

# TI mmWave Labs

## Driver Vital Signs – Developer's Guide

# Lab Overview

- This lab exercise demonstrates the ability of AWR-1642 TI-mmWave sensor to measure body displacements due to breathing and heart beat
- Typical body surface displacement parameters due to breathing and heart-beat are

		From Front	From Back
Vital Signs	Frequency	Amplitude	Amplitude
Breathing Rate (Adults)	0.1 – 0.5 Hz	~ 1- 12 mm	~ 0.1 – 0.5 mm
Heart Rate (Adults)	0.8 – 2.0 Hz	~ 0.1 – 0.5 mm	~ 0.01 – 0.2 mm

- To measure these small scale vibrations/displacements, we measure the change in phase of the FMCW signal with time at the target range bin

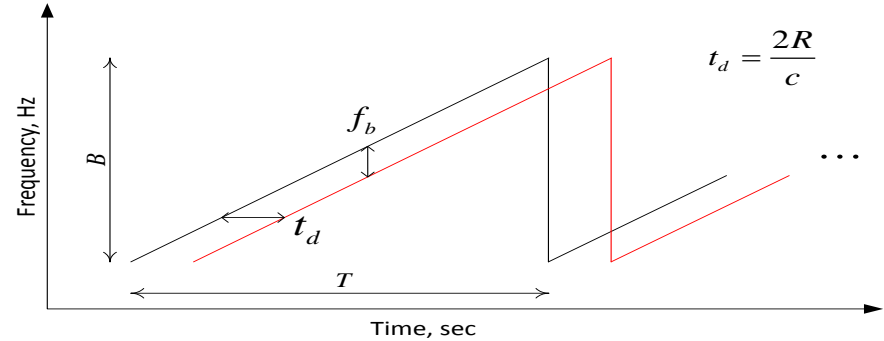
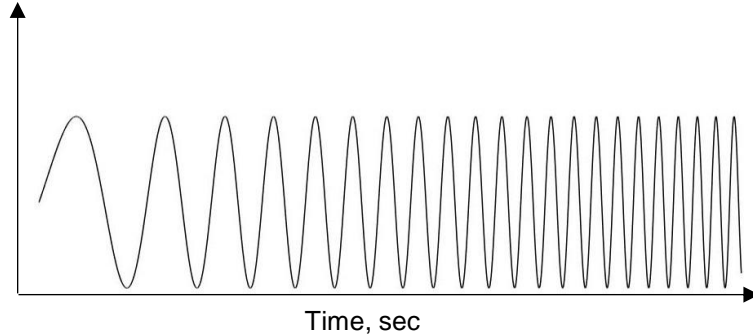
$$\Delta\phi_b = \frac{4\pi}{\lambda} \Delta R$$

-  $\Delta\phi_b$  corresponds to the change in phase when the target moves a distance  $\Delta R$   
- Note that a smaller wavelength  $\lambda$  will give better displacement sensitivity

- Code Composer Studio (CCS) project along with source code is provided for this lab
- Pre-built binary files are also provided that can be loaded on to the AWR-1642 EVM

# FMCW Radar Basics

- Periodic linearly-increasing frequency chirps (known as **Frequency-Modulated Continuous Wave (FMCW)**) are transmitted by radar towards the object



- Transmitted FMCW signal is given by  $s(t) = e^{j\left(2\pi f_c t + \pi \frac{B}{T} t^2\right)}$
- Signal at the receiver is a delayed version of the transmitted signal  $r(t) = e^{j\left(2\pi f_c (t-t_d) + \pi \frac{B}{T} (t-t_d)^2\right)}$
- The received signal from an object at range R after mixing and filtering is given by

$$s(t) \cdot r(t) \approx e^{j\left(4\pi \frac{BR}{cT} t + \frac{4\pi}{\lambda} R\right)} = e^{j(f_b t + \phi_b)}$$

# FMCW Radar – Vital signs Measurements

- Note that for a single object, the beat signal  $b(t)$  is a sinusoidal and has both frequency  $f_b$  and phase  $\phi_b$

$$b(t) = e^{j\left(\underbrace{4\pi \frac{BR}{cT}}_{f_b} t + \underbrace{\frac{4\pi}{\lambda} R}_{\phi_b}\right)} = e^{j(f_b t + \phi_b)}$$

- To measure small scale vibrations, we measure the change in phase of the FMCW signal with time at the object range bin. If an object moves a distance  $\Delta R$  then the change in phase between consecutive measurements is given by

$$\Delta\phi_b = \frac{4\pi}{\lambda} \Delta R$$

As an example at  $\lambda=4$  mm when we have displacements as small as  $\Delta R = 1$  mm, the corresponding phase change is  $\Delta\phi_b = \pi$

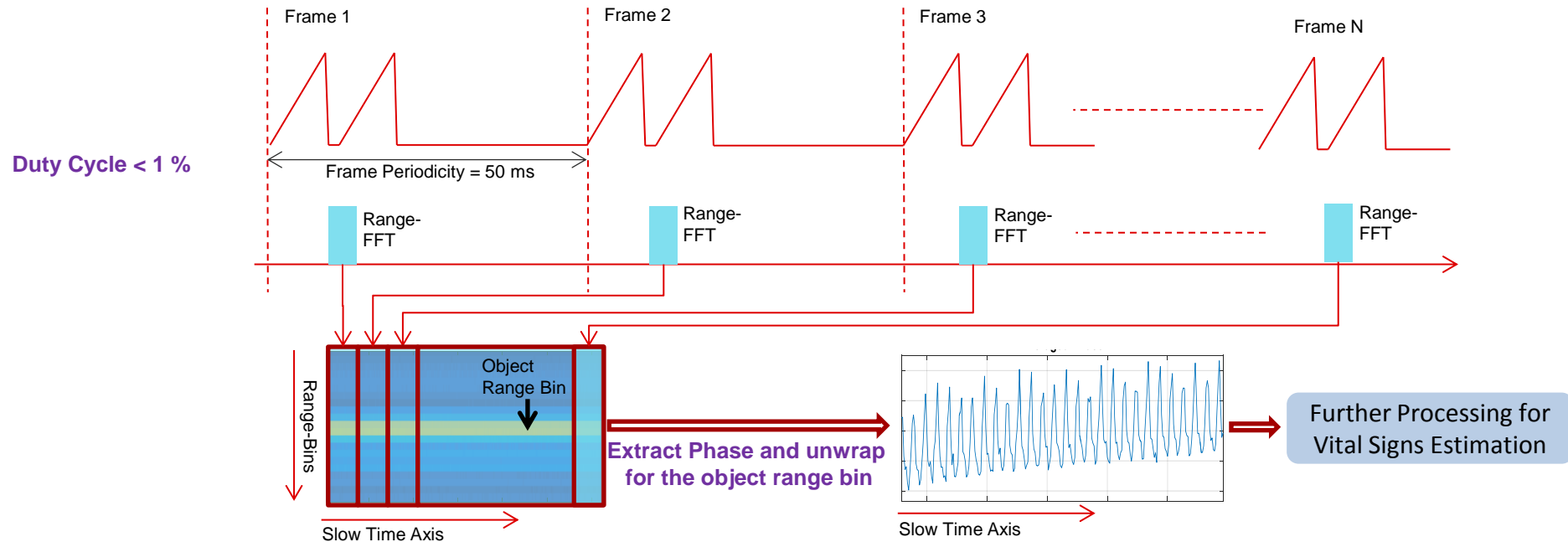
- Phase can be measured by taking the FFT of the beat signal  $b(t)$  and computing the phase at the object range-bin.
- Suppose we take the FFT and the object is at range-bin  $m$ , then the vibration signal  $x(t)$  can be extracted by measuring the phase at range-bin  $m$  at time indices  $nT_s$ , where  $n$  is the chirp index and  $T_s$  is the time between consecutive measurements

