

SCHOOL *of* BUSINESS AND TECHNOLOGY

Department of Engineering and Aviation Sciences

**Design of a Ground Rover with TI mmWave Radar Sensor for Geographical Acquisition in Disaster Rescue Applications**

**Favour Dada**

**Aduralere Sulaiman**

**Jesudara Omidokun**

Advisor: Dr. Lei Zhang

May, 21, 2019

Design of a Ground Rover with TI mmWave Radar Sensor for Geographical Acquisition in Disaster Rescue Applications

By

Favour Dada, Aduralere Sulaiman, Jesudara Omidokun

Submitted to the Department of Engineering and Aviation Sciences in partial fulfillment of the requirements for the degree of Bachelor of Science in Engineering at the

UNIVERSITY OF MARYLAND EASTERN SHORE

Date May, 21, 2019

The author hereby grants the University of Maryland Eastern Shore permission to reproduce and distribute publicly, paper and or electronic copies of this document in whole or in parts.

Authors Favour Dada, Aduralere Sulaiman, Jesudara Omidokun

Signature

Date May, 21, 2019

Department of Engineering and Aviation Sciences

Certified by

List of Contents

[List of Contents 3](#_Toc9275485)

[List of Figures 2](#_Toc9275486)

[List of Tables 3](#_Toc9275487)

[1. Introduction 4](#_Toc9275488)

[1.1 Backgound/Motivation 4](#_Toc9275489)

[1.2 Objective 5](#_Toc9275490)

[1.3 Design Requirements 6](#_Toc9275491)

[1.4 Design Constraints 6](#_Toc9275492)

[1.5 Design Method (Approach) 6](#_Toc9275493)

[1.6 Standards 6](#_Toc9275494)

[2. Project Description 7](#_Toc9275495)

[2.1 System Description 7](#_Toc9275496)

[2.2 System Diagram 7](#_Toc9275497)

[2.3 System Functions 7](#_Toc9275498)

[3. Implementation Plan 8](#_Toc9275499)

[3.1 Tasks 8](#_Toc9275500)

[3.2 Team Organization 9](#_Toc9275501)

[3.2.1. Responsibility of Team member 1 9](#_Toc9275502)

[3.2.2. Responsibility of Team member 2 9](#_Toc9275503)

[3.2.3. Responsibility of Team member 3 9](#_Toc9275504)

[3.3 Timeline/Milestones/Delivery Plan 9](#_Toc9275505)

[4. Implementation 11](#_Toc9275506)

[4.1 Implementation of Task 1. 11](#_Toc9275507)

[4.1.1. Implementation of Subtask 1.1 11](#_Toc9275508)

[4.1.2. Implementation of Subtask 1.2 20](#_Toc9275509)

[4.1.3. Implementation of Subtask 1.3 20](#_Toc9275510)

[4.2 Implementation of Task 2. 21](#_Toc9275511)

[4.2.1. Implementation of Task 2 Subtask 1 21](#_Toc9275512)

[4.2.2. Implementation of Task 2 subtask 2 22](#_Toc9275513)

[4.3 Implementation of Task 4. 22](#_Toc9275514)

[4.4 Implementation of Task 5. 25](#_Toc9275515)

[4.4.1. Implementation of Task 5 Subtask 1 25](#_Toc9275516)

[4.4.2. Implementation of Task 5 Subtask 2 26](#_Toc9275517)

[Acknowledgment 30](#_Toc9275518)

[Appendix 31](#_Toc9275519)

[A. Source Code. 31](#_Toc9275520)

[1. Source Code of Line Sensor 31](#_Toc9275521)

[2. Source Code of Bump Switches 31](#_Toc9275522)

[3. Source Code of 4k FFT Stiching Algorithm 32](#_Toc9275523)

[4. Source Code of .Obj to .Pcd Converter 47](#_Toc9275524)

[5. Source Code for ROS Installation 47](#_Toc9275525)

[REFERENCES 49](#_Toc9275526)

List of Figures

[Fig. 1 A Robot Used to Diffuse a Bomb 6](#_Toc9261170)

[Fig. 2 A Hornet Drone Used for Engaging Targets 6](#_Toc9261171)

[Fig. 3 Momaro Robot Used in Disaster Rescue 6](#_Toc9261172)

[Fig. 4 System Diagram. 8](#_Toc9261173)

[Fig. 5 Componets Used in Assembling the RSLK 13](#_Toc9261174)

[Fig. 6 Tools Used in Assembling the RSLK 14](#_Toc9261175)

[Fig. 7 Assembled TI RSLK 14](#_Toc9261176)

[Fig. 8 Battery Connections Soldered as Part of the Attachment to the Chassis 15](#_Toc9261177)

[Fig. 9 LaunchPad Soldered Together with Long pins Facing Upward. 16](#_Toc9261178)

[Fig. 10 Encoders Soldered to Motor Drivers. 16](#_Toc9261179)

[Fig. 11 Bump Switches 17](#_Toc9261180)

[Fig. 12 Motor Board on Chassis 17](#_Toc9261181)

[Fig. 13 Ball Caster 18](#_Toc9261182)

[Fig. 14 Wheels of the Car 18](#_Toc9261183)

[Fig. 15 Bump Switches Wiring 19](#_Toc9261184)

[Fig. 16 Motor Board Wiring to LaunchPad 19](#_Toc9261185)

[Fig. 17 Line Sensor Wiring Guide 20](#_Toc9261186)

[Fig. 18 Error Message Received 21](#_Toc9261187)

[Fig. 19 Switching off the Antivirus 21](#_Toc9261188)

[Fig. 20 Assembled TI RSLK 22](#_Toc9261189)

[Fig. 21 Flow Chart for Line Sensor 23](#_Toc9261190)

[Fig. 22 5V Power Supply 24](#_Toc9261191)

[Fig. 23 Schematics of Buck Converter 24](#_Toc9261192)

[Fig. 24 Front view of the IWR1443 25](#_Toc9261193)

[Fig. 25 Back view of the IWR 1443 25](#_Toc9261194)

[Fig. 26 Block diagram of the IWR1443 25](#_Toc9261195)

[Fig. 27 Solution to the 27](#_Toc9261196)

[Fig. 28 3D Stitching of Point Cloud Data 28](#_Toc9261197)

[Fig. 29 Prebuilt in Binaries 29](#_Toc9261198)

[Fig. 30 COMuart Entry 29](#_Toc9261199)

List of Tables

[Table 1. Task Assignment for Spring 2019 11](#_Toc9261200)

[Table 2. Task Assignment for Fall 2019 12](#_Toc9261201)

[Table 3. Components of the Ti Rslk 14](#_Toc9261202)

[Table 4. Bump Switches Connection Guide 19](#_Toc9261203)

[Table 5. Motorboard Wiring Guide 19](#_Toc9261204)

[Table 6. Motorboard Wiring From Ground And Vreg 19](#_Toc9261205)

[Table 7. Line Sensor Wiring Guide 20](#_Toc9261206)

1. Introduction

In this project, we are going to develop an autonomous car using the TI-RSLK and the mmWave sensor (IWR1443BOOST). The autonomous car will be capable of different functions such as providing geographical information (such as range, velocity and angle) and proximity & position sensing.

## Backgound/Motivation

The TI Robotics Systems Learning Kit was created by TI as a low-cost robotics kit to provide students with a profound understanding of how electronic devices work. It was developed in conjunction with Dr. Jon Valvano, professor, electrical and computer engineering at The University of Texas at Austin. There are two different RSLK kits; the basic kit which teaches the foundations of an electric system and the advanced kit which contains all components found in basic kit, adding wireless communication to enable robot to solve its way through a maze by detecting lines and obstacles.

The mmWave sensor is an extremely valuable sensing technology for detection of objects and providing the range, velocity and angle of these objects. It is a contactless-technology which operates in the spectrum between 30GHz and 300GHz. Due to the technology’s use of small wavelengths it can provide sub-mm range accuracy and is able to penetrate certain materials such as plastic, drywall, clothing, and is impervious to environmental conditions such as rain, fog, dust and snow. The mmWave sensor has a clear advantage over other sensing technologies such as ultrasonic sensors and LIDAR. It has a long detection range, narrow and wide detection angle, good range resolution, and has good night operation performance.

The rover can be used in many situations such as in disaster rescuing. In such situations, instead of law enforcement agencies to go in with guns blazing, it is better to access the situation at hand. Many products are on the market, some of which map and maneuver disaster areas after calamities has taken place, assist fire fighters, and many more. The mmWave sensor on the rover can provide 3D point cloud information on the building where the situation occurred. It will also work better than a camera because of its long detection range, it’s narrow and wide detection angle, as well as it is good range resolution. From the 3D point cloud information provided by the mmWave sensor, the rover would be able to provide the best possible route for the officers to go through without being noticed by the “bad guys”.

The purpose of this project is to develop a system that would help with rescuing in disasters that share the functionalities of robots that help in rescuing. As previously stated there are many devices on the market as seen in the figure below and the devices can accomplish similar tasks as our devices. However from a financial standpoint these products cost more than our device.

A picture containing indoor, road

Description automatically generated

Fig. 1 A Robot Used to Diffuse a Bomb

A close up of an animal

Description automatically generated

Fig. 2 A Hornet Drone Used for Engaging Targets



Fig. 3 Momaro Robot Used in Disaster Rescue

## Objective

The objective of the project is to design a rover that can acquire geographical acquisition such as the range, velocity, angle and 3D imaging of objects in an unknown area.

## Design Requirements

1. The device will be able to produce data using a point cloud information.
2. The device will be able to avoid obstacles while providing autonomous navigation
3. The payload of the car will be 15 to 20lbs
4. The device will detect object within the range 3cm to 10 m.
5. The device will have a resolution range of 10cm

## Design Constraints

1. The tires might not be able to move well in certain terrains such as rocky areas.
2. Battery will have to be recharged after about 4 hours of use
3. The device would only work in mild weather conditions.

## Design Method (Approach)

The first step for this design is to solder, assemble, and wire the TI RSLK kit. The second step is to program each of the sensors for the bump sensors, and the line following sensors. The second step is to install Ubuntu 16.04 64 bits on the microcomputer in this case an ODRIOD XU4Q as recommended by TI. TI recommended Ubuntu operating system because the robotic operating system libraries which is needed for navigation and point cloud system only support Ubuntu 16.04 64bit at present. The third step is to design a power supply that generates 5 Volts and 2.5 Amp to power on the mmWave. The fourth step is to implement the mmWave interface with the microcomputer. The fifth step is to implement the mmWave with the RSLK Board. The sixth step is to acquire the point cloud data from the radar using the microcomputer by launching the TI mmWave­\_rospkg node to start the mmWave IWR1443 BOOST EVM. The seventh step is to stitch the 3D point cloud data generated. The eighth step is to implement object detection behind a glass wall. The ninth step is to use a tachometer to calculate the speed of the rover. Finally the tenth step is system finalization, testing, and refinement.

## Standards

1. This project follows the standard [IEC TC 47](http://www.iec.ch/tc47) that puts safety first for disaster rescue.
2. This project follows the standard [IEC TC 2](http://www.iec.ch/tc2) that covers reliability of electronic components.
3. This project follows the standard [IEC TC 91](http://www.iec.ch/tc91) for assembly electronic technologies including components
4. This project follows the international standards [IEC SC 21A](http://www.iec.ch/sc21a) for batteries containing alkaline or other non-acid electrolytes.
5. This project follows the standard DHS, NIST standards for rescue robots.
6. The IWR1443 evaluation module (IWR1443BOOST) is in compliance with Directive 2014/53/EU.
7. Project Description

## System Description

An autonomous robot is controlled by a TI microcontroller MSP432 power by a 12-volt battery with a voltage regulator controlling the voltage input to the microcontrollers. The battery powers the DC motor and sends the DC signal to a PWM input on the microcontroller. An mmWave IWR1443 Boost is connected to the microcontroller to server as a senor to the car. Also, an ODRIOD XU4Q is connected to the output signal of the microcontroller to control, access and save the data acquire from the sensors. The mmWave radar would collect 3D point cloud data, detect objects and humans. The tachometer would calculate the speed of the rover.

## System Diagram

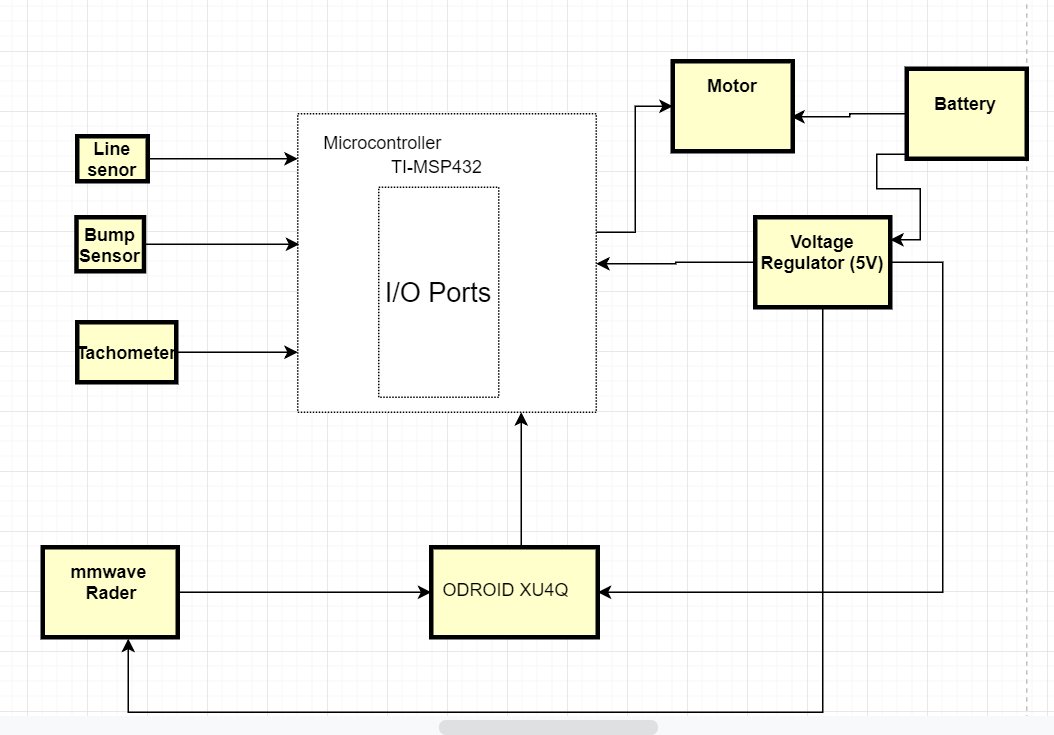


Fig. 4 System Diagram.

