of the individual blocks of the data path.

5. 3. mmWave SDK - TI components

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

5. 3. 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.

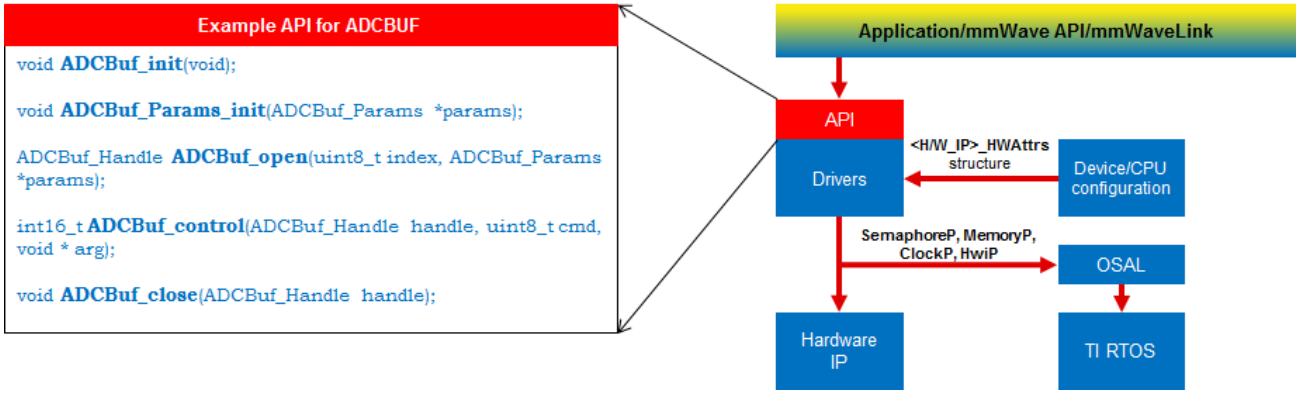


Figure 18: mmWave SDK Drivers - Internal software design

5. 3. 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 AR16xx 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. 3. 3. mmWaveLink

mmWaveLink is a control layer and primarily implements the protocol that is used to communicate between the BIST subsystem (BSS) and the controlling entity which can be either Master subsystem (MSS R4F) and/or DSP subsystem (DSS C674x, AR16xx 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 BSS. It provides a well defined interface for the application to plug in the correct communication driver APIs to communicate with the BSS. 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\ .

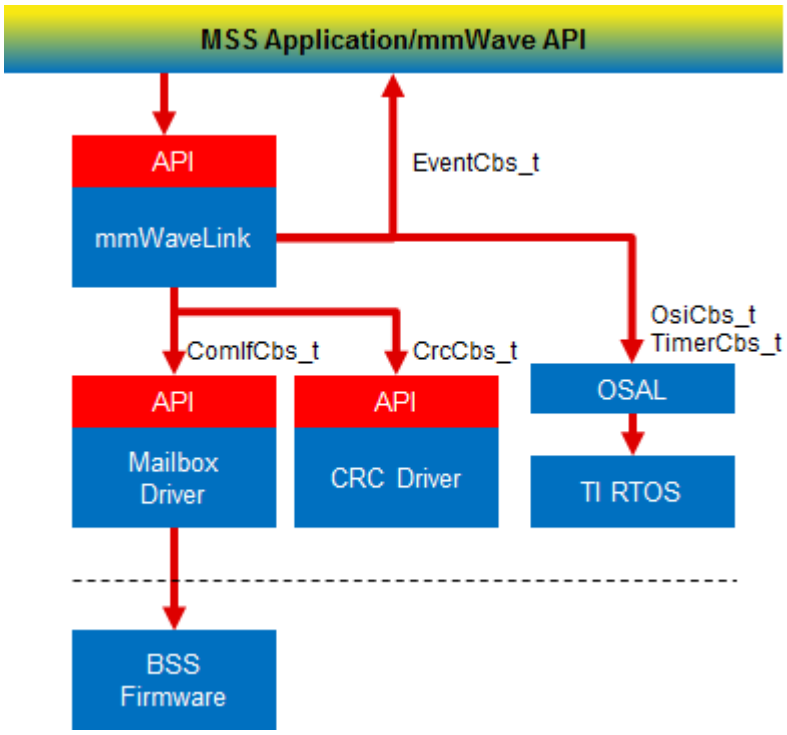


Figure 19: mmWaveLink - Internal software design

5. 3. 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. 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\ .