$$x(m, nT_s) = \frac{\lambda}{4\pi} \phi_b(m, nT_s)$$

Note that we are assuming that the vibrations  $x(t)$  are small so that the object remains in the same range-bin during the duration of the measurements

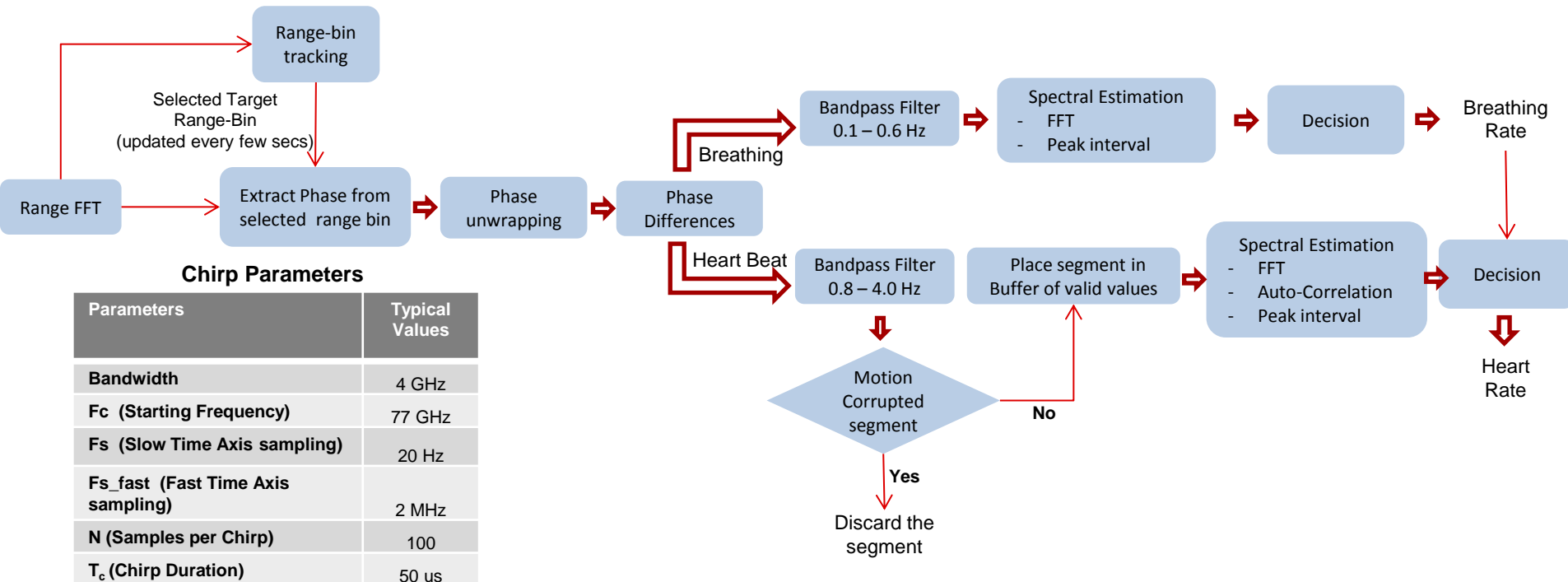
# Chirp Configuration for Demo

- 100 ADC Samples per chirp. Chirp duration is 50  $\mu$ s based on the IF sampling rate of 2 MHz
- Each frame is configured to have 2 chirps. However only the 1st Chirp in the frame is used for processing
- A single TX-RX antenna pair is currently used for processing (Although all the RX antennas are enabled)
- Vital signs waveform is sampled along the “slow time axis” hence the vital signs sampling rate is equal to the Frame-rate of system



# Implementation on the AWR-1642

- Real-time implementation (20 fps) on the C674x DSP Processing Core
- Processing done over a running window of  $T \sim 16$  seconds. New estimates are updated every 1 second
- Memory Requirements  $\sim 16$  kB, CPU Processing time for a single estimate  $\sim 4$  ms