## System Functions

The system designed has;

1. Bump switches that edge-triggered interrupts to detect collisions
2. Line sensors that implement a simple line-following algorithm with a finite state machine (FSM)
3. Battery that powers the LaunchPad with 12V
4. Buck converter that steps down the 9v voltage battery supply to 5v and steps up the current.
5. Motor driver that allows control of large loads
6. Tachometer that calculates the speed of the RSLK by explaining the relationship between duty cycle and speed, experiencing the effect of friction.
7. Implementation Plan

## Tasks

* Task 1. RLSK Assembly
  + Subtask 1. Soldering, Assembling and Wiring of TI RSLK Kit
  + Subtask 2. Writing and Modifying the Code for Bumps Sensors
  + Subtask 3. Writing and Modifying of the Code for Line Follow Sensor
* Task 2. Design of the computer system
  + Subtask 1. Setting up Ubuntu on the ODRIOD XU4Q
  + Subtask 2. Design of a power adapter for mmWave
  + Subtask 3. Implementing the mmWave interface with the microcomputer
* Task 3. Implementation and development of the mmWave sensor (IWR 1443 Boost)
  + Subtask 1. Mounting the mmWave to RSLK board
  + Subtask 2. Design and print a 3D part to help mount the mmWave
* Task 4. Acquisition of the point cloud data from the radar with the microcomputer.
  + Subtask 1. Launch the ti\_mmWave\_rospkg node to start the mmWave EVM
  + Subtask 2. Visualize the Radar point cloud using RVI
* Task 5. Stitching Algorithm
  + Subtask 1. Stitching the 3D point cloud data
  + Subtask 2. 4K FFT stitching Algorithm
* Task 6. Detecting and calculating the distance of an object behind the glass wall.
* Task 7. Using the Tachometer to calculate the speed of the rover.
* Task 8. System Finalization, testing and refinement.

## Team Organization

Team member 1: Favour Dada (Electrical Engineering)

Team member 2: Aduralere Sulaiman (Computer Engineering)

Team member 3: Jesudara Omidokun (Electrical Engineering)

### Responsibility of Team member 1

Task 1, Subtask 1.1, 1.3

Task 2, Subtask 2.1, 2.2

Task 5, Subtask 5.1, 5.2

Task 6

Task 8

### Responsibility of Team member 2

Task 1, Subtask 1.2, 1.3,

Task 2, Subtask 2.3

Task 4, Subtask 4.1

Task 6

Task 8

### Responsibility of Team member 3

Task 1, Subtask 1.1, 1.3

Task 3, Subtask 3.1, 3.2

Task 4

Task 7

Task 8

## Timeline/Milestones/Delivery Plan

Table 1. Task Assignment for Spring 2019

|  |  |  |  |
| --- | --- | --- | --- |
| Time | Jesudara | Aduralere | Favour |
| Week 1 | Task 1  1.1 Soldering Components of RSLK task | Task 1  1.2 Write the code for the bump Sensors. | Task 1  1.3 Write the code for the line Sensors |
| Week 2 | Task 1  1.2 Wiring Components of RSLK | Task 1  1.2 Write the code for the bump Sensors | Task 1  1.3Write the code for the line Sensors |
| Week 3 | Task 1  1.3 Assembling of RSLK Components | Task 1  1.3 Assembling of RSLK Components | Task 1  1.3 Write the code for the line Sensors |
| Week 4 | Task 1  1.3 Assembling of RSLK Components | Task 1  1.3 Assembling of RSLK Components | Task 1  1.1 Assembling of RSLK. Compiling and Uploading Codes |
| Week 5-6 | Task 3  3.1 Mounting mmWave to the RSLK  3.2 3D design and printing | Task 2  2.3 Implementing the mmWwave interface with the microcomputer | Task 2  2.1 Design Power supply for ODROID Using Buck converter  2.2 Setting up Ubuntu on ODROID XU4Q |
| Week 7- 11 | Task 4  Acquisition of the point cloud data from the radar with the microcomputer | Task 4  4.1 Launch the ti\_mmWave\_rospkg node to start the mmWave EVM | Task 5  5.1 4K FFT stitching of reflective object  5.2 Stitching 3D point cloud data |

Table 2. Task Assignment for Fall 2019

|  |  |  |  |
| --- | --- | --- | --- |
| Time | Jesudara | Aduralere | Favour |
| Week 11-13 | Task 4  Acquisition of the point cloud data from the radar with the microcomputer | Task 4  Acquisition of the point cloud data from the radar with the microcomputer | Task 5  5.2 Stitching the 3D point Cloud data. |
| Week 14-17 | Task 7  Using the Tachometer to calculate the speed | Task 6  Detecting and calculating the distance of an object behind the glass wall. | Task 6  Detecting and calculating the distance of an object behind the glass wall. |
| Week 18-20 | Task 8  System Finalization, testing and refinement. | Task 8  System Finalization, testing and refinement. | Task 8  System Finalization, testing and refinement. |

1. Implementation

## Implementation of Task 1.

### Implementation of Subtask 1.1

This involves the use of different tools, wires and supplies to assemble the TI-RSLK. After the RSLK parts were assembled, the final assembly looked similar to the picture in Fig. 7. The tasks for the TI-RSLK was divided into four main sections;

* Soldering
* Assembly
* Software
* Wiring

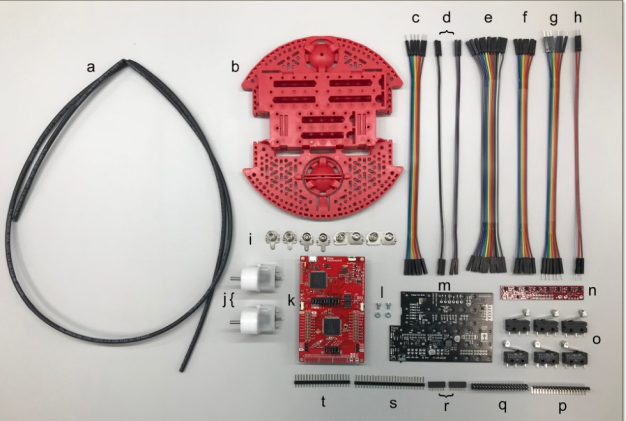


Fig. 5 Componets Used in Assembling the RSLK

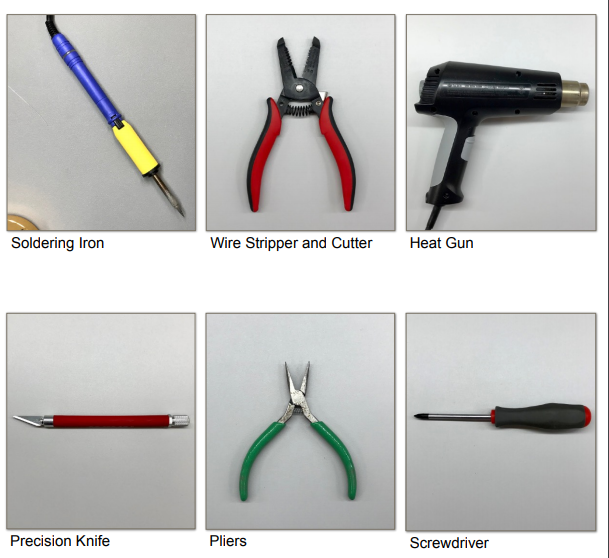


Fig. 6 Tools Used in Assembling the RSLK

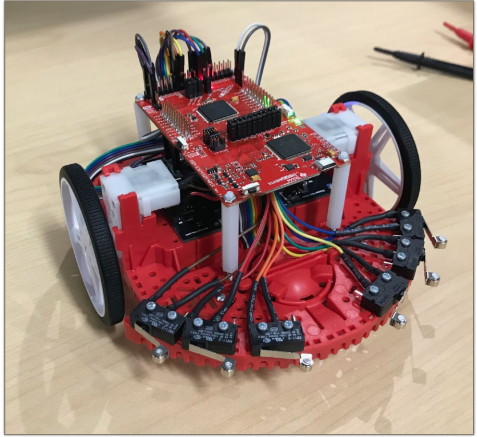


Fig. 7 Assembled TI RSLK

Table 3. Components of the Ti Rslk

|  |  |  |
| --- | --- | --- |
|  | Description |  |
| A | Heat Shrink Tube | 1 |
| B | Chassis | 1 |
| C | 6 Female to Male Wires | 1 |
| D | 2 Female to Male Wires | 4 |
| E | 11 Female to Female Wires | 1 |
| F | 6 Female to Female Wires | 1 |
| G | 6 Male to Male Wires | 1 |
| H | 2 Male to Male Wires | 1 |
| I | Battery Terminals | 1 |
| J | Motor | 2 |
| K | TI Launchpad kit | 1 |
| L | Motor Board Screw | 1 |
| M | Motor Board | 1 |
| N | Line Follower Sensor | 1 |
| O | Bump Switch | 6 |
| P | 90 Bent Header | 1 |
| Q | 2x20 Header | 1 |
| R | 1x6 Header | 1 |
| S | 1x25 Header | 1 |
| T | 1x20 Header | 1 |
| U | Motor Encoders | 2 |
| V | Bump switch wire spade connector | 12 |

* SOLDERING

A precision knife is used to cut the VPU-VREG, VCCMD-VREG, and SLP L-R traces. A deep solid cut is then made through the small line connection between the square pads. Two 1x6 female socket headers are connected to the yellow connections before connected to the 1x2 headers and then to the ELA and ELB connections. The ELA and ELB connections need to be bent to a 45-degree angle after soldering. A 1x2 header cut is made on the VPU connection. A 1x3 header is made and so is a 1x6 header cut to the left and right motor driver connections. 1x3 and 1x2 header cuts are made to the VREG terminals. The 1x8, 1x3 and 1x2 header cuts are soldered to the GND terminals. The four battery connections were soldered as part of the attachment to the chassis and the motor board was secured with two support screws. Pliers are then used to bend out the ELA and ELB & ERA and ERB connections so the connections can be attached by wires to the Launchpad.

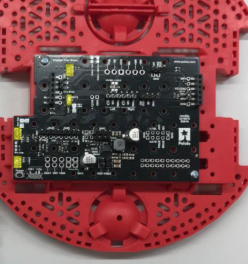


Fig. 8 Battery Connections Soldered as Part of the Attachment to the Chassis

LaunchPad: The 2x19 header is then soldered on the J5 pinout at the bottom of the LaunchPad with the long pins facing upward as seen in Fig. 9. The prepared encoders are then soldered on the motors. The bump switches; one male and one female wire each is soldered to each bump sensor. This is done for the 6 bump sensors as seen in Fig. 10

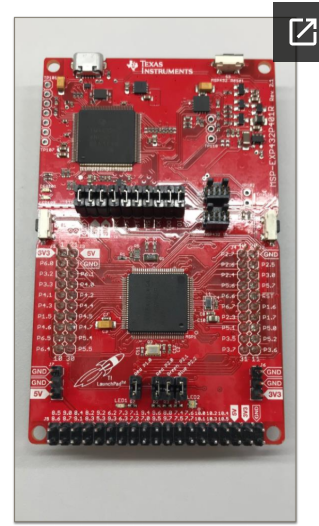


Fig. 9 LaunchPad Soldered Together with Long pins Facing Upward.

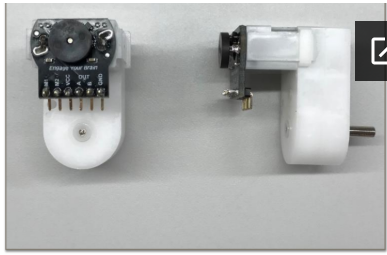


Fig. 10 Encoders Soldered to Motor Drivers.

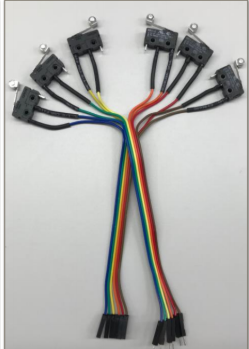


Fig. 11 Bump Switches

* ASSEMBLING

The motors are attached to the motor board on the chassis

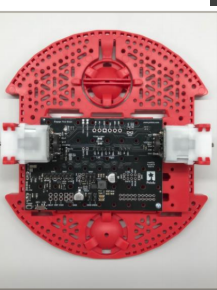


Fig. 12 Motor Board on Chassis

The ball caster is then attached to the chassis before the wheels are attached.



Fig. 13 Ball Caster

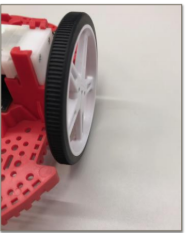


Fig. 14 Wheels of the Car

* WIRING

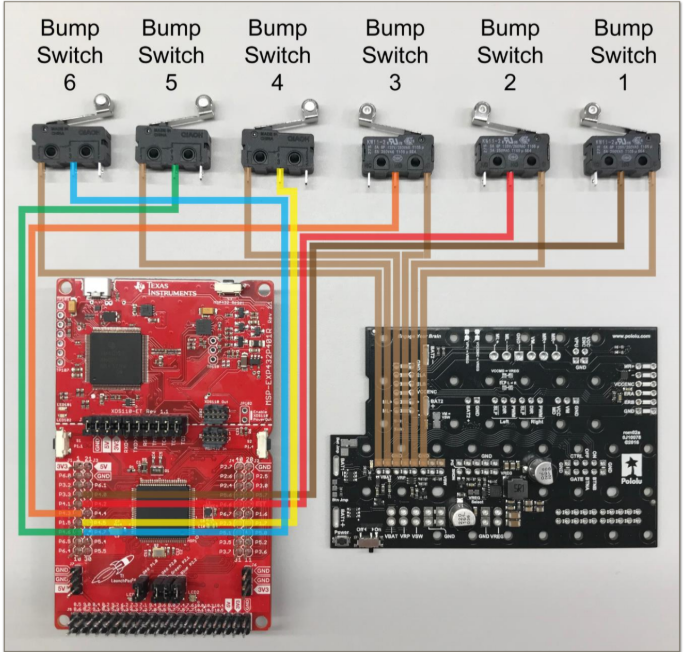


Fig. 15 Bump Switches Wiring

Table 4. Bump Switches Connection Guide

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Bump 1 | Bump 2 | Bump 3 | Bump 5 | Bump 6 |
| LaunchPad | P 4.0 | P.4.2 | P.4.5 | P.4.6 | P 4.7 |

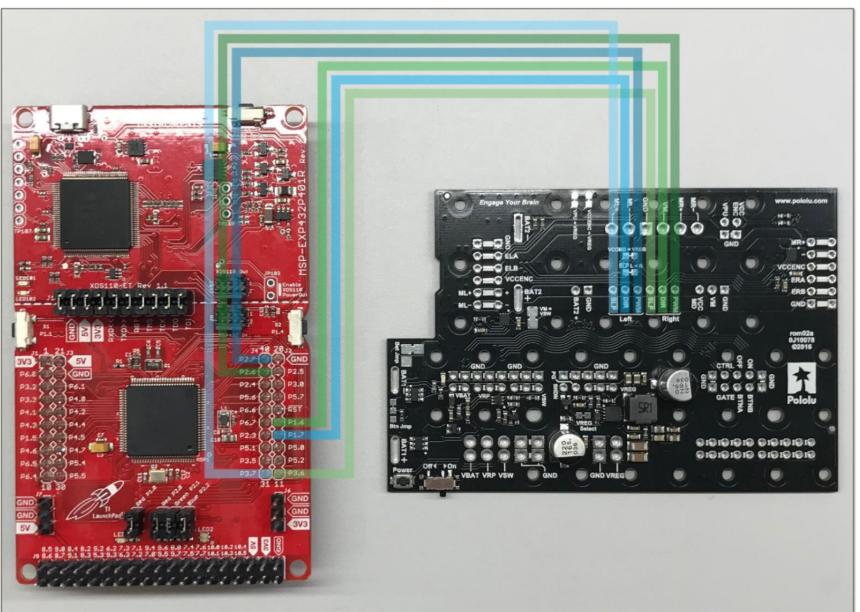


Fig. 16 Motor Board Wiring to LaunchPad

Table 5. Motorboard Wiring Guide

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Motor Board | Left SLP | Left DIR | Left PWM | Right SLP | Right DIR | Right PWM |
| Launchpad | P3.7 | P1.7 | P2.7 | P3.6 | P1.6 | P2.6 |