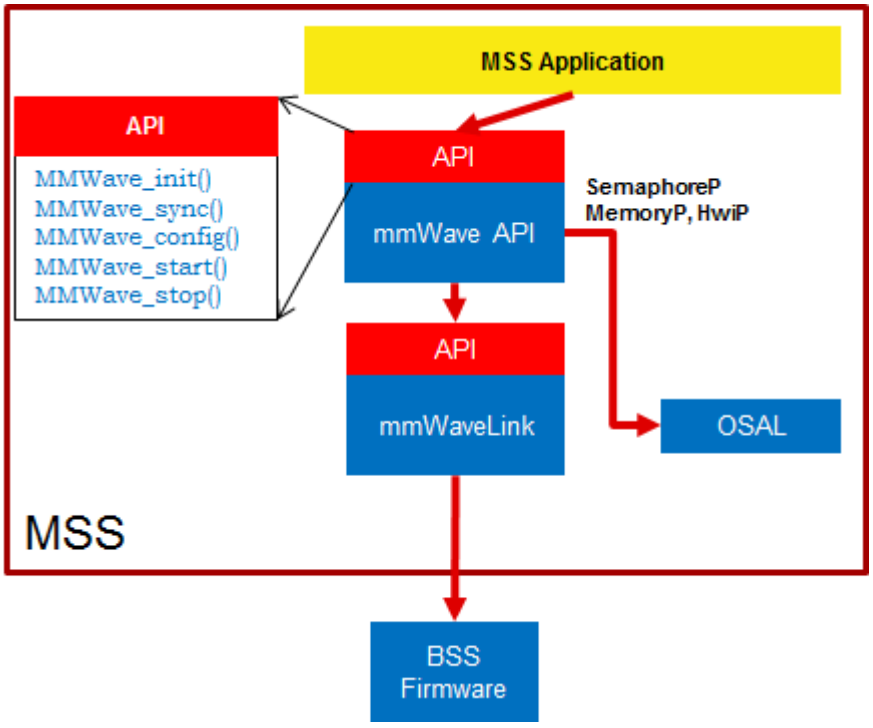


Figure 20: mmWave API - Internal software design

5. 3. 5. mmWaveLib

mmWaveLib is a collection of algorithms that provide basic functionality needed for FMCW radar-cube processing. This component is available for AR16xx 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\ .

5. 3. 6. BSS Firmware

This is a binary (mmwave_sdk_<ver>\firmware\bss) that runs on BIST subsystem of the AR14xx/AR16xx 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. BSS 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.

5. 3. 7. Board Setup and Flash Utilities

This component is a container for all the board setup utilities that would be required by the customer to program the AR14xx/AR16xx device:

- Device support package
  - See more details under "How to connect AR14xx/AR16xx EVM to CCS using JTAG"
- Programming flash utilities
  - See more details under "How to flash an image onto AR14xx/AR16xx EVM"
- CCS debug binary
  - 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 CCSdevelopmentmode for more details.

5. 3. 8. 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. 3. 8. 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. 3. 8. 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 AR16xx 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 AR16xx MSS provides this functionality under its hood.

System Clock

To enable selection of system frequency to use "closed loop APLL", mmWave SDK provides an API: SOC_waitAPLLCalibration(). This function unhalts the BSS and then spins around waiting for acknowledgement from the BSS 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 AR14xx and AR16xx 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: AR14xx) 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 (AR16xx C674)

The cacheability property of the various regions as seen by the DSP (C674x in AR16xx) 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. 3. 8. 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.

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.out and .bin files.