# Block Diagram Description

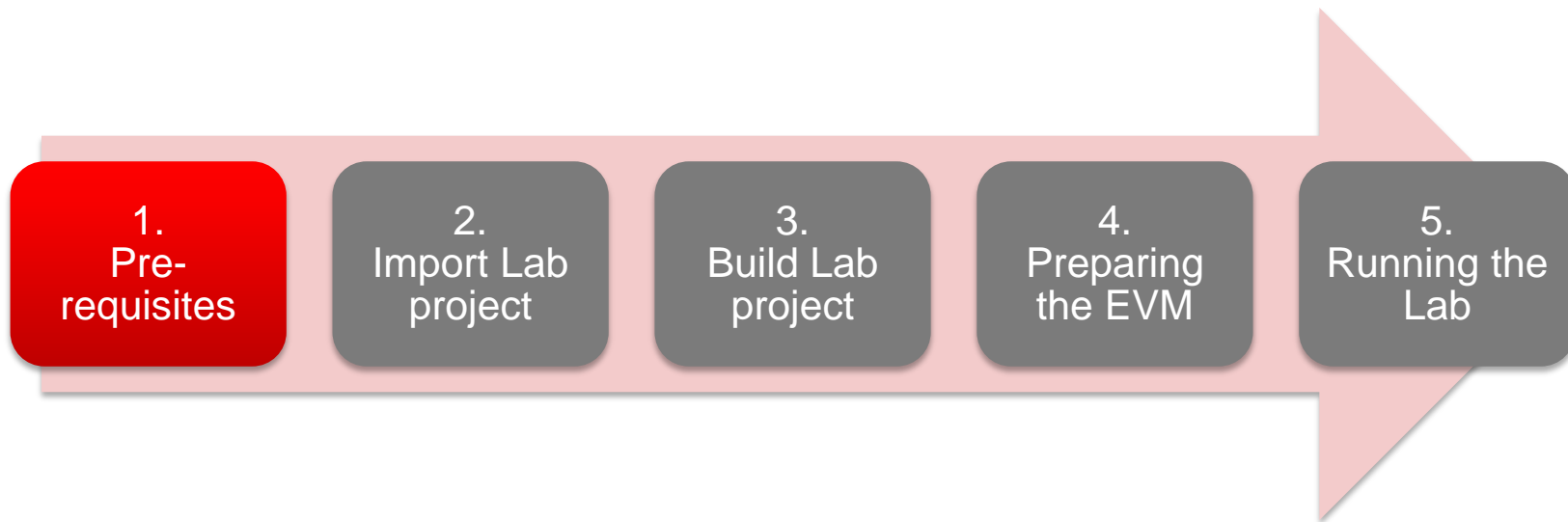
- **Range FFT** : A Fast Fourier Transform (FFT) is performed on the ADC data to obtain the range profile. The magnitude of the range-profile is displayed on the PC-GUI.
- **Target Range-bin** : The range-bin corresponding to the target is found by finding the max-value in the range profile within the user-specified range limits.
- **Phase Extraction** : The phase value of the selected range-bin is computed from the complex range profile data and these phase values are measured over time. The assumption is that the subject is in the same range-bin throughout these measurements. If the subject moves to a different range-bin then it will take a few seconds before the algorithm locks into the new target range-bin.
- **Phase Unwrapping** : Phase values are between  $[-\pi, \pi]$  and need to be unwrapped to obtain the actual displacement profiles. Phase unwrapping is performed by adding/subtracting  $2\pi$  from the phase whenever the phase difference between consecutive values is greater/less than  $\pm \pi$ . The unwrapped phase is displayed as the chest displacement on the PC-GUI.
- **Phase Difference** : The phase difference operation is performed on the unwrapped phase by subtracting successive phase values. This helps in enhancing the heart-beat signal and removing any phase drifts.
- **Impulsive Noise Removal** : The un-wrapped differential phase might be corrupted by several noise-induced phase wrapping errors. This impulse-like noise is removed by computing a forward  $a(m)-a(m+1)$  and backward  $a(m)-a(m-1)$  phase difference for each  $a(m)$  and if these exceed a certain threshold then  $a(m)$  is replaced by an interpolated value.

# Block Diagram Description

- **Bandpass Filtering** : The phase values (after unwrapping and phase differences) are passed through two band-pass filters (serially-cascaded Bi-Quad IIR filter). These band-pass filters operate in real-time input data to generate a continuous stream of output data. The data after band-pass filtering is displayed as the breathing waveform and heart waveform in the PC-GUI.
- **Motion corrupted segment removal/Gain Control** : The purpose of this block is to reduce the impact of any large amplitude movements on the heart-rate estimates. The waveform is divided into segment of  $L=20$  samples (corresponding to 1 sec). If the energy within this data segment exceeds a user-defined threshold ( $E > E_{Th}$ ), then all the samples in that data segment/block are either scaled by  $\sqrt{E_{Th}/E}$  or are alternatively discarded from the time-domain cardiac waveform.
- **Vital Signs Waveforms** : Band-pass filter outputs are stored in the breathing-waveform and cardiac-waveform buffer. Pre-processing steps such as windowing, gain control can be done on these prior to spectral estimation.
- **Spectral Estimation** : These buffers are passed on to the spectral estimation block. Several different types of spectral estimation techniques can be implemented. The current implementation provides a FFT, auto-correlation and an estimate based on the inter-peak distances in the time-domain waveforms to estimate the vital signs.
- **Vital Signs Decision** : The final heart-rate and breathing-rate decisions are made based on the confidence metric from different spectral estimation methods.