Table 6. Motorboard Wiring From Ground And Vreg

|  |  |  |
| --- | --- | --- |
| Motor Board | VREG | GND |
| Launchpad | 5V | GND |

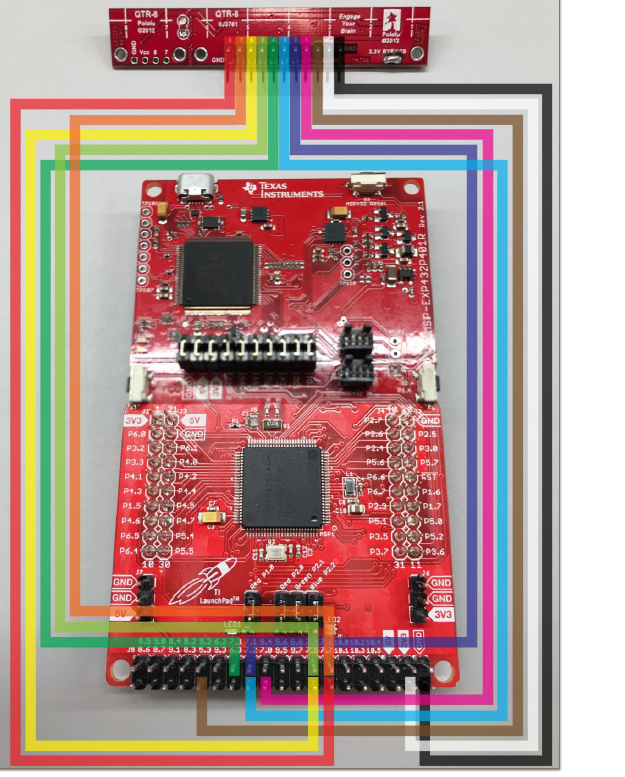


Fig. 17 Line Sensor Wiring Guide

Table 7. Line Sensor Wiring Guide

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Line Sensor | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | LED ON | VCC | GND |
| Launchpad | P7.7 | P7.6 | P7.5 | P7.4 | P7.3 | P7.2 | P7.1 | P7.0 | P5.3 | 3V3 | GND |

* SOFTWARE

The steps to get the software working were;

1. Install CCS (Code Composer Studios)
2. Import the TI Drivers
3. Download RSLK Energia Library

* TROUBLESHOOTING
* Soldering:

The first issue encountered when soldering was having to solder the connectors upside down as this affected the wiring because the female connector wire removed due to the fact that the pins have different diameter from top to bottom.

The solution to this issue was to de-solder the connectors and then re-solder.

Another issue that was encountered when soldering was with the encoders. The encoder connector pins are flexible which made to fragile. During the attachment of the motor board to the motor, an encoder pin broke causing the motor to stop working.

The solution to this issue was to de-solder the encoder and find a less fragile pin and connector to use.

The motor also had a similar issue, but the only solution was to buy another motor.

* Assembly:

The main issue when assembling the parts was from the chassis motor clips. The chassis motor clips from the TI RSLK kit didn’t sit well on the chassis.

The solution to this issue was to break part of the motor clips in order to fit the motors. The motors where then slid into the motor clips without over-stressing the clips.

* Software:

One of the challenges was installing CCStudio on the computer. Different errors occurred during the installations.

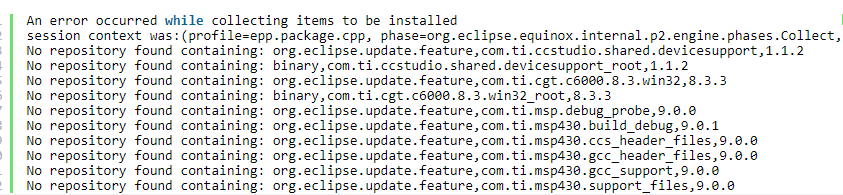


Fig. 18 Error Message Received

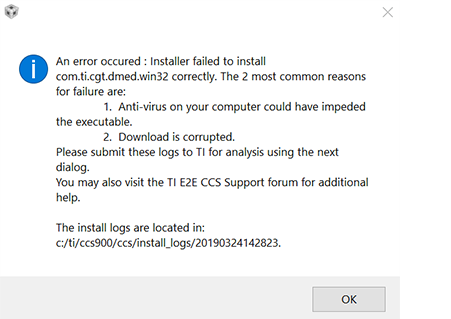


Fig. 19 Switching off the Antivirus

This problem was then solved by turning off the antivirus on the computer before the installation. After the RSLK parts were assembled, the final assembly looked similar to the picture below;

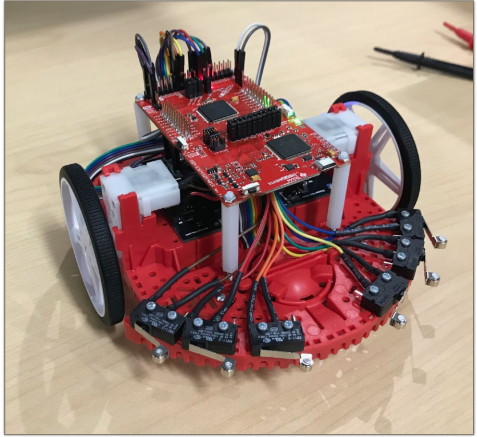


Fig. 20 Assembled TI RSLK

### Implementation of Subtask 1.2

For this task the assembling of the bump switches and the codes were written and compiled for the six bump switches. The following are gathered;

1. 6 bump switches
2. 12 Heat shrink tubes that are already cut
3. 6 female wires with one end cut and stripped
4. 6 male wires with one end cut and stripped

The first step in attaching the bumper switches to the TI RSLK is to slide the heat shrink tubes that are already cut on each female wire. Next, solder one female wire on the “1” or “C” connection on each bump switch. After soldering the bump switches on the female wires, heat shrink tubes are slid on 6 male wires. Lastly, solder one male wire on the “3” or “NO” connection on each bump switch (0). Color coding each switch will helps later when wiring. The codes for the bump switches can be found in the appendix section source code.

### Implementation of Subtask 1.3

For this task the code for the line tracker was written and compiled. The RSLK has two line sensors one on the left and one for the right. The RSLK has two motors.

For the two sensors (left right)

1 1 explains that the RSLK is on the line

1 0 explains that the RSLK is off to the right

0 1 explains that the RSLK is off to the left

0 0 explains that the RSLK is lost

If the car detects black lines when there are white line the RSLK would only go through the white lines. The line sensor follows black lines only. The motor driver runs the finite state machine every 50 microseconds. The flow chart in Fig. 21 summarizes the code. The source code for the line tracker can be found in Appendix section under source code 1.

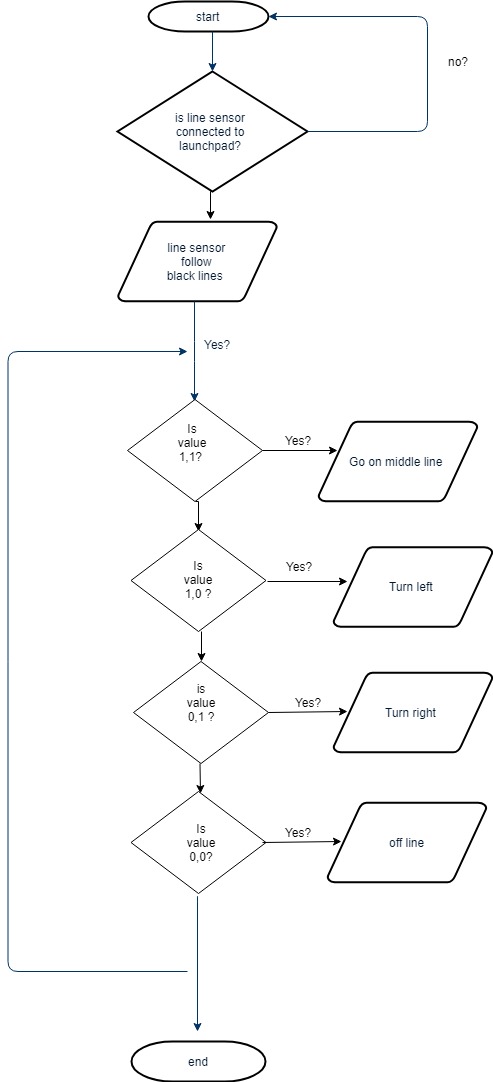


Fig. 21 Flow Chart for Line Sensor

## Implementation of Task 2.

### Implementation of Task 2 Subtask 1

For this task a LM2596 (buck converter) which steps down converter is also which DC-DC power converter which steps down voltage and steps up current to 2.5 Amp from the input voltage of 9 volts to the output load 5 volts. There is a potentiometer that regulates an output voltage to a voltage of 5.

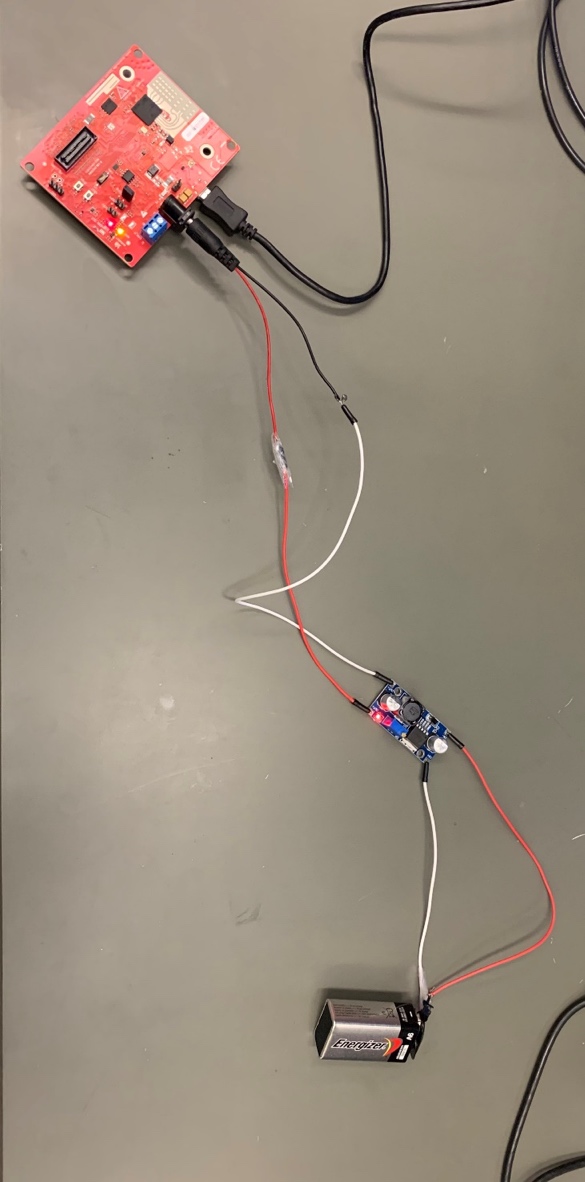


Fig. 22 5V Power Supply

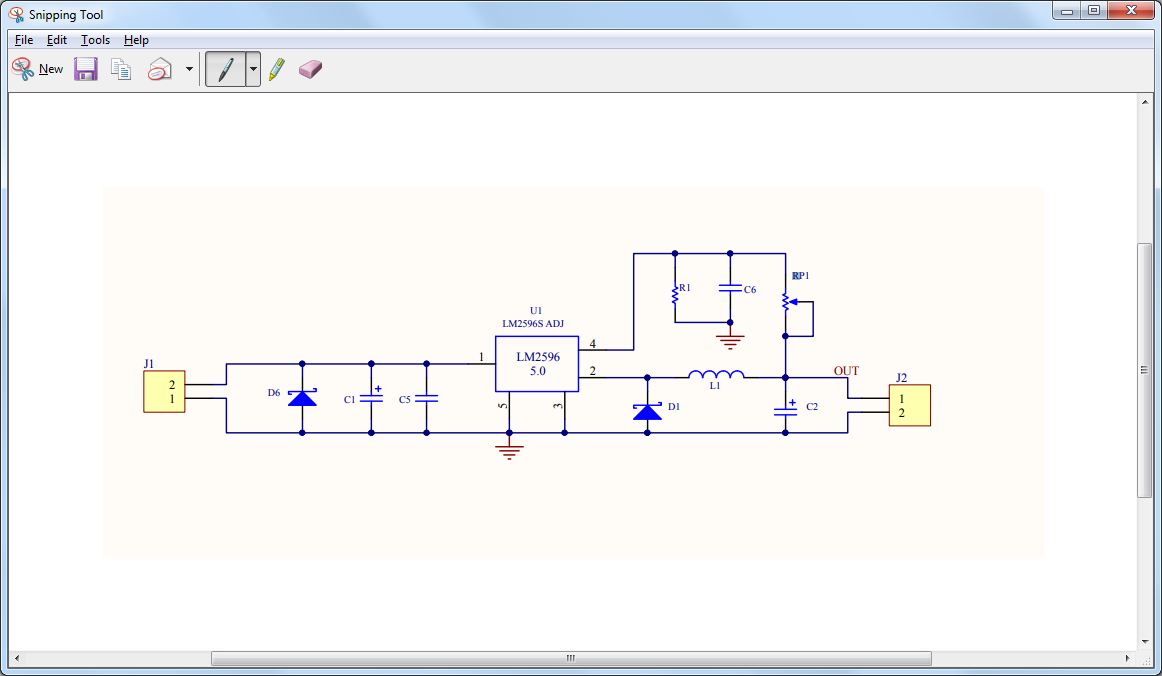


Fig. 23 Schematics of Buck Converter

### Implementation of Task 2 subtask 2

For this task Ubuntu 16.04 64bits was installed to enable the ROS library. After downloading Ubuntu, an Ubuntu drive was created to enable booting from a USB drive. Shrinking the windows 10 partition to make space was Ubuntu, boot into Ubuntu live environment and install Ubuntu.

## Implementation of Task 4.

The IWR1443 BoosterPack™ is an easy-to-use evaluation board for the single-chip IWR1443 mmWave sensing device from TI, with direct connectivity to the TI MCU LaunchPad™ ecosystem. The evaluation board contains everything needed to start developing on a low-power ARM®-R4F controller. The evaluation board includes onboard emulation for programming and debugging, onboard buttons, and LEDs, for quick integration of a simple user interface. The standard 20-pin BoosterPack headers make the evaluation board compatible with a wide variety of TI MCU LaunchPads and enables easy prototyping.