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

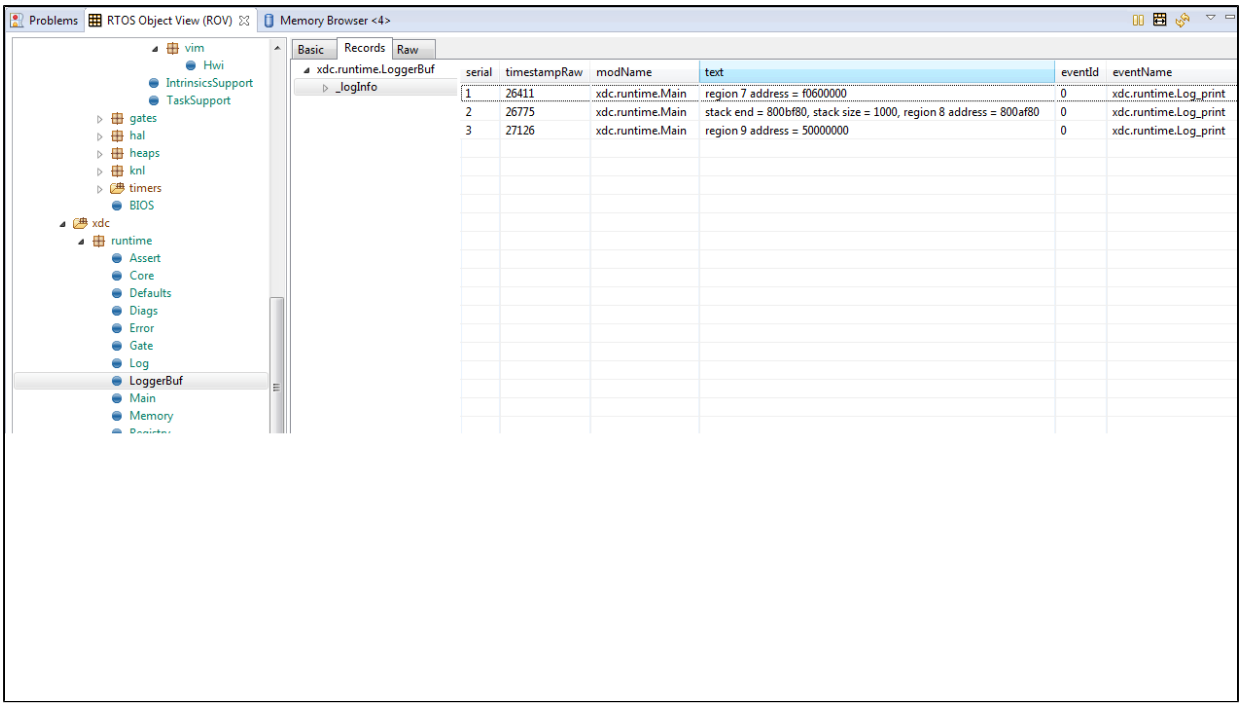
```
var Log       = xdc.useModule('xdc.runtime.Log');
var Main = xdc.useModule('xdc.runtime.Main');
var Diags = xdc.useModule('xdc.runtime.Diags');
var LoggerBuf                   = xdc.useModule('xdc.runtime.LoggerBuf');
LoggerBuf.TimestampProxy        = xdc.useModule('xdc.runtime.Timestamp');

/* 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 (AR1642)

Existing SDK demos (capture, mmw) for AR1642 assigns 5 banks of L3 memory to DSS (i.e. 640KB) and 1 bank of L3 memory to BSS (128KB). 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 "AR1xxx 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 AR1642 device determines the shared memory allocation. So in CCS development mode, its the allocation defined in ccsdebug image that applies and not the application that you download via CCS.

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

- Linker command files in mmwave_sdk_<ver>\packages\ti\platform\ar16xx show the 640KB allocation (0xA0000) for the L3 memory section
- Makefiles for the following components use the value 0x01000005 for SHMEM_ALLOC parameter when invoking the generateMetaImage script. (See AR1xxx Image Creator section)
  - mmwave_sdk_<ver>\packages\ti\utils\ccsdebug
  - mmwave_sdk_<ver>\packages\ti\demo\ar16xx\mmw
  - mmwave_sdk_<ver>\packages\ti\demo\ar16xx\capture

6. 5. AR1xxx 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 AR1xxx 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. AR14xx

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=ar14xx** 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 AR14xx/AR16xx EVM)

6. 5. 2. AR16xx

Application Image generation is two-step process for AR16xx. Refer to "AR1xxx 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=ar16xx** and **MMWAVE_SDK_INSTALL_PATH=mmwave_sdk_<ver>\packages** in your environment before calling this script. This script needs 2 parameters:
   - .out file : this is the input file and the parameter represents the file generated after the link step for the application (application executable).
   - .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 BSS binaries into one metaImage and appending correct CRC. Set **MMWAVE_SDK_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 BSS,TCMB,TCMA and DSS. Refer to the the TRM on details on L3 shared memory layout and "AR1xxx 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] BSS input image in RPRC (,bin) format, use keyword NULL if not needed. Use mmwave_sdk_<ver>\firmware\bss\ar1xxx_bss_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 AR14xx/AR16xx EVM)

6. 6. AR16xx 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;
```