# Steps for Building from the Source Code and Run



# 1. Pre-requisites

## 1. Install Pre-requisites

2

3

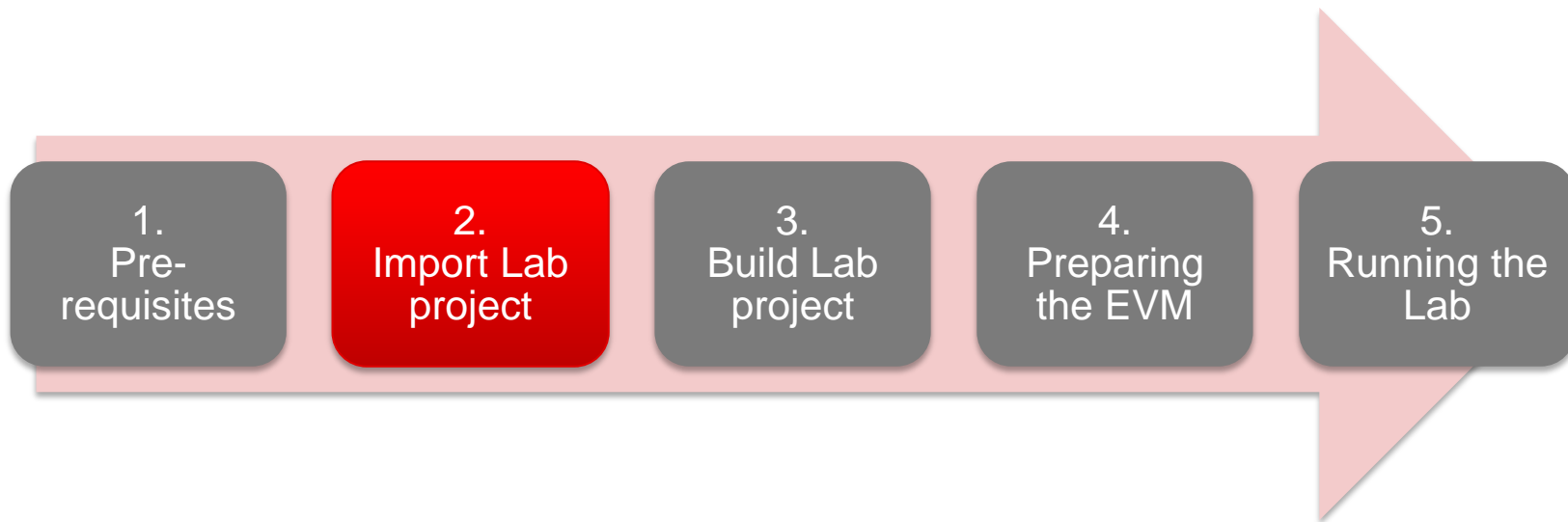
4

5




- It is assumed that you have the TI mmWave SDK 1.1.0.2 and all the related tools installed as mentioned in the mmWave SDK release notes.
  - The mmWave SDK release notes include the links for downloading the required versions of the above tools.
- If you have already installed the mmWave SDK and all the required tools, you can move on to the next step i.e. downloading the lab on to your machine.

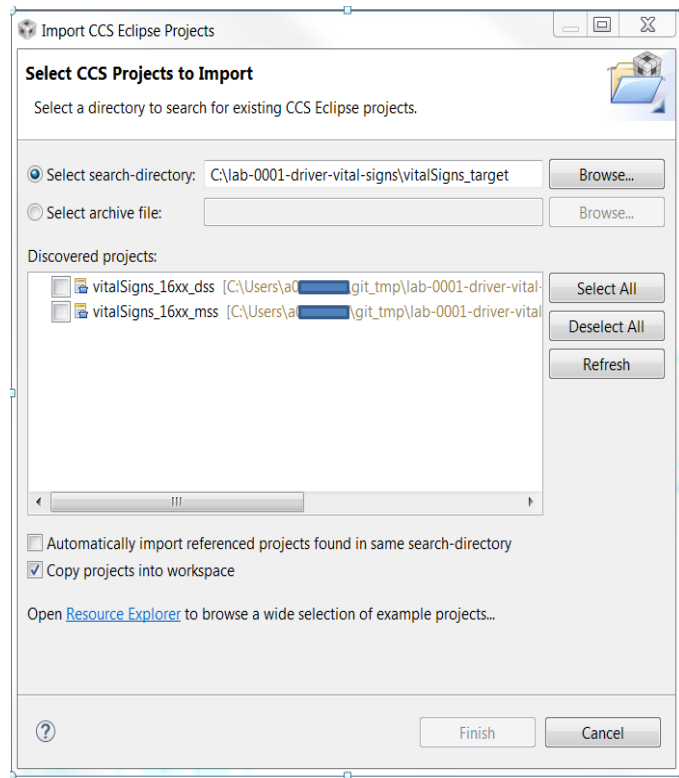
Tool	Version	Download link
CCS	7.1 or later	<a href="#">download link</a> Please note that CCS v7.1 or later is mandatory. CCSv6.x cannot be used
TI SYS/BIOS	6.52.00.12	Included in mmwave sdk installer
TI ARM compiler	16.9.1.LTS	Included in mmwave sdk installer
TI CGT compiler	8.1.3	Included in mmwave sdk installer
XDC	3.50.00.10	Included in mmwave sdk installer
C64x+ DSPLIB	3.4.0.0	Included in mmwave sdk installer
C674x DSPLIB	3.4.0.0	Included in mmwave sdk installer
C674x MATHLIB (little-endian, elf/coff format)	3.1.2.1	Included in mmwave sdk installer
Mono JIT compiler	3.2.8	Only for Linux builds
mmwave device support packages	1.5.3 or later	Upgrade to the latest using CCS update process (see SDK user guide for more details)
TI Emulators package	6.0.0576.0 or later	Upgrade to the latest using CCS update process (see SDK user guide for more details)

# Steps



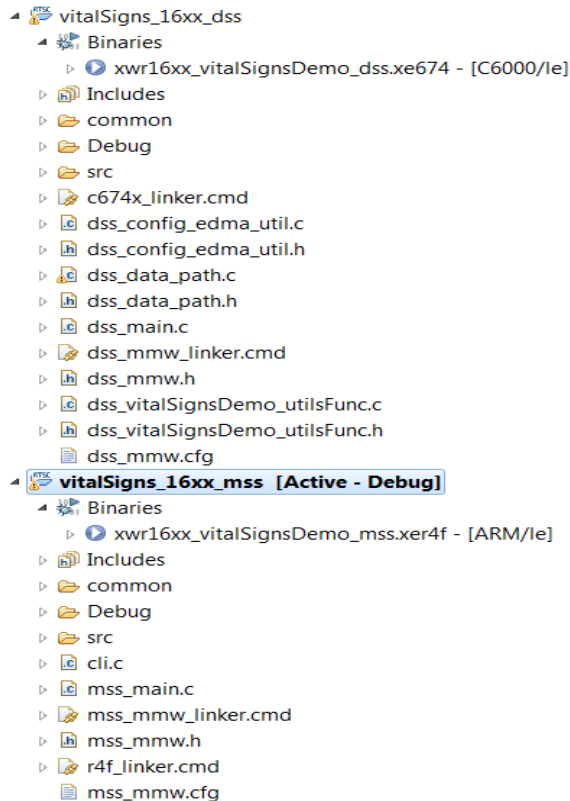
## 2. Import Lab Project

- The **Driver Vital Signs** Lab consists of two CCS projects, one for the R4F core and one for the C674x DSP core
- The CCS projects are included in the **Automotive Toolbox** zip file installed as described in   
- Open CCS. Select the “CCS Edit” perspective
- Select **Project → Import CCS Projects**. Browse to select the lab folder
- The labs projects should be discovered by CCS. **Select All** and **Finish**
- This copies the projects in the user’s workspace and imports it into the CCS project explorer.
  - It is important to note that the copy created in the workspace is the one that gets imported in CCS. The original project downloaded in mmwave automotive toolbox is not touched.