The Key features of the mmWave are;

1. XDS110-based JTAG emulation with serial port, for onboard QSPI flash programming
2. Backchannel UART through USB to PC, for logging purposes
3. Onboard antenna
4. One button and two LEDs, for user interaction 5-V power jack, to power the board
5. 40-pin Launchpad standard that leverages the Launchpad ecosystem.

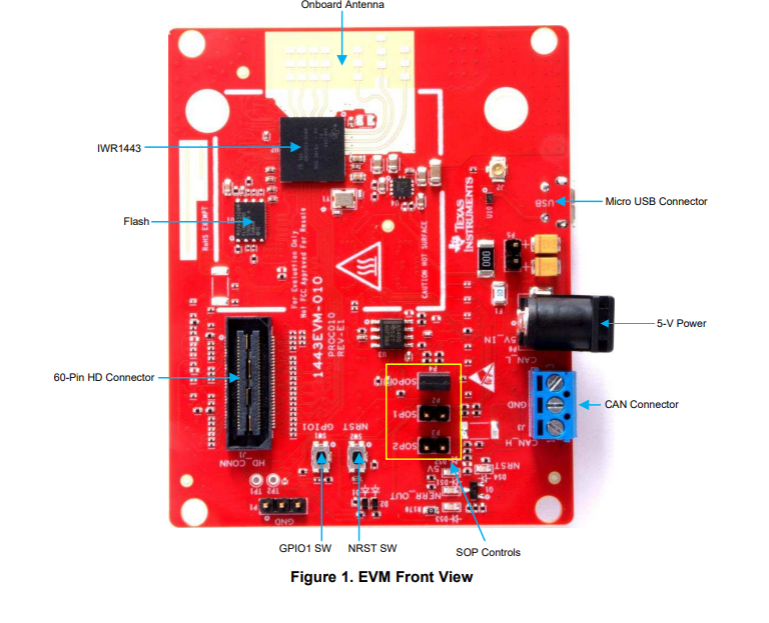


Fig. 24 Front view of the IWR1443

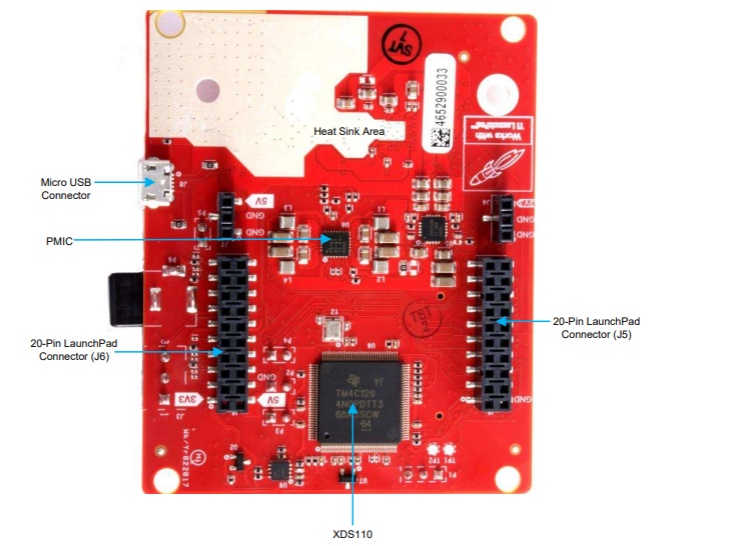


Fig. 25 Back view of the IWR 1443

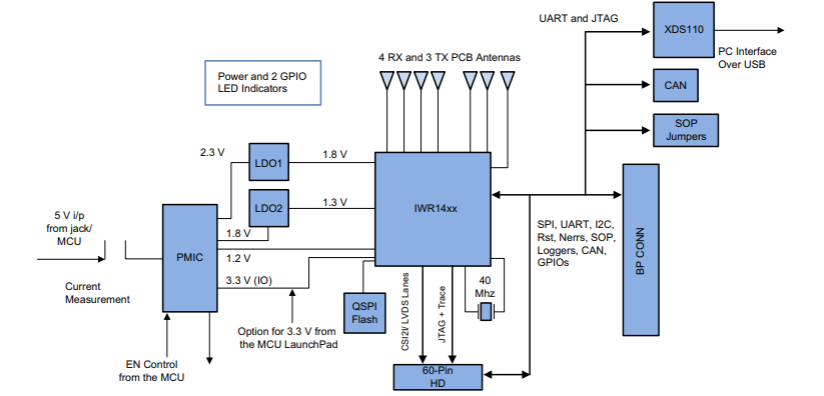


Fig. 26 Block diagram of the IWR1443

To generate a point cloud from mmWave with a Robotic Operating System, the following requirements are needed;

1. Host PC (laptop or desktop)
2. TI mmWave 1443 ES3.0 EVM
3. USB to microUSB cable
4. 5VDC power supply (center positive barrel jack, 2.5A)
5. Ubuntu 16.04 LTS operating system
6. Robot Operating System (ROS)
7. ti\_mmwave\_rospkg source code
8. mmWave SDK software
9. UniFlash

Why Robotic Operating System? The Robot Operating System (ROS) is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms. mmWave FMCW Radar chirp profile through command line interface (CLI) on a UART port, Do 1D, 2D, CFAR, Azimuth and Elevation processing and stream out intensity, relative radial velocity and three spatial coordinates (x,y,z) of the detected objects in real-time. Because the ROS framework and libraries support point cloud data, ROS is useful for developing and prototyping with the mmWave EVMs for robotics applications.

* Installing the ROS Library

The ROS kinetic Kame was selected among the two distribution that is commonly used now. The prerequisites for install ROS kinetic kame is install bootstrap dependencies, installing ROSDEP. The Installing Building the catkin Packages, Create a catkin workspace , Set up your keys, Install Debian package that contains(ROS, rqt, rviz, robot-generic libraries, 2D/3D simulators, navigation and 2D/3D perception and Environment setup under the ROS Workspace note install the rosbuild Not catkin.

1. Install the ROS ‘serial’ package:
2. Use GIT to download (clone) the ROS serial driver into your /src directory: $ cd /src $ git clone <https://github.com/wjwwood/serial.git>.
3. More documentation on this library can be found at: http://wjwwood.io/serial/ TI mmWave ROS Driver Setup Guide Version 1.5 6 3. Download the TI mmWave ROS Driver (ti\_mmwave\_rospkg) and copy it to your /src.
4. After opening the ti\_mmave\_rospkg\_.zip file, see the ti\_mmwave\_rospkg. Open a new file browser and navigate to your /src directory. Copy the ti\_mmwave\_rospkg folder to the /src directory.
5. Build your project: In the Terminal/shell, go to your directory and build the driver: $ cd $ catkin\_make.
6. Flash the mmWave using the UniFlash
7. Launch the TI mmWave ROS node and Rviz visualizer by running $ rosrun rviz rviz.

* Troubleshooting

The main issue faced was when flashing the IWR1443 device with UniFlash.

Solution:

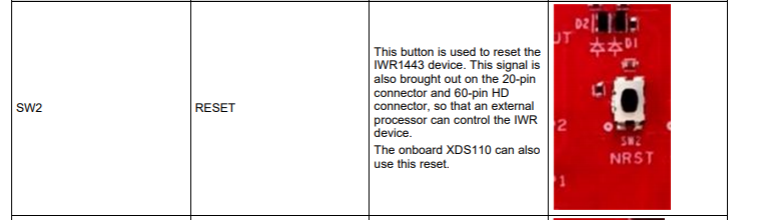


Fig. 27 Solution to the Issues Encountered

## Implementation of Task 5.

### Implementation of Task 5 Subtask 1

Fig. 28 below explains that for a stitching of a 3D point cloud data the first step is to register a collection of point cloud data that was collected from a robotic operating system visualizer. The robotic operating system visualizer stores files as an .obj file. To stitch the 3D point cloud data collected, the .obj file is converted to a .pcd file. After the file is converted from .obj to .pcd the file would be loaded unto the Pcd visualizer app to stitch the 3D point cloud data. The code for converting the .obj file to .pcd can be found in the appendix section.

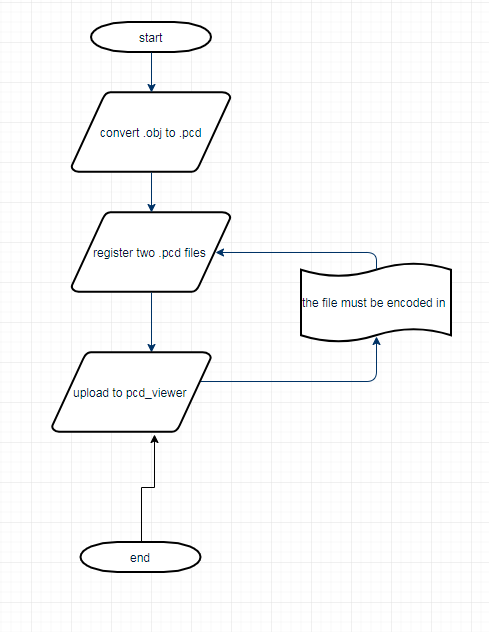


Fig. 28 3D Stitching of Point Cloud Data

### Implementation of Task 5 Subtask 2

The usefulness of this task is to get the range profile of the highly reflective materials in the range distance. A radar is for ranging and detection, pulses are sent out and received by an object and reflected on to the radar. The time of flight information is gathered and based on the principle of electromagnetic theory its wave transports its energy through a vacuum at speed and the distance. The radar captures data in time domain and process it using a FFT algorithm. The FFT algorithm is used to convert from time domain to frequency domain. The stitching process is done by taking 1000 sampling points of the information being processed 4 times. It uses an analog and digital converter ping pong (ADC\_Ping\_Pong). It uses a ping pong memory Mx. The ping pong memory M0 is loaded with information coming from the radar. The ping pong memory M2 processes previous 1000 sampled points from M0. The ping pong memory M1 is loaded with information coming from the radar then the sampled points from M1 is processed on the ping pong memory M3. The ADC output buffer is shared between local memory 1 (M0) and (M1). There is no need for EDMA; radar transferred to set up data between the two system. To process these information the first step in this process is to put the mmWave IWR1443 Boost EVM in flashing mode by connecting the jumpers on SOP0 and SOP2. Open the UniFlash tool downloaded on Ti.com/tool/uniflash to flash the prebuilt in binaries mmWave IWR1443 Boost EVM. Power cycle the mmWave and connect the mmWave with a USB to the computer being used, the mmWave has two virtual COM port. In the new configuration section locate and select the IWR1443 Boost EVM. Click start to proceed. In the program tab browse and locate the Radar SS and MSS images for the 4k FFT prebuilt in binaries as seen in the Fig. 29 below. In the settings and utilities port fill in the noted earlier as shown in the Fig. 30 below to flash the previous binaries. Return to the program tab and power cycle the device and click on load images of the prebuilt in binaries specified earlier. When the flash procedure is complete the UniFlash console should indicate [SUCCESS] Program Load completed successfully. Power off the board and remove the jumper from only SOP2, removing it puts the board in functional mode. Then run the lab on the demo visualizer, once the demo visualizer is loaded go to the options then serial port. In the serial port window enter the appropriate port numbers. Click on the configure and the demo visualizer will connect to the mmwave IWR1443. Click on the plot tab and load the code listed in the appendix section and radar would begin detecting reflective objects.

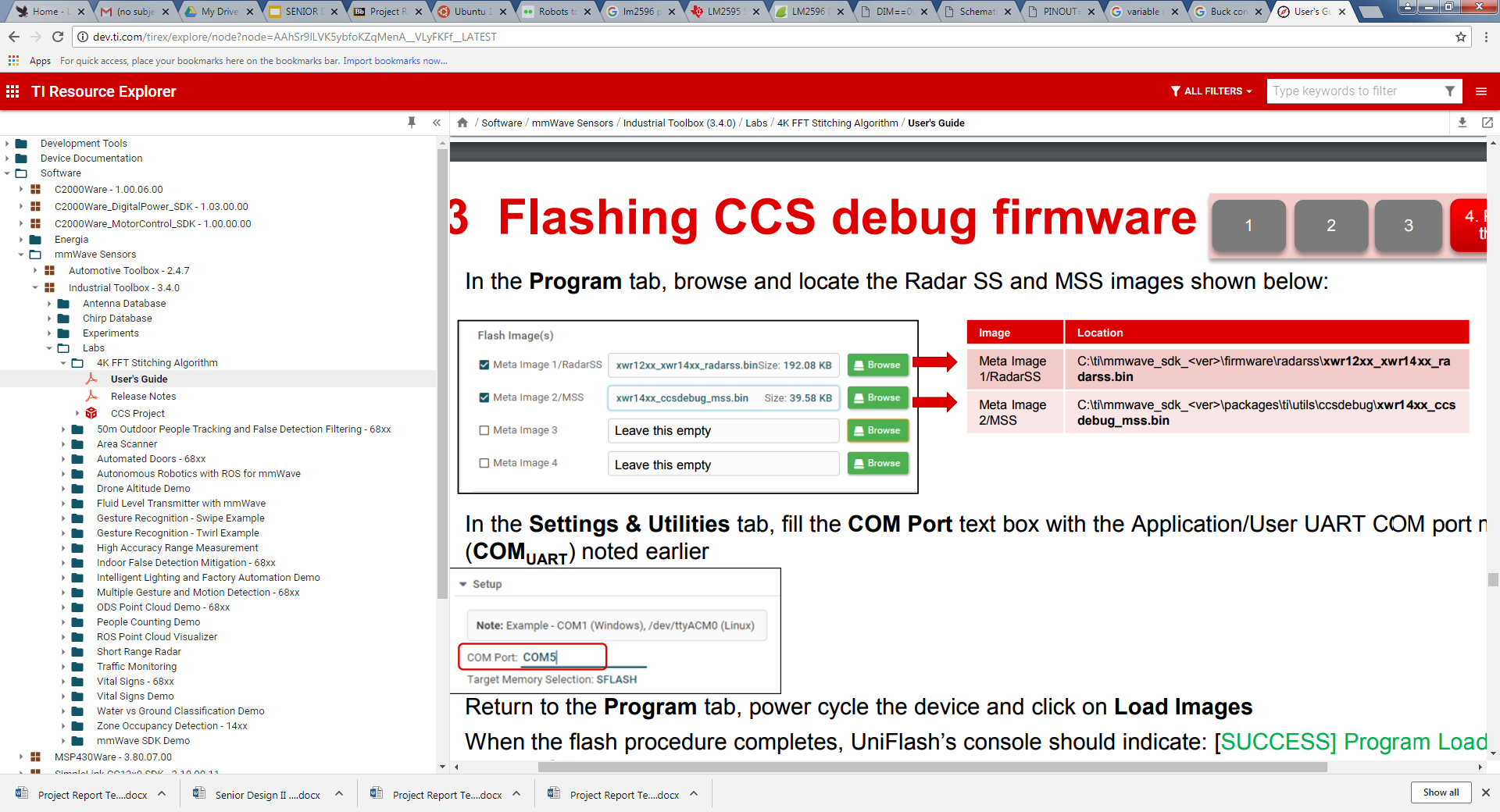


Fig. 29 Prebuilt in Binaries

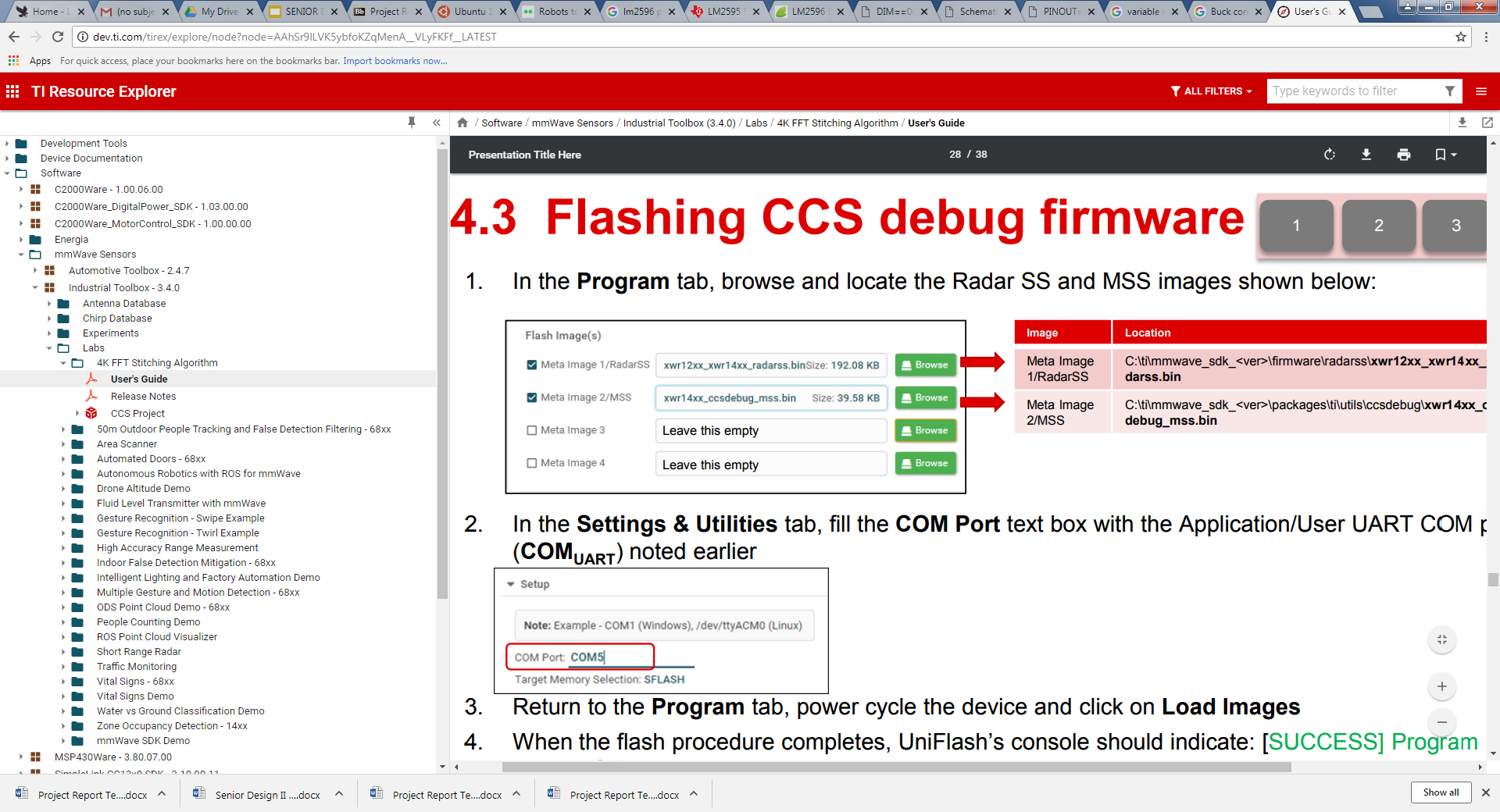


Fig. 30 COMuart Entry

* Trouble shooting

For the stitching to take place the mmWave required flashing the mmWave was not working as expected. The UniFlash showed a couple of error messages such as;

1. Aborting flash: loading images failed
2. Image loading failed aborting error of specified files in Fig. 31
3. Image loading failed: COM port not able to connect as seen in Fig. 32

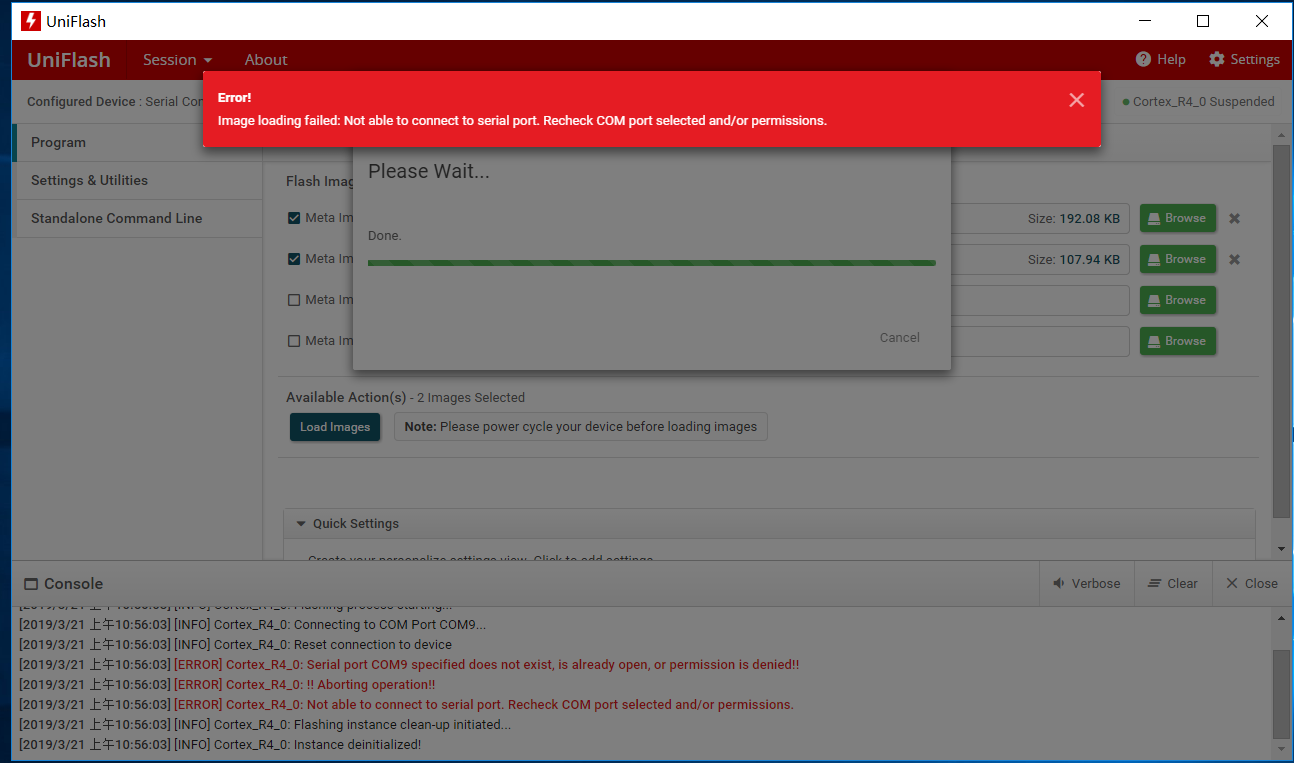


Fig. 31 Image Loading Error Message



Fig. 32 Error Message loading Image Failed on the UniFlash

To correct this error message;

1. Reinstallation of the UniFlash was done
2. Double checking the prebuilt in binaries selected
3. Ensuring the right COM Port was selected
4. Reinstalling the right mmWave SDK version was downloaded

After trying all of these, TI was contacted and concluded that the prebuilt in binaries had errors.

Acknowledgment

Dr. Alveron Walker

David Goslee

Appendix

1. Source Code.
2. Source Code of Line Sensor

struct State {

uint32\_t out; // 2-bit output

uint32\_t delay; // time to delay in 1ms

const struct State \*next[4]; // Next if input is 0-3

};

typedef const struct State State\_t;

#define Center &fsm[0]

#define Left &fsm[1]

#define Right &fsm[2]

StateType fsm[3]={

{0x03, 500, { Right, Left, Right, Middle }},

{0x02, 500, { Left, Middle, Right, Middle }},

{0x01, 500, { Right, Left, Middle, Middle }}

};

State\_t \*Spt;

uint32\_t Input;

uint32\_t Output;

int main(void){ uint32\_t heart=0;

Clock\_Init48MHz();

LaunchPad\_Init();

TExaS\_Init(LOGICANALYZER);

Spt = Middle;

while(1){

Output = Spt->out;

LaunchPad\_Output(Output);

TExaS\_Set(Input<<2|Output);

Clock\_Delay1ms(Spt->delay);

Input = LaunchPad\_Input();

Spt = Spt->next[Input];

heart = heart^1;

LaunchPad\_LED(heart);

}

}

1. Source Code of Bump Switches

void Poll(void){// 10 ms

if(P6->OUT&0x04){

SW1 = 1;

}

if(P6->OUT&0x08){

SW2 = 1;

}

}

int main(void){

Clock\_Init48MHz();

P6->DIR &= ~0x0C;

TimerA2\_Init(&Poll, 5000);

EnableInterrupts();

while(1){}

}

void VectorButtons\_Init(void){

P5->SEL0 &= ~0x08;

P5->SEL1 &= ~0x08;

P5->DIR &= ~0x08;

P5->IES &= ~0x08;

P5->IFG &= ~0x08;

P5->IE |= 0x08;

NVIC->IP[9]=(NVIC->IP[9]&0x00FFFFFF)|0x40000000;

NVIC->ISER[1] = 0x00000080;

P6->SEL0 &= ~0x08;

P6->SEL1 &= ~0x08;

P6->DIR &= ~0x08;

P6->IES &= ~0x08;

P6->IFG &= ~0x08;

P6->IE |= 0x08;

NVIC->IP[10]=(NVIC->IP[10]&0xFFFFFF00)|0x00000040;

NVIC->ISER[1] = 0x00000100;} // interrupt 40

void PORT5\_IRQHandler(void){

P5->IFG &= ~0x08;

SW1 = 1;

}

void PORT6\_IRQHandler(void){

P6->IFG &= ~0x08;

SW2 = 1;

}

1. Source Code of 4k FFT Stiching Algorithm

#include <stdint.h>

#include <stdlib.h>

#include <stddef.h>

#include <string.h>

#include <stdio.h>

/\* BIOS/XDC Include Files. \*/

#include <xdc/std.h>

#include <xdc/runtime/System.h>

#include <ti/drivers/hwa/hwa.h>

#include "config\_hwa\_util\_4k\_fft.h"

#include "data\_path\_4k\_fft.h"

#include "mmw.h"

extern mmwHwaBuf\_t gMmwHwaMemBuf[MMW\_HWA\_NUM\_MEM\_BUFS];

uint32\_t log2Approx(uint32\_t x)

{

uint32\_t idx, detectFlag = 0;

if ( x < 2)

{

return (0);

}

idx = 32U;

while((detectFlag==0U) || (idx==0U))

{

if(x & 0x80000000U)

{

detectFlag = 1;

}

x <<= 1U;

idx--;

}

if(x != 0)

{

idx = idx + 1;

}

return(idx);

}

void HWAutil\_configRangeFFT(HWA\_Handle handle,

uint32\_t paramSetStartIdx,

uint32\_t numAdcSamples,

uint32\_t numRangeBins,

uint8\_t numRxAnt,

uint32\_t windowOffsetBytes,

uint8\_t dmaTriggerSourcePing,

uint8\_t dmaTriggerSourcePong,

uint8\_t dmaDestChannelPing,

uint8\_t dmaDestChannelPong,

uint16\_t hwaMemAdcBufOffset,

uint8\_t hwaTriggerMode)

{

HWA\_InterruptConfig paramISRConfig;

int32\_t errCode = 0;

uint32\_t paramsetIdx = paramSetStartIdx;

uint32\_t pingParamSetIdx1 = 0;

uint32\_t pingParamSetIdx2 = 0;

HWA\_ParamConfig hwaParamCfg[HWAUTIL\_NUM\_PARAM\_SETS\_1D];

memset(hwaParamCfg,0,sizeof(hwaParamCfg));

hwaParamCfg[paramsetIdx].triggerMode = HWA\_TRIG\_MODE\_DMA;

hwaParamCfg[paramsetIdx].dmaTriggerSrc = dmaTriggerSourcePing;

hwaParamCfg[paramsetIdx].accelMode = HWA\_ACCELMODE\_NONE; //dummy

errCode = HWA\_configParamSet(handle,paramsetIdx,&hwaParamCfg[paramsetIdx],NULL);

/\* if (errCode != 0)

{

MmwDemo\_debugAssert (0);

return;