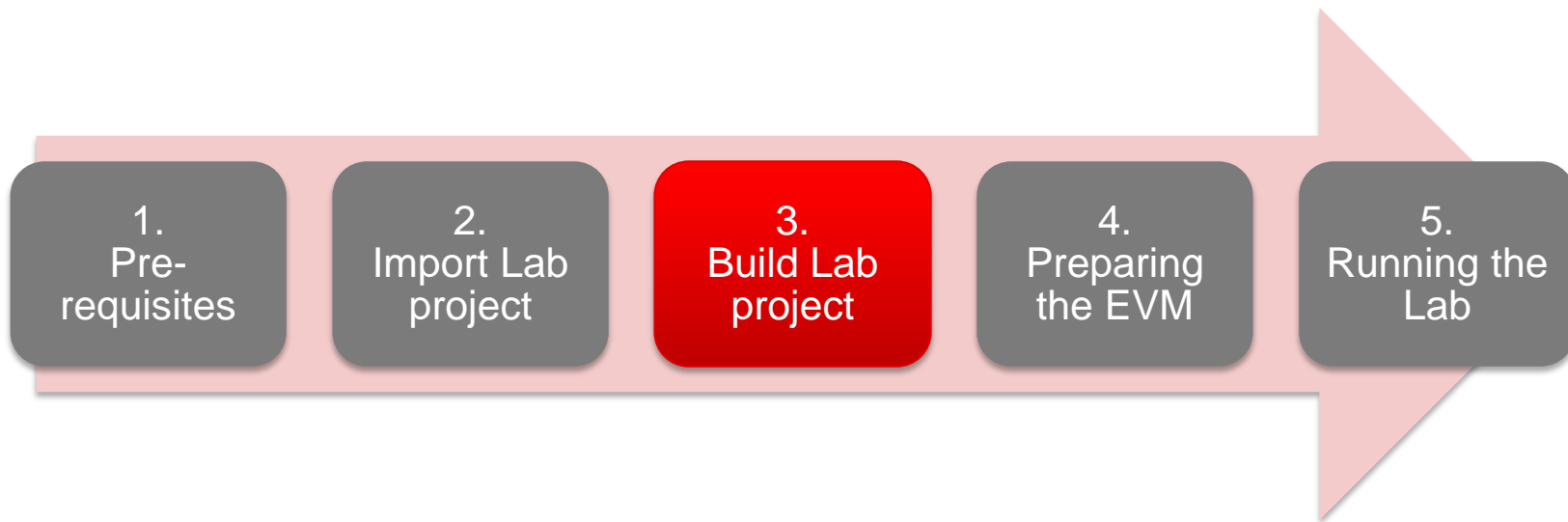


## 2. Import - continued

- The projects should be visible in CCS Project Explorer as shown here.
- We are ready to move on to the next step i.e. Building the project.



# Steps

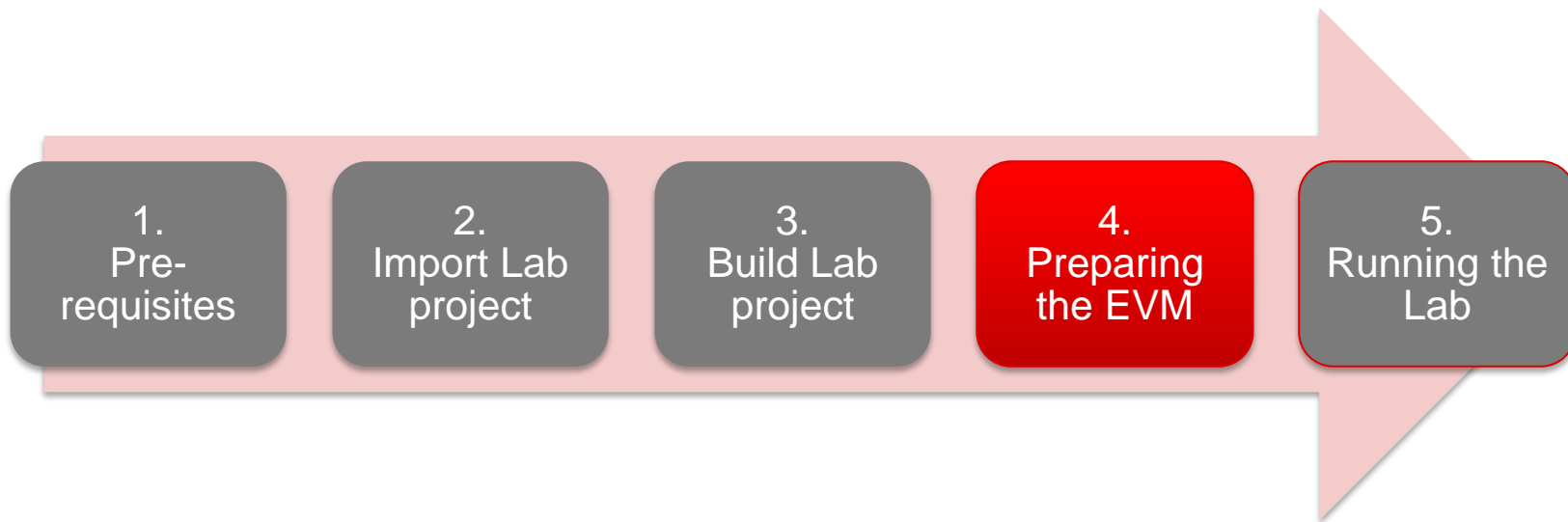


### 3. Build the Lab



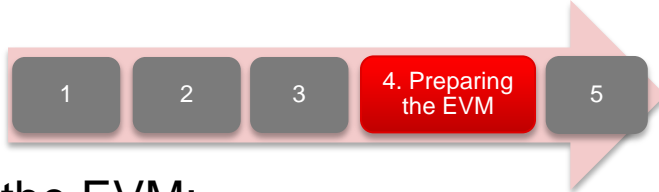
- At this point we assume the two projects have been imported in CCS. If they have not please go to Step 2.
- With the **dss** project selected in Project Explorer, right click on the project and select **Rebuild Project**.
  - Selecting **Rebuild** instead of **Build** ensures that the project is always re-compiled. This is especially important in case the previous build failed with errors.
- On successful completion of the build, you should see the output in CCS console as shown here and the following two files should be produced in the project debug directory
  - xwr16xx\_vitalSignsDemo\_dss.xe674
  - xwr16xx\_vitalSignsDemo\_dss.bin
- If the build fails with errors, please ensure that all the pre-requisites are installed as mentioned in the mmWave SDK release notes.
- The **dss** project must be built BEFORE the **mss** project.
- With the **mss** project selected in Project Explorer, right click on the project and select **Rebuild Project**
- On successful completion of the build, you should see the output in CCS console as shown here and the following three files should be produced in the project debug directory
  - xwr16xx\_vitalSignsDemo\_lab.bin
  - xwr16xx\_vitalSignsDemo\_mss.bin
  - xwr16xx\_vitalSignsDemo\_mss.xer4f
- If the build fails with errors, please ensure that all the pre-requisites are installed as mentioned in the mmWave SDK release notes.