}\*/

paramsetIdx++;

pingParamSetIdx1 = paramsetIdx;

hwaParamCfg[paramsetIdx].triggerMode = hwaTriggerMode;

hwaParamCfg[paramsetIdx].accelMode = HWA\_ACCELMODE\_FFT;

hwaParamCfg[paramsetIdx].source.srcAddr = hwaMemAdcBufOffset;

hwaParamCfg[paramsetIdx].source.srcAcnt = numAdcSamples/4 - 1;

hwaParamCfg[paramsetIdx].source.srcAIdx = 4 \* numRxAnt \* sizeof(uint32\_t);

hwaParamCfg[paramsetIdx].source.srcBcnt = 3;

hwaParamCfg[paramsetIdx].source.srcBIdx = sizeof(uint32\_t);

hwaParamCfg[paramsetIdx].source.srcShift = 0;

hwaParamCfg[paramsetIdx].source.srcCircShiftWrap = 0;

hwaParamCfg[paramsetIdx].source.srcRealComplex = HWA\_SAMPLES\_FORMAT\_COMPLEX;

hwaParamCfg[paramsetIdx].source.srcWidth = HWA\_SAMPLES\_WIDTH\_16BIT;

hwaParamCfg[paramsetIdx].source.srcSign = HWA\_SAMPLES\_SIGNED;

hwaParamCfg[paramsetIdx].source.srcConjugate = 0;

hwaParamCfg[paramsetIdx].source.srcScale = 8;

hwaParamCfg[paramsetIdx].source.bpmEnable = 0;

hwaParamCfg[paramsetIdx].source.bpmPhase = 0;

hwaParamCfg[paramsetIdx].dest.dstAddr = ADDR\_TRANSLATE\_CPU\_TO\_HWA(MMW\_HWA\_1D\_OUT\_PING\_M0\_M2);

hwaParamCfg[paramsetIdx].dest.dstAcnt = numRangeBins/4-1;

hwaParamCfg[paramsetIdx].dest.dstAIdx = 4 \* numRxAnt \* sizeof(uint32\_t); //

hwaParamCfg[paramsetIdx].dest.dstBIdx = sizeof(uint32\_t);

hwaParamCfg[paramsetIdx].dest.dstRealComplex = HWA\_SAMPLES\_FORMAT\_COMPLEX;

hwaParamCfg[paramsetIdx].dest.dstWidth = HWA\_SAMPLES\_WIDTH\_16BIT;

hwaParamCfg[paramsetIdx].dest.dstSign = HWA\_SAMPLES\_SIGNED;

hwaParamCfg[paramsetIdx].dest.dstConjugate = 0;

hwaParamCfg[paramsetIdx].dest.dstScale = 0;

hwaParamCfg[paramsetIdx].dest.dstSkipInit = 0;

hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.fftEn = 1;

hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.fftSize = log2Approx(numRangeBins/4);

hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.butterflyScaling = 0x3;

hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.interfZeroOutEn = 0;

hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.windowEn = 1;

hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.windowStart = windowOffsetBytes; //start of window RAM

hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.winSymm = 0;

hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.winInterpolateMode = 1; //4Kstitching, enabling window interpolation

hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.magLogEn = HWA\_FFT\_MODE\_MAGNITUDE\_LOG2\_DISABLED;

hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.fftOutMode = HWA\_FFT\_MODE\_OUTPUT\_DEFAULT;

hwaParamCfg[paramsetIdx].complexMultiply.mode = HWA\_COMPLEX\_MULTIPLY\_MODE\_DISABLE;

errCode = HWA\_configParamSet(handle,paramsetIdx,&hwaParamCfg[paramsetIdx],NULL);

paramsetIdx++;

pingParamSetIdx2 = paramsetIdx;

hwaParamCfg[paramsetIdx].triggerMode = HWA\_TRIG\_MODE\_IMMEDIATE;

hwaParamCfg[paramsetIdx].accelMode = HWA\_ACCELMODE\_FFT; //do FFT

hwaParamCfg[paramsetIdx].source.srcAddr = ADDR\_TRANSLATE\_CPU\_TO\_HWA(MMW\_HWA\_1D\_OUT\_PING\_M0\_M2);

hwaParamCfg[paramsetIdx].source.srcAcnt = 3; //4 point FFT

hwaParamCfg[paramsetIdx].source.srcAIdx = 4;

hwaParamCfg[paramsetIdx].source.srcBcnt = numRangeBins/4 - 1;

hwaParamCfg[paramsetIdx].source.srcBIdx = 16;

hwaParamCfg[paramsetIdx].source.srcShift = 0;

hwaParamCfg[paramsetIdx].source.srcCircShiftWrap = 0;

hwaParamCfg[paramsetIdx].source.srcRealComplex = HWA\_SAMPLES\_FORMAT\_COMPLEX;

hwaParamCfg[paramsetIdx].source.srcWidth = HWA\_SAMPLES\_WIDTH\_16BIT;

hwaParamCfg[paramsetIdx].source.srcSign = HWA\_SAMPLES\_SIGNED;

hwaParamCfg[paramsetIdx].source.srcConjugate = 0;

hwaParamCfg[paramsetIdx].source.srcScale = 8;

hwaParamCfg[paramsetIdx].source.bpmEnable = 0;

hwaParamCfg[paramsetIdx].source.bpmPhase = 0;

hwaParamCfg[paramsetIdx].dest.dstAddr = ADDR\_TRANSLATE\_CPU\_TO\_HWA(MMW\_HWA\_1D\_OUT\_PING\_M2\_M3);

hwaParamCfg[paramsetIdx].dest.dstAcnt = 3;

hwaParamCfg[paramsetIdx].dest.dstAIdx = numRangeBins;

hwaParamCfg[paramsetIdx].dest.dstBIdx = 4;

hwaParamCfg[paramsetIdx].dest.dstRealComplex = HWA\_SAMPLES\_FORMAT\_COMPLEX;

hwaParamCfg[paramsetIdx].dest.dstWidth = HWA\_SAMPLES\_WIDTH\_16BIT;

hwaParamCfg[paramsetIdx].dest.dstSign = HWA\_SAMPLES\_SIGNED;

hwaParamCfg[paramsetIdx].dest.dstConjugate = 0;

hwaParamCfg[paramsetIdx].dest.dstScale = 0;

hwaParamCfg[paramsetIdx].dest.dstSkipInit = 0;

hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.fftEn = 1;

hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.fftSize = 2;

hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.butterflyScaling = 0x3;

hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.interfZeroOutEn = 0;

hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.windowEn = 0;

hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.windowStart = windowOffsetBytes;

hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.winSymm = 0; //non-symmetric - in demo do we make this symmetric

hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.winInterpolateMode = 0;

hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.magLogEn = HWA\_FFT\_MODE\_MAGNITUDE\_LOG2\_DISABLED;

hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.fftOutMode = HWA\_FFT\_MODE\_OUTPUT\_DEFAULT;

hwaParamCfg[paramsetIdx].complexMultiply.mode = HWA\_COMPLEX\_MULTIPLY\_MODE\_FFT\_STITCHING;

hwaParamCfg[paramsetIdx].complexMultiply.cmpMulArgs.twidIncrement = 1;

errCode = HWA\_configParamSet(handle,paramsetIdx,&hwaParamCfg[paramsetIdx],NULL);

if (errCode != 0)

{

System\_printf("Error: HWA\_configParamSet(%d) returned %d\n",errCode,paramsetIdx);

return;

}

paramISRConfig.interruptTypeFlag = HWA\_PARAMDONE\_INTERRUPT\_TYPE\_DMA;

paramISRConfig.dma.dstChannel = dmaDestChannelPing; //TODO sync this define EDMA channel to trigger to copy the data out

errCode = HWA\_enableParamSetInterrupt(handle,paramsetIdx,&paramISRConfig);

if (errCode != 0)

{

//System\_printf("Error: HWA\_enableParamSetInterrupt(PING DMA) returned %d\n",errCode);

MmwDemo\_debugAssert (0);

return;

}

paramsetIdx++;

hwaParamCfg[paramsetIdx].triggerMode = HWA\_TRIG\_MODE\_DMA;

hwaParamCfg[paramsetIdx].dmaTriggerSrc = dmaTriggerSourcePong;

hwaParamCfg[paramsetIdx].accelMode = HWA\_ACCELMODE\_NONE;

errCode = HWA\_configParamSet(handle,paramsetIdx,&hwaParamCfg[paramsetIdx],NULL);

if (errCode != 0)

{

MmwDemo\_debugAssert (0);

return;

}

paramsetIdx++;

hwaParamCfg[paramsetIdx] = hwaParamCfg[pingParamSetIdx1];

hwaParamCfg[paramsetIdx].dest.dstAddr = ADDR\_TRANSLATE\_CPU\_TO\_HWA(MMW\_HWA\_1D\_OUT\_PONG\_M1\_M3);

errCode = HWA\_configParamSet(handle,paramsetIdx,&hwaParamCfg[paramsetIdx],NULL);

if (errCode != 0)

{

System\_printf("Error: HWA\_configParamSet(%d) returned %d\n",errCode,paramsetIdx);

return;

}

paramsetIdx++;

hwaParamCfg[paramsetIdx] = hwaParamCfg[pingParamSetIdx2];

hwaParamCfg[paramsetIdx].source.srcAddr = ADDR\_TRANSLATE\_CPU\_TO\_HWA(MMW\_HWA\_1D\_OUT\_PONG\_M1\_M3);

hwaParamCfg[paramsetIdx].dest.dstAddr = ADDR\_TRANSLATE\_CPU\_TO\_HWA(MMW\_HWA\_1D\_OUT\_PONG\_M3\_M2); //hwaMemDestPongOffset;

errCode = HWA\_enableParamSetInterrupt(handle,paramsetIdx,&paramISRConfig);

if (errCode != 0)

{

MmwDemo\_debugAssert (0);

return;

}

/\* enable the DMA hookup to this paramset so that data gets copied out \*/

paramISRConfig.interruptTypeFlag = HWA\_PARAMDONE\_INTERRUPT\_TYPE\_DMA;

paramISRConfig.dma.dstChannel = dmaDestChannelPong;

errCode = HWA\_enableParamSetInterrupt(handle,paramsetIdx,&paramISRConfig);

if (errCode != 0)

{

System\_printf("Error: HWA\_enableParamSetInterrupt(PING DMA) returned %d\n",errCode);

return;

}

}

void HWAutil\_configDopplerFFT

(

HWA\_Handle handle,

uint32\_t paramSetStartIdx,

uint32\_t dopplerFftSize,

uint8\_t numVirtualAnt,

uint32\_t numRangeBinsPerIter,

uint32\_t windowOffsetBytes,

uint8\_t dmaTriggerSourcePing,

uint8\_t dmaTriggerSourcePong,

uint8\_t dmaDestChannelPing,

uint8\_t dmaDestChannelPong,

uint16\_t hwaMemSourcePingOffset,

uint16\_t hwaMemSourcePongOffset,

uint16\_t hwaMemDestPingOffset,

uint16\_t hwaMemDestPongOffset,

uint32\_t option

)

{

HWA\_ParamConfig hwaParamCfg[HWAUTIL\_NUM\_PARAM\_SETS\_2D];

uint32\_t paramsetIdx = 0;

int32\_t errCode = 0;

uint32\_t k;

HWA\_InterruptConfig paramISRConfig;

uint32\_t numDopplerBins = dopplerFftSize;

memset( (void\*) &hwaParamCfg[paramsetIdx], 0, sizeof(HWA\_ParamConfig));

//Doppler FFT

hwaParamCfg[paramsetIdx].triggerMode = HWA\_TRIG\_MODE\_DMA;

hwaParamCfg[paramsetIdx].dmaTriggerSrc = dmaTriggerSourcePing;

hwaParamCfg[paramsetIdx].accelMode = HWA\_ACCELMODE\_FFT;

hwaParamCfg[paramsetIdx].source.srcAddr = hwaMemSourcePingOffset; // address is relative to start of MEM0

hwaParamCfg[paramsetIdx].source.srcAcnt = dopplerFftSize - 1; //size in samples - 1

hwaParamCfg[paramsetIdx].source.srcAIdx = numVirtualAnt \* sizeof(uint32\_t); //

hwaParamCfg[paramsetIdx].source.srcBcnt = numVirtualAnt - 1;

hwaParamCfg[paramsetIdx].source.srcBIdx = sizeof(uint32\_t);

hwaParamCfg[paramsetIdx].source.srcShift = 0;

hwaParamCfg[paramsetIdx].source.srcCircShiftWrap = 0;

hwaParamCfg[paramsetIdx].source.srcRealComplex = 0;

hwaParamCfg[paramsetIdx].source.srcWidth = HWA\_SAMPLES\_WIDTH\_16BIT;

hwaParamCfg[paramsetIdx].source.srcSign = HWA\_SAMPLES\_SIGNED;

hwaParamCfg[paramsetIdx].source.srcConjugate = 0;

hwaParamCfg[paramsetIdx].source.srcScale = 0;

hwaParamCfg[paramsetIdx].source.bpmEnable = 0;

hwaParamCfg[paramsetIdx].source.bpmPhase = 0;

hwaParamCfg[paramsetIdx].dest.dstAddr = hwaMemDestPingOffset;

hwaParamCfg[paramsetIdx].dest.dstAcnt = dopplerFftSize - 1;

hwaParamCfg[paramsetIdx].dest.dstAIdx = numVirtualAnt \* sizeof(uint32\_t); // 16 bytes

hwaParamCfg[paramsetIdx].dest.dstBIdx = sizeof(uint32\_t); //should be dont care

hwaParamCfg[paramsetIdx].dest.dstRealComplex = 0; //same as input - complex

hwaParamCfg[paramsetIdx].dest.dstWidth = HWA\_SAMPLES\_WIDTH\_16BIT; //same as input - 16 bit

hwaParamCfg[paramsetIdx].dest.dstSign = HWA\_SAMPLES\_SIGNED; //same as input - signed

hwaParamCfg[paramsetIdx].dest.dstConjugate = 0; //no conjugate

hwaParamCfg[paramsetIdx].dest.dstScale = 8;

hwaParamCfg[paramsetIdx].dest.dstSkipInit = 0; // no skipping

hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.fftEn = 1;

hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.fftSize = log2Approx(dopplerFftSize);

hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.butterflyScaling = (dopplerFftSize - 1) >> 1;

hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.interfZeroOutEn = 0; //disabled

hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.windowEn = 0; //enabled

hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.windowStart = 0; //start of window RAM

hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.winSymm = 1; //non-symmetric - in demo do we make this symmetric

hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.winInterpolateMode = 0; //fftsize is less than 1k

if(option == (uint32\_t)DATA\_PATH\_CHAIN\_SEPARATE\_LOGMAG)

{

hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.magLogEn = HWA\_FFT\_MODE\_MAGNITUDE\_LOG2\_DISABLED;

}

else

{

hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.magLogEn = HWA\_FFT\_MODE\_MAGNITUDE\_LOG2\_ENABLED;

}

hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.fftOutMode = HWA\_FFT\_MODE\_OUTPUT\_DEFAULT;

hwaParamCfg[paramsetIdx].complexMultiply.mode = HWA\_COMPLEX\_MULTIPLY\_MODE\_DISABLE;

errCode = HWA\_configParamSet(handle, paramSetStartIdx+paramsetIdx, &hwaParamCfg[paramsetIdx], NULL);

if (errCode != 0)

{

//System\_printf("Error: HWA\_configParamSet(%d) returned %d\n", paramSetStartIdx+paramsetIdx, errCode);

MmwDemo\_debugAssert (0);

return;

}

for (paramsetIdx = 1; paramsetIdx < numRangeBinsPerIter; paramsetIdx++)

{

hwaParamCfg[paramsetIdx] = hwaParamCfg[paramsetIdx-1];

hwaParamCfg[paramsetIdx].source.srcAddr += sizeof(uint32\_t) \* numVirtualAnt \* dopplerFftSize;

hwaParamCfg[paramsetIdx].dest.dstAddr += sizeof(uint32\_t) \* numVirtualAnt \* dopplerFftSize;

hwaParamCfg[paramsetIdx].triggerMode = HWA\_TRIG\_MODE\_IMMEDIATE;

errCode = HWA\_configParamSet(handle, paramSetStartIdx+paramsetIdx, &hwaParamCfg[paramsetIdx], NULL);

if (errCode != 0)

{

paramSetStartIdx+paramsetIdx, errCode);

MmwDemo\_debugAssert (0);

return;

}

}

memset( (void\*) &hwaParamCfg[paramsetIdx], 0, sizeof(HWA\_ParamConfig));

hwaParamCfg[paramsetIdx].triggerMode = HWA\_TRIG\_MODE\_IMMEDIATE; //Immediate following first - in demo this should be HWA\_TRIG\_MODE\_DFE

hwaParamCfg[paramsetIdx].accelMode = HWA\_ACCELMODE\_FFT; //do FFT

hwaParamCfg[paramsetIdx].source.srcAddr = hwaMemDestPingOffset; // address is relative to start of MEM0

hwaParamCfg[paramsetIdx].source.srcAcnt = numDopplerBins \* numVirtualAnt \* numRangeBinsPerIter - 1; //size in samples - 1

hwaParamCfg[paramsetIdx].source.srcAIdx = sizeof(uint32\_t); //

hwaParamCfg[paramsetIdx].source.srcBcnt = 0; //no iterations here

hwaParamCfg[paramsetIdx].source.srcBIdx = 0; //should be dont care

hwaParamCfg[paramsetIdx].source.srcShift = 0; //no shift

hwaParamCfg[paramsetIdx].source.srcCircShiftWrap = 0; //no shift

hwaParamCfg[paramsetIdx].source.srcRealComplex = 0; //complex data

hwaParamCfg[paramsetIdx].source.srcWidth = HWA\_SAMPLES\_WIDTH\_16BIT; //16-bit

hwaParamCfg[paramsetIdx].source.srcSign = HWA\_SAMPLES\_SIGNED; //signed

hwaParamCfg[paramsetIdx].source.srcConjugate = 0; //no conjugate

hwaParamCfg[paramsetIdx].source.srcScale = 0;

hwaParamCfg[paramsetIdx].source.bpmEnable = 0; //bpm removal not enabled

hwaParamCfg[paramsetIdx].source.bpmPhase = 0; //dont care

hwaParamCfg[paramsetIdx].dest.dstAddr = hwaMemSourcePingOffset; // address is relative to start of MEM0

hwaParamCfg[paramsetIdx].dest.dstAcnt = numDopplerBins \* numVirtualAnt \* numRangeBinsPerIter - 1; //this is samples - 1

hwaParamCfg[paramsetIdx].dest.dstAIdx = sizeof(uint16\_t); // 2 bytes

hwaParamCfg[paramsetIdx].dest.dstBIdx = 0; //should be dont care

hwaParamCfg[paramsetIdx].dest.dstRealComplex = 1; //same as input - complex

hwaParamCfg[paramsetIdx].dest.dstWidth = HWA\_SAMPLES\_WIDTH\_16BIT; //same as input - 16 bit

hwaParamCfg[paramsetIdx].dest.dstSign = HWA\_SAMPLES\_UNSIGNED; //same as input - signed

hwaParamCfg[paramsetIdx].dest.dstConjugate = 0; //no conjugate

hwaParamCfg[paramsetIdx].dest.dstScale = 0;

hwaParamCfg[paramsetIdx].dest.dstSkipInit = 0; // no skipping

hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.fftEn = 0;

hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.fftSize = 1;//TODO remove later after driver corrects

if(option == (uint32\_t)DATA\_PATH\_CHAIN\_SEPARATE\_LOGMAG)

{

hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.magLogEn = HWA\_FFT\_MODE\_MAGNITUDE\_LOG2\_ENABLED;

}

else

{

hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.magLogEn = HWA\_FFT\_MODE\_MAGNITUDE\_LOG2\_DISABLED;

}

errCode = HWA\_configParamSet(handle, paramSetStartIdx+paramsetIdx, &hwaParamCfg[paramsetIdx], NULL);

if (errCode != 0)

{

//System\_printf("Error: Log2 HWA\_configParamSet(%d) returned %d \n", paramSetStartIdx+paramsetIdx, errCode);

//System\_printf("numDopplerBins %d numVirtualAnt %d numRangeBinsPerIter %d\n", numDopplerBins,numVirtualAnt, numRangeBinsPerIter);

MmwDemo\_debugAssert (0);

return;

}

paramsetIdx++;

memset( (void\*) &hwaParamCfg[paramsetIdx], 0, sizeof(HWA\_ParamConfig));

hwaParamCfg[paramsetIdx].triggerMode = HWA\_TRIG\_MODE\_IMMEDIATE; //Immediate

hwaParamCfg[paramsetIdx].accelMode = HWA\_ACCELMODE\_FFT; //do FFT

hwaParamCfg[paramsetIdx].source.srcAddr = hwaMemSourcePingOffset; // address is relative to start of MEM0

hwaParamCfg[paramsetIdx].source.srcAcnt = numVirtualAnt-1; //size in samples - 1

hwaParamCfg[paramsetIdx].source.srcAIdx = sizeof(uint16\_t); //

hwaParamCfg[paramsetIdx].source.srcBcnt = numDopplerBins \* numRangeBinsPerIter - 1; //no iterations here

hwaParamCfg[paramsetIdx].source.srcBIdx = numVirtualAnt \* sizeof(uint16\_t);

hwaParamCfg[paramsetIdx].source.srcShift = 0; //no shift

hwaParamCfg[paramsetIdx].source.srcCircShiftWrap = 0; //no shift

hwaParamCfg[paramsetIdx].source.srcRealComplex = 1; //real data

hwaParamCfg[paramsetIdx].source.srcWidth = HWA\_SAMPLES\_WIDTH\_16BIT; //16-bit

hwaParamCfg[paramsetIdx].source.srcSign = HWA\_SAMPLES\_UNSIGNED; //signed

hwaParamCfg[paramsetIdx].source.srcConjugate = 0; //no conjugate

hwaParamCfg[paramsetIdx].source.srcScale = 2;

hwaParamCfg[paramsetIdx].source.bpmEnable = 0; //bpm removal not enabled

hwaParamCfg[paramsetIdx].source.bpmPhase = 0; //dont care

hwaParamCfg[paramsetIdx].dest.dstAddr = (uint16\_t) (hwaMemDestPingOffset + numDopplerBins \*

numVirtualAnt \* numRangeBinsPerIter \* sizeof(uint32\_t)); // address is relative to start of MEMO

hwaParamCfg[paramsetIdx].dest.dstAcnt = 1 - 1; //this is samples - 1

hwaParamCfg[paramsetIdx].dest.dstAIdx = sizeof(uint16\_t); // 16 bytes

hwaParamCfg[paramsetIdx].dest.dstBIdx = sizeof(uint16\_t); //should be dont care

hwaParamCfg[paramsetIdx].dest.dstRealComplex = 1; //same as input - complex

hwaParamCfg[paramsetIdx].dest.dstWidth = HWA\_SAMPLES\_WIDTH\_16BIT; //same as input - 16 bit

hwaParamCfg[paramsetIdx].dest.dstSign = HWA\_SAMPLES\_UNSIGNED; //same as input - signed

hwaParamCfg[paramsetIdx].dest.dstConjugate = 0; //no conjugate

hwaParamCfg[paramsetIdx].dest.dstScale = 8;

hwaParamCfg[paramsetIdx].dest.dstSkipInit = 0; // no skipping

if(numVirtualAnt == 1)

{

/\*If number of virtual antennas is 1, do not use FFT to compute sum magnitude.

There is nothing to sum and azimuth can not be estimated.\*/

hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.fftEn = 0;

hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.fftSize = 1;

}

else

{

hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.fftEn = 1;

hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.fftSize = log2Approx(numVirtualAnt);

}

hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.butterflyScaling = 0x3FF; //LSB fftSize bits are relevant - revisit this for all FFT size and data size

hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.interfZeroOutEn = 0; //disabled

hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.windowEn = 0; //enabled

hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.windowStart = 0; //start of window RAM

hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.winSymm = 1; //non-symmetric - in demo do we make this symmetric

hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.winInterpolateMode = 0; //fftsize is less than 1K

hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.magLogEn = HWA\_FFT\_MODE\_MAGNITUDE\_LOG2\_DISABLED; //disabled

hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.fftOutMode = HWA\_FFT\_MODE\_OUTPUT\_DEFAULT; // output FFT samples

hwaParamCfg[paramsetIdx].complexMultiply.mode = HWA\_COMPLEX\_MULTIPLY\_MODE\_DISABLE;

errCode = HWA\_configParamSet(handle, paramSetStartIdx+paramsetIdx, &hwaParamCfg[paramsetIdx], NULL);

if (errCode != 0)

{

//System\_printf("Error: Sum of mag HWA\_configParamSet(%d) returned %d\n", paramSetStartIdx+paramsetIdx, errCode);

MmwDemo\_debugAssert (0);

return;

}

/\* Enable the DMA hookup to this paramset so that data gets copied out \*/

paramISRConfig.interruptTypeFlag = HWA\_PARAMDONE\_INTERRUPT\_TYPE\_DMA;

paramISRConfig.dma.dstChannel = dmaDestChannelPing; //EDMA channel to trigger to copy the data out

paramISRConfig.cpu.callbackArg = NULL;

errCode = HWA\_enableParamSetInterrupt(handle, paramSetStartIdx+paramsetIdx, &paramISRConfig);

if (errCode != 0)

{

//System\_printf("Error: HWA\_enableParamSetInterrupt(PING DMA) returned %d\n",errCode);

MmwDemo\_debugAssert (0);

return;

}

//programming HWACC for the pong buffer

for (k=0; k < numRangeBinsPerIter + 2; k++)

{

paramsetIdx = k + numRangeBinsPerIter + 2;

hwaParamCfg[paramsetIdx] = hwaParamCfg[k];

hwaParamCfg[paramsetIdx].source.srcAddr += (hwaMemSourcePongOffset - hwaMemSourcePingOffset);

hwaParamCfg[paramsetIdx].dest.dstAddr += (hwaMemDestPongOffset - hwaMemDestPingOffset);//#def??

if (k == 0)

{

hwaParamCfg[paramsetIdx].dmaTriggerSrc = dmaTriggerSourcePong;

}

errCode = HWA\_configParamSet(handle, paramSetStartIdx + paramsetIdx, &hwaParamCfg[paramsetIdx], NULL);

if (errCode != 0)

{

//System\_printf("Error: HWA\_configParamSet(%d) returned %d\n", paramSetStartIdx + paramsetIdx, errCode);

MmwDemo\_debugAssert (0);

return;

}

}

/\* Enable the DMA hookup to the last paramset \*/

paramISRConfig.interruptTypeFlag = HWA\_PARAMDONE\_INTERRUPT\_TYPE\_DMA;

paramISRConfig.dma.dstChannel = dmaDestChannelPong; //EDMA channel to trigger to copy the data out

paramISRConfig.cpu.callbackArg = NULL;

errCode = HWA\_enableParamSetInterrupt(handle, paramSetStartIdx + 2\*(numRangeBinsPerIter+2)-1, &paramISRConfig);

if (errCode != 0)

{

//System\_printf("Error: HWA\_enableParamSetInterrupt(PONG DMA) returned %d\n",errCode);

MmwDemo\_debugAssert (0);

return;

}

//errCode = HWA\_configRam(handle, HWA\_RAM\_TYPE\_WINDOW\_RAM, (uint8\_t \*)win\_data16, sizeof(win\_data16), windowOffsetBytes);

}

void HWAutil\_configDopplerFFTSingleRangeBin(HWA\_Handle handle,

uint32\_t paramSetStartIdx,

uint32\_t dopplerFftSize,

uint8\_t numVirtualAnt,

uint32\_t windowOffsetBytes,

uint8\_t dmaTriggerSource,

uint16\_t hwaMemAzimSource,

uint16\_t hwaMemAzimDest)

{

HWA\_ParamConfig hwaParamCfg;

int32\_t errCode = 0;

memset( (void\*) &hwaParamCfg, 0, sizeof(HWA\_ParamConfig));

hwaParamCfg.dmaTriggerSrc = dmaTriggerSource;

hwaParamCfg.accelMode = HWA\_ACCELMODE\_FFT; //do FFT

hwaParamCfg.source.srcAddr = hwaMemAzimSource;

hwaParamCfg.source.srcAcnt = dopplerFftSize - 1; //size in samples - 1

hwaParamCfg.source.srcAIdx = numVirtualAnt \* sizeof(uint32\_t); //

hwaParamCfg.source.srcBcnt = numVirtualAnt -1; //no iterations here

hwaParamCfg.source.srcBIdx = sizeof(uint32\_t);

hwaParamCfg.source.srcShift = 0; //no shift

hwaParamCfg.source.srcCircShiftWrap = 0; //no shift

hwaParamCfg.source.srcRealComplex = HWA\_SAMPLES\_FORMAT\_COMPLEX; //complex data

hwaParamCfg.source.srcWidth = HWA\_SAMPLES\_WIDTH\_16BIT; //16-bit

hwaParamCfg.source.srcSign = HWA\_SAMPLES\_SIGNED; //signed

hwaParamCfg.source.srcConjugate = 0; //no conjugate

hwaParamCfg.source.srcScale = 0;

hwaParamCfg.source.bpmEnable = 0; //bpm removal not enabled

hwaParamCfg.source.bpmPhase = 0; //dont care

hwaParamCfg.dest.dstAddr = hwaMemAzimDest; // address is relative to start of MEM0

hwaParamCfg.dest.dstAcnt = dopplerFftSize - 1; //this is samples - 1

hwaParamCfg.dest.dstAIdx = numVirtualAnt \* sizeof(uint32\_t); //

hwaParamCfg.dest.dstBIdx = sizeof(uint32\_t) ; //should be dont care

hwaParamCfg.dest.dstRealComplex = HWA\_SAMPLES\_FORMAT\_COMPLEX; //same as input - complex

hwaParamCfg.dest.dstWidth = HWA\_SAMPLES\_WIDTH\_16BIT; //same as input - 16 bit

hwaParamCfg.dest.dstSign = HWA\_SAMPLES\_SIGNED; //same as input - signed

hwaParamCfg.dest.dstConjugate = 0; //no conjugate

hwaParamCfg.dest.dstScale = 8 ;

hwaParamCfg.dest.dstSkipInit = 0; // no skipping

hwaParamCfg.accelModeArgs.fftMode.fftEn = 1;

hwaParamCfg.accelModeArgs.fftMode.fftSize = log2Approx(dopplerFftSize);

hwaParamCfg.accelModeArgs.fftMode.butterflyScaling = 0x3FF; //LSB fftSize bits are relevant - revisit this for all FFT size and data size

hwaParamCfg.accelModeArgs.fftMode.interfZeroOutEn = 0; //disabled

hwaParamCfg.accelModeArgs.fftMode.windowEn = 1; //enabled

hwaParamCfg.accelModeArgs.fftMode.windowStart = windowOffsetBytes; //start of window RAM

hwaParamCfg.accelModeArgs.fftMode.winSymm = 1; //non-symmetric - in demo do we make this symmetric

hwaParamCfg.accelModeArgs.fftMode.winInterpolateMode = 0; //fftsize is less than 1K

hwaParamCfg.accelModeArgs.fftMode.magLogEn = HWA\_FFT\_MODE\_MAGNITUDE\_LOG2\_DISABLED;//HWA\_FFT\_MODE\_MAGNITUDE\_LOG2\_DISABLED; //disabled

hwaParamCfg.accelModeArgs.fftMode.fftOutMode = HWA\_FFT\_MODE\_OUTPUT\_DEFAULT; // output FFT samples

hwaParamCfg.complexMultiply.mode = HWA\_COMPLEX\_MULTIPLY\_MODE\_DISABLE;

errCode = HWA\_configParamSet(handle, paramSetStartIdx, &hwaParamCfg, NULL);

if (errCode != 0)