# Steps



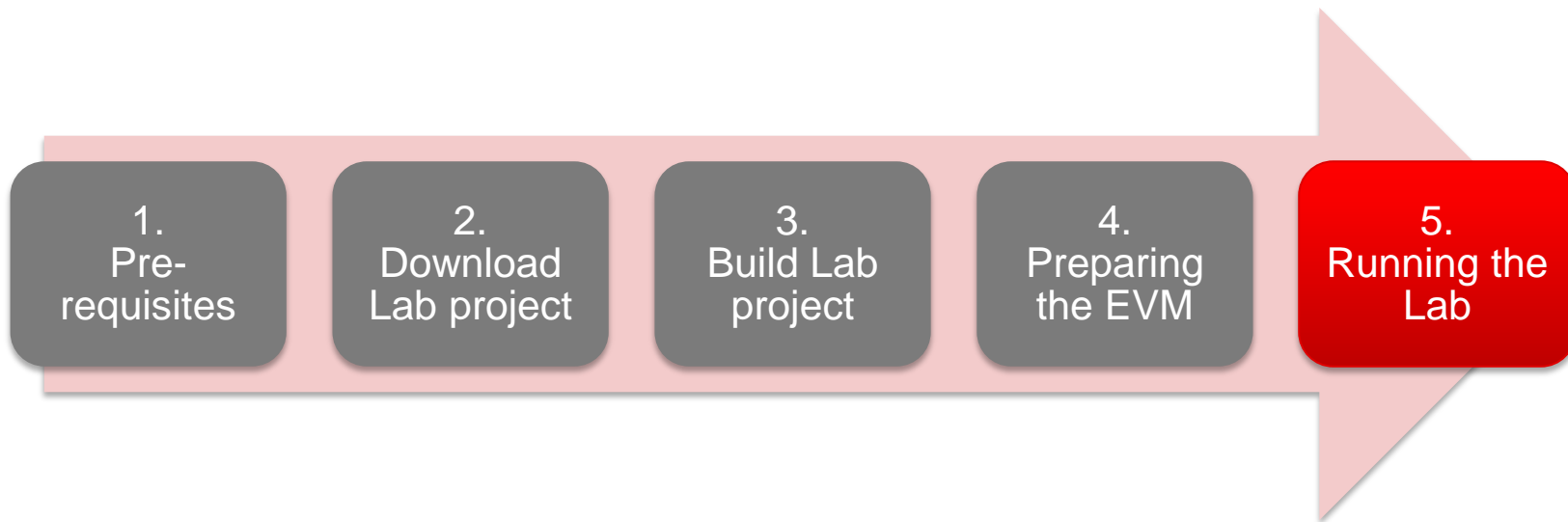


## 4. Preparing the EVM



- There are two ways to execute the compiled code on the EVM:
  - Deployment mode : Flashing the binary (.bin image) on to the EVM serial flash
    - In this mode, the EVM boots autonomously from flash and starts running the bin image.
  - Debug mode: Downloading and running the executable (.xer4f image) from CCS.
    - You will need to flash a small CCS debug firmware on the EVM (one time) to allow connecting with CCS. This debug firmware image is provided with the mmWave SDK.
- The Getting Started Guide has demonstrated the deployment mode
- The following presentation explains the second method i.e. Debug mode (CCS).
  - To prepare the EVM for debug mode, we start with flashing the CCS debug firmware image.
  - Please use the flashing process described in the Getting Started Guide to flash the following binary to the EVM.
    - **C:\ti\mmwave\_sdk\_xx\_xx\_xx\_xx\packages\ti\utils\ccsdebug\xwr16xx\_ccsdebug.bin**

# Steps



# 5. Connecting EVM to CCS

1

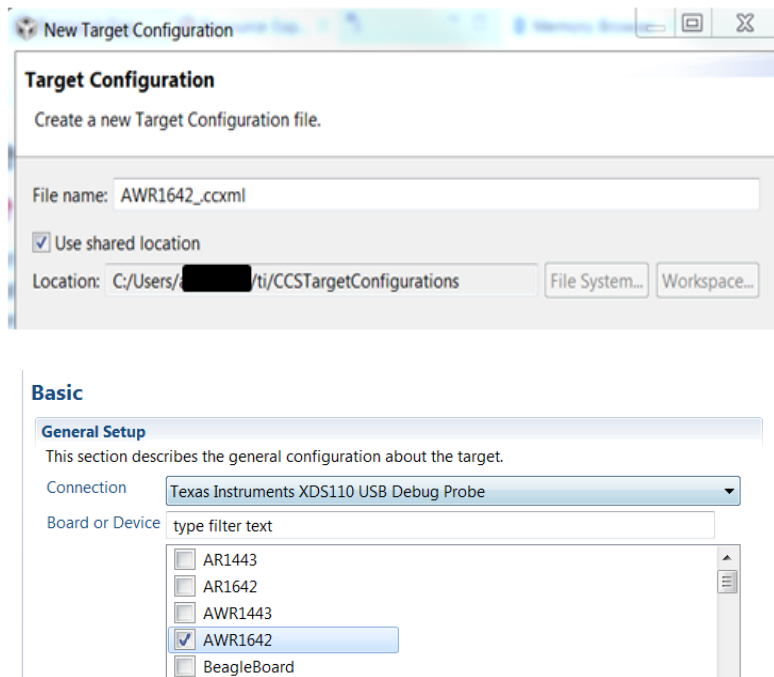
2

3

4

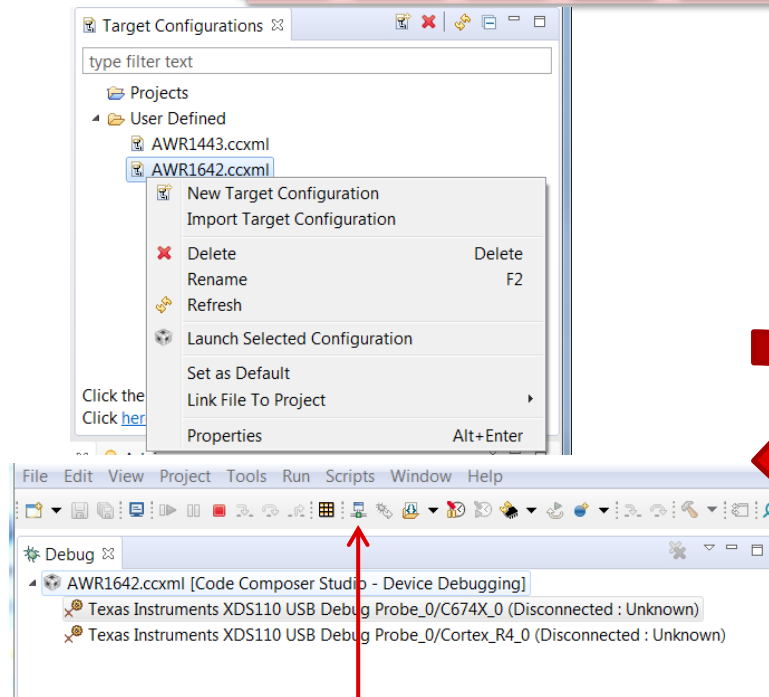
5. Running the Lab

- It is assumed that you were able to download and build the Lab in CCS (completed steps 1, 2 and 3)
- To connect the Radar EVM to CCS, we need to create a target configuration
  - Go to File ► New ► New Target Configuration File
  - Name the target configuration accordingly and check the “Use shared location” checkbox. Press Finish
  - In the configuration editor window:
    - Select “Texas Instruments XDS110 USB Debug Probe” for **Connection**
    - Select AWR1642 device in the **Board or Device** text box.
    - Press the **Save** button to save the target configuration.
    - You can press the **Test Connection** button to check the connection with the board.



# 5. Connecting - continued

- Go to **View ► Target Configurations** to open the target configuration window.
- You should see your target configuration under **User Defined** configurations.
- Right click on the target configuration and select **Launch Select Configuration**.
- This will launch the target configuration in the debug window.
- Select the Texas Instruments XDS110 USB Debug probe/C674X\_0 and press the **Connect Target** button
- Select the Texas Instruments XDS110 USB Debug probe/Cortex\_R4\_0 and press the **Connect Target** button



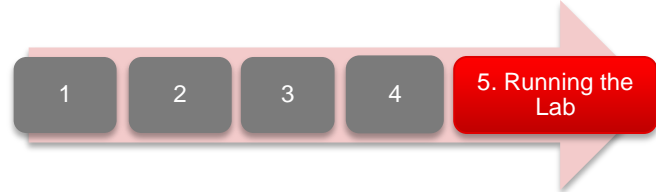
Click here to Connect  
to the target CPU

## 5. Loading the binary



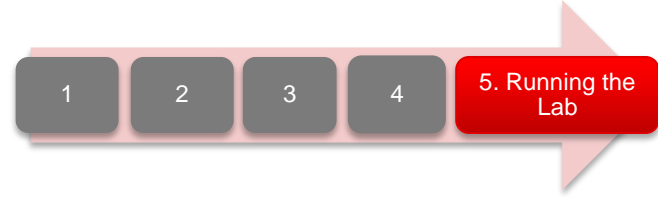
- Once both targets are connected, select the C674X\_0 target, and click on the **Load** button in the toolbar
- In the **Load Program** dialog, press the **Browse Project** button .
- Select the lab executable (.xe674) found in the project as shown, and press OK.
- Press OK again in the **Load Program** dialog.


## 5. Loading the binary



- Now select the Cortex\_R4\_0 target, and click on the **Load** button in the toolbar
- In the **Load Program** dialog, press the **Browse Project** button .
- Select the lab executable (.xer4f) found in the project and press OK.
- Press OK again in the **Load Program** dialog.

# 5. Running the binary



- With both executables loaded, select `mss_main.c` and press the Run/Resume button 
- The program should start executing and generate console output.
- If everything goes fine, you should see the “MMWDDemoMSS mmWave Control Initialization was successful” message which indicates that the program is waiting for the DSS to be started
- Select `dss_main.c`, as shown, and press the Run/Resume button
- Further console output should be generated as shown.
- You should see the “CLI is operational” message which indicates that the program is ready and waiting for the sensor configuration
- The sensor configuration is sent using the GUI. Follow Getting Started Guide for more details on placing the sensor and starting the GUI.

# Running the Lab PC-GUI

1

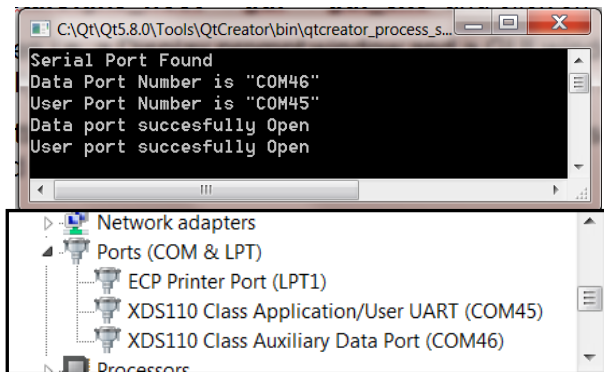
2

3

4

5. Running the Lab

1. Navigate to the folder **vitalSigns\_host ► gui ► gui\_exe** and click on **VitalSignsRadar\_Demo.exe**
2. Two windows should open i.e. a Display prompt window and a GUI window. If the EVM is connected to the PC, then the display prompt window should successfully open the COM ports (to double check, make sure they match with the port numbers on the Device Manager).
3. In the GUI window, the **User UART COM Port** and **Data COM Port** fields should automatically be filled with the correct port numbers (Make sure that no other EVM is connected to the USB ports of the PC)



If the GUI does not open you might need the vc runtime which can be downloaded from the link below  
<https://www.microsoft.com/en-us/download/details.aspx?id=48145>



COM ports  
(The GUI should automatically fill these fields)



# Running the Lab PC-GUI

1

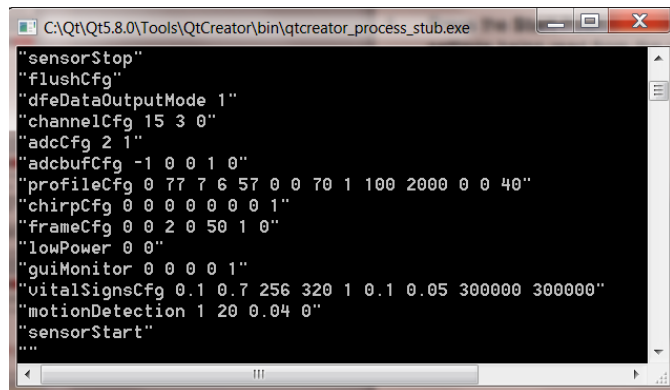
2

3

4

5. Running the Lab

1. Press the **Start** Push button in the GUI. In the Display Prompt window you should see the configuration settings being read from the configuration text file and sent through the UART to the EVM
2. As soon as the **sensorStart** command is sent, the GUI should start displaying the data
3. Please follow the Getting Started Guide for more details on placing the sensor and starting the GUI.



```
CA\Qt5.8.0\Tools\QtCreator\bin\qtcreator_process_stub.exe
"sensorStop"
"flushCfg"
"dfcDataOutputMode 1"
"channelCfg 15 3 0"
"adcCfg 2 1"
"adcbufCfg -1 0 0 1 0"
"profileCfg 0 77 7 6 57 0 0 70 1 100 2000 0 0 40"
"chirpCfg 0 0 0 0 0 0 1"
"frameCfg 0 0 2 0 50 1 0"
"lowPower 0 0"
"guiMonitor 0 0 0 0 1"
"vitalSignsCfg 0.1 0.7 256 320 1 0.1 0.05 300000 300000"
"motionDetection 1 20 0.04 0"
"sensorStart"
***
```



# Running GUI – (Configuration)

- The mmWave sensor device and algorithm configurations are set through the configuration text file. These files are located in vitalSigns\_host\gui\profiles\
- Configuration commands relevant to the vital signs algorithm are **vitalSignsCfg** and **motionDetection**

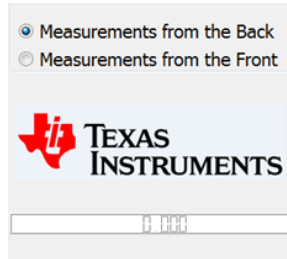
Configuration	Parameters	Values	Comments
vitalSignsCfg	Start Range (meters)	0.1	The subject/person is expected to be within the Start Range and End Ranges. The program searches for the maximum peak within these ranges and assumes that peak corresponds to the subject.
	End Range (meters)	0.7	
	Breathing Waveform Size	256	Specifies the number of points within the waveforms. As an example, given a frame-rate of 20 Hz and 256 number of samples in the waveform then the time duration of the waveform would be $= 256/20 \sim 12.8$ seconds. In general, larger the time duration, better the frequency resolution after the FFT and higher the FFT processing gain. However, due to the inherent time-frequency resolution tradeoff, we lose the ability to measure instantaneous changes in the heart-rate and breathing-rate
	Heart-rate Waveform Size	320	
	Rx-Antenna to Process	1	Rx receiver number to process. Data from a single RX antenna is processed in the current implementation
	Alpha filter value for Breathing waveform energy computation	0.1	Alpha filter values for recursive averaging of the waveform energies based on the equation below where $x(n)$ is the current waveform value while $E(n)$ is the energy. $E(n) = \alpha x^2(n) + (1 - \alpha)E(n - 1)$
	Alpha filter value for heart-beat waveform energy computation	0.05	
	Scale Factor for breathing waveform	300000	Scaling factors to convert waveform values in floating points to 32 bit integers required for the FFT.
	Scale Factor for heart-beat waveform	300000	

# Running GUI – (Configuration)

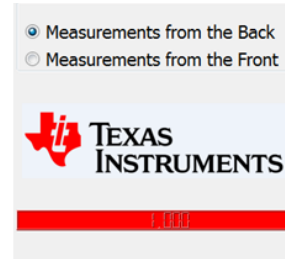
- **motionDetection**

The purpose of this block is to discard the data segments that might be corrupted by large amplitude movements. The heart waveform is divided into segment of  $L$  samples. If the energy within this data segment exceeds a user-defined threshold  $E_{TH}$  then all the samples are discarded from the time-domain heart waveform.

Configuration	Parameters	Values	Comments
<b>motionDetection</b>	Enable	1	0: Disable the Block 1: Enable the Block
	Data segments Length (L)	20	Data segment over which the energy is computed
	Threshold ( $E_{TH}$ )	0.04	Energy threshold value. If the energy in the data segment length exceeds this value then the data segment is discarded
	Gain Control	0	0: Disable Gain Control 1: Enable Gain Control



Data segment valid



Data segment discarded

Note : The display panel below the TEXAS INSTRUMENTS icon will turn RED if the current data segment is discarded due to motion corruption.

# Limitations of the Current Demo

- The user has to be relatively still for at least **10-15 seconds** for the demo to effectively work
- The breathing signal from the back might be very weak for some people.
- The heart-rate value might jump during measurements. This can be due to several reasons (e.g. noise, alignment issues, interference from other objects, breathing harmonics overlapping with the heart rate frequency etc. ). If the subject stays stationary, the heart-rate values ultimately should converge to the correct value
- One reason the heart rate might display a wrong value is the presence of breathing harmonic overlapping the heart-rate spectrum region i.e.  $[0.8 - 2.0]$  Hz. In the current demo the 2<sup>nd</sup> breathing harmonic is cancelled. For example if the person has a breathing rate of 26 bpm and the heart rate happens to be  $\sim 52$  bpm it will be discarded as the algorithm will interpret this as a breathing harmonic rather than a correct heart-rate

# Learn more about TI mmWave Sensors

- Learn more about xWR1x devices, please visit the product pages
  - IWR1443: <http://www.ti.com/product/IWR1443>
  - IWR1642: <http://www.ti.com/product/IWR1642>
  - AWR1443: <http://www.ti.com/product/AWR1443>
  - AWR1642: <http://www.ti.com/product/AWR1642>
- Get started evaluating the platform with xWR1x EVMs, purchase EVM at
  - IWR1443 EVM: <http://www.ti.com/tool/IWR1443BOOST>
  - IWR1642 EVM: <http://www.ti.com/tool/IWR1642BOOST>
  - AWR1443 EVM: <http://www.ti.com/tool/AWR1443BOOST>
  - AWR1642 EVM: <http://www.ti.com/tool/AWR1642BOOST>
- Download mmWave SDK @ <http://www.ti.com/tool/MMWAVE-SDK>
- Ask question on TI's E2E forum @ <http://e2e.ti.com>



© Copyright 2017 Texas Instruments Incorporated. All rights reserved.

This material is provided strictly “as-is,” for informational purposes only, and without any warranty.  
Use of this material is subject to TI’s **Terms of Use**, viewable at [TI.com](http://TI.com)