{

System\_printf("Error: HWA\_configParamSet(%d) returned %d\n", paramSetStartIdx, errCode);

return;

}

}

void HWAutil\_configCFAR(HWA\_Handle handle,

uint32\_t paramSetStartIdx,

uint32\_t numRangeBins,

uint32\_t numDopplerBins,

uint32\_t winLen,

uint32\_t guardLen,

uint32\_t noiseDivRightShift,

uint8\_t peakGrouping,

uint8\_t cyclicMode,

uint8\_t nAvgMode,

uint16\_t detObjectListSize,

uint8\_t dmaTriggerSource,

uint8\_t dmaDestChannel,

uint16\_t hwaSourceBufOffset,

uint16\_t hwaDestBufOffset

)

{

HWA\_ParamConfig hwaParamCfg;

HWA\_InterruptConfig paramISRConfig;

int32\_t errCode = 0;

memset( (void\*) &hwaParamCfg, 0, sizeof(hwaParamCfg));

hwaParamCfg.triggerMode = HWA\_TRIG\_MODE\_DMA;

hwaParamCfg.dmaTriggerSrc = dmaTriggerSource;

hwaParamCfg.accelModeArgs.cfarMode.peakGroupEn = peakGrouping;

hwaParamCfg.accelMode = HWA\_ACCELMODE\_CFAR;

hwaParamCfg.accelModeArgs.cfarMode.operMode = HWA\_CFAR\_OPER\_MODE\_LOG\_INPUT\_REAL;

hwaParamCfg.source.srcAddr = hwaSourceBufOffset;

hwaParamCfg.source.srcAcnt = numRangeBins-1;

hwaParamCfg.source.srcRealComplex = HWA\_SAMPLES\_FORMAT\_REAL;

hwaParamCfg.source.srcAIdx = numDopplerBins\*2;

hwaParamCfg.source.srcBIdx = sizeof(uint16\_t);

hwaParamCfg.source.srcBcnt = numDopplerBins-1;

hwaParamCfg.source.srcScale = 8;

hwaParamCfg.dest.dstAddr = hwaDestBufOffset;

hwaParamCfg.dest.dstAcnt = detObjectListSize - 1;

hwaParamCfg.dest.dstRealComplex = HWA\_SAMPLES\_FORMAT\_COMPLEX;

hwaParamCfg.dest.dstWidth = HWA\_SAMPLES\_WIDTH\_32BIT;

hwaParamCfg.dest.dstAIdx = 8;

hwaParamCfg.dest.dstBIdx = 4096;

hwaParamCfg.dest.dstScale = 8;

hwaParamCfg.accelModeArgs.cfarMode.numGuardCells = guardLen;

hwaParamCfg.accelModeArgs.cfarMode.nAvgDivFactor = noiseDivRightShift;

hwaParamCfg.accelModeArgs.cfarMode.cyclicModeEn = cyclicMode;

hwaParamCfg.accelModeArgs.cfarMode.nAvgMode = nAvgMode;

hwaParamCfg.accelModeArgs.cfarMode.numNoiseSamplesRight = winLen >> 1;

hwaParamCfg.accelModeArgs.cfarMode.numNoiseSamplesLeft = winLen >> 1;

hwaParamCfg.accelModeArgs.cfarMode.outputMode = HWA\_CFAR\_OUTPUT\_MODE\_I\_PEAK\_IDX\_Q\_NEIGHBOR\_NOISE\_VAL;

errCode = HWA\_configParamSet(handle, paramSetStartIdx, &hwaParamCfg, NULL);

if (errCode != 0)

{

//System\_printf("Error: HWA\_configParamSet(%d) returned %d\n", paramSetStartIdx, errCode);

MmwDemo\_debugAssert (0);

return;

}

paramISRConfig.interruptTypeFlag = HWA\_PARAMDONE\_INTERRUPT\_TYPE\_DMA;

paramISRConfig.dma.dstChannel = dmaDestChannel; //EDMA channel to trigger to copy the data out

paramISRConfig.cpu.callbackArg = NULL;

errCode = HWA\_enableParamSetInterrupt(handle, paramSetStartIdx, &paramISRConfig);

if (errCode != 0)

{

//System\_printf("Error: HWA\_enableParamSetInterrupt(PONG DMA) returned %d\n",errCode);

MmwDemo\_debugAssert (0);

return;

}

}

void HWAutil\_configAngleEstAzimuth(HWA\_Handle handle,

uint32\_t paramSetStartIdx,

uint32\_t numVirtualAnt,

uint32\_t fftOutSize,

uint32\_t numIter,

uint16\_t hwaSourceBufOffset,

uint16\_t hwaDestBufOffset)

{

HWA\_ParamConfig hwaParamCfg;

int32\_t errCode = 0;

memset( (void\*) &hwaParamCfg, 0, sizeof(hwaParamCfg));

hwaParamCfg.triggerMode = HWA\_TRIG\_MODE\_IMMEDIATE;

hwaParamCfg.source.srcAddr = hwaSourceBufOffset;

hwaParamCfg.source.srcAcnt = numVirtualAnt-1;

hwaParamCfg.source.srcAIdx = sizeof(uint32\_t);

hwaParamCfg.source.srcBIdx = numVirtualAnt \* sizeof(uint32\_t);

hwaParamCfg.source.srcBcnt = numIter-1;

hwaParamCfg.source.srcSign = HWA\_SAMPLES\_SIGNED;

hwaParamCfg.source.srcScale = 8;

hwaParamCfg.complexMultiply.mode = HWA\_COMPLEX\_MULTIPLY\_MODE\_DISABLE;

hwaParamCfg.accelModeArgs.fftMode.fftEn = 1;

hwaParamCfg.accelModeArgs.fftMode.magLogEn = HWA\_FFT\_MODE\_MAGNITUDE\_ONLY\_ENABLED;

hwaParamCfg.accelModeArgs.fftMode.fftSize = log2Approx(fftOutSize);//assumes power of 2;

hwaParamCfg.accelModeArgs.fftMode.windowEn = 0;

hwaParamCfg.accelModeArgs.fftMode.winSymm = 1;

hwaParamCfg.accelModeArgs.fftMode.windowStart = 512; //do not care

hwaParamCfg.accelModeArgs.fftMode.butterflyScaling = 0; //no scaling

hwaParamCfg.dest.dstAddr = hwaDestBufOffset;

hwaParamCfg.dest.dstAcnt = fftOutSize-1;

hwaParamCfg.dest.dstAIdx = sizeof(uint16\_t);//abs

hwaParamCfg.dest.dstBIdx = fftOutSize \* sizeof(uint16\_t);

hwaParamCfg.dest.dstSign = HWA\_SAMPLES\_UNSIGNED;

hwaParamCfg.dest.dstScale = 3;

hwaParamCfg.dest.dstRealComplex = HWA\_SAMPLES\_FORMAT\_REAL;

errCode = HWA\_configParamSet(handle, paramSetStartIdx, &hwaParamCfg, NULL);

if (errCode != 0)

{

MmwDemo\_debugAssert (0);

return;

}

}

void HWAutil\_configAngleEstAzimuthElevation(HWA\_Handle handle,

uint32\_t paramSetStartIdx,

uint32\_t numVirtualAntAzim,

uint32\_t numVirtualAntElev,

uint32\_t fftOutSize,

uint32\_t numIter,

uint16\_t hwaSourceAzimBufOffset,

uint32\_t hwaSourceElevBufOffset,

uint16\_t hwaDestAzimAbsBufOffset,

uint16\_t hwaDestAzimCplxBufOffset,

uint16\_t hwaDestElevCplxBufOffset)

{

HWA\_ParamConfig hwaParamCfg;

int32\_t errCode = 0;

memset( (void\*) &hwaParamCfg, 0, sizeof(hwaParamCfg));

hwaParamCfg.triggerMode = HWA\_TRIG\_MODE\_IMMEDIATE;

hwaParamCfg.source.srcAddr = (uint16\_t) hwaSourceElevBufOffset;

hwaParamCfg.source.srcAcnt = numVirtualAntElev-1;

hwaParamCfg.source.srcAIdx = sizeof(uint32\_t);

hwaParamCfg.source.srcBIdx = numVirtualAntElev \* sizeof(uint32\_t);

hwaParamCfg.source.srcBcnt = numIter-1;

hwaParamCfg.source.srcSign = HWA\_SAMPLES\_SIGNED;

hwaParamCfg.source.srcScale = 8;

hwaParamCfg.complexMultiply.mode = HWA\_COMPLEX\_MULTIPLY\_MODE\_DISABLE;

hwaParamCfg.accelModeArgs.fftMode.fftEn = 1;

hwaParamCfg.accelModeArgs.fftMode.fftSize = log2Approx(fftOutSize);

hwaParamCfg.accelModeArgs.fftMode.windowEn = 0;

hwaParamCfg.accelModeArgs.fftMode.butterflyScaling = 0;

hwaParamCfg.dest.dstAddr = (uint16\_t) hwaDestElevCplxBufOffset;

hwaParamCfg.dest.dstAcnt = fftOutSize-1;

hwaParamCfg.dest.dstAIdx = sizeof(uint32\_t);

hwaParamCfg.dest.dstBIdx = fftOutSize \* sizeof(uint32\_t);

hwaParamCfg.dest.dstSign = HWA\_SAMPLES\_SIGNED;

hwaParamCfg.dest.dstScale = 3;

errCode = HWA\_configParamSet(handle, paramSetStartIdx, &hwaParamCfg, NULL);

if (errCode != 0)

{

MmwDemo\_debugAssert (0);

return;

}

memset( (void\*) &hwaParamCfg, 0, sizeof(hwaParamCfg));

hwaParamCfg.triggerMode = HWA\_TRIG\_MODE\_IMMEDIATE;

hwaParamCfg.source.srcAddr = (uint16\_t) hwaSourceAzimBufOffset;

hwaParamCfg.source.srcAcnt = numVirtualAntAzim-1;

hwaParamCfg.source.srcAIdx = sizeof(uint32\_t);

hwaParamCfg.source.srcBIdx = numVirtualAntAzim \* sizeof(uint32\_t);

hwaParamCfg.source.srcBcnt = numIter-1;

hwaParamCfg.source.srcSign = HWA\_SAMPLES\_SIGNED;

hwaParamCfg.source.srcScale = 8;

hwaParamCfg.complexMultiply.mode = HWA\_COMPLEX\_MULTIPLY\_MODE\_DISABLE;

hwaParamCfg.accelModeArgs.fftMode.fftEn = 1;

hwaParamCfg.accelModeArgs.fftMode.magLogEn = HWA\_FFT\_MODE\_MAGNITUDE\_ONLY\_ENABLED;

hwaParamCfg.accelModeArgs.fftMode.fftSize = log2Approx(fftOutSize);

hwaParamCfg.accelModeArgs.fftMode.windowEn = 0;

hwaParamCfg.accelModeArgs.fftMode.butterflyScaling = 0;

hwaParamCfg.dest.dstAddr = (uint16\_t) hwaDestAzimAbsBufOffset;

hwaParamCfg.dest.dstAcnt = fftOutSize-1;

hwaParamCfg.dest.dstAIdx = sizeof(uint16\_t);//abs

hwaParamCfg.dest.dstBIdx = fftOutSize \* sizeof(uint16\_t);

hwaParamCfg.dest.dstSign = HWA\_SAMPLES\_UNSIGNED;

hwaParamCfg.dest.dstScale = 3;

hwaParamCfg.dest.dstRealComplex = HWA\_SAMPLES\_FORMAT\_REAL;

errCode = HWA\_configParamSet(handle, paramSetStartIdx + 1, &hwaParamCfg, NULL);

if (errCode != 0)

{

MmwDemo\_debugAssert (0);

return;

}

memset( (void\*) &hwaParamCfg, 0, sizeof(hwaParamCfg));

hwaParamCfg.triggerMode = HWA\_TRIG\_MODE\_IMMEDIATE;

hwaParamCfg.source.srcAddr = (uint16\_t) hwaSourceAzimBufOffset;

hwaParamCfg.source.srcAcnt = numVirtualAntAzim-1;

hwaParamCfg.source.srcAIdx = sizeof(uint32\_t);

hwaParamCfg.source.srcBIdx = numVirtualAntAzim \* sizeof(uint32\_t);

hwaParamCfg.source.srcBcnt = numIter-1;

hwaParamCfg.source.srcSign = HWA\_SAMPLES\_SIGNED;

hwaParamCfg.source.srcScale = 8;

hwaParamCfg.complexMultiply.mode = HWA\_COMPLEX\_MULTIPLY\_MODE\_DISABLE;

hwaParamCfg.accelModeArgs.fftMode.fftEn = 1;

hwaParamCfg.accelModeArgs.fftMode.fftSize = log2Approx(fftOutSize);

hwaParamCfg.accelModeArgs.fftMode.windowEn = 0;

hwaParamCfg.accelModeArgs.fftMode.butterflyScaling = 0;

hwaParamCfg.dest.dstAddr = (uint16\_t) hwaDestAzimCplxBufOffset;

hwaParamCfg.dest.dstAcnt = fftOutSize-1;

hwaParamCfg.dest.dstAIdx = sizeof(uint32\_t);

hwaParamCfg.dest.dstBIdx = fftOutSize \* sizeof(uint32\_t);

hwaParamCfg.dest.dstSign = HWA\_SAMPLES\_SIGNED;

hwaParamCfg.dest.dstScale = 3;

errCode = HWA\_configParamSet(handle, paramSetStartIdx + 2, &hwaParamCfg, NULL);

if (errCode != 0)

{

MmwDemo\_debugAssert (0);

return;

}

}

1. Source Code of .Obj to .Pcd Converter

$ ./obj2pcd

obj2pcd [options] [.pcd or .obj file]

Options:

-h Print this help

Example for .obj to .pcd:

$ ./obj2pcd file.obj

Converting file.obj -> file.pcd

Done.

$ ./obj2pcd file.obj output.pcd

Converting file.obj -> output.pcd

Done.

Example for .pcd to .obj:

$ ./obj2pcd file.pcd

Converting file.pcd -> file.obj

Done.

$ ./obj2pcd file.pcd output.obj

Converting file.pcd -> output.obj

1. Source Code for ROS Installation

sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb\_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'

sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-key 421C365BD9FF1F717815A3895523BAEEB01FA116

sudo apt-get update

sudo apt-get install ros-kinetic-desktop-full

sudo rosdep init

rosdep update

echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc

source ~/.bashrc

source /opt/ros/kinetic/setup.bash

echo "source /opt/ros/kinetic/setup.zsh" >> ~/.zshrc

source ~/.zshrc

sudo apt install python-rosinstall python-rosinstall-generator python-wstool build-essential

$ printenv | grep ROS

$ source /opt/ros/<distro>/setup.bash

$ source /opt/ros/kinetic/setup.bash

rosws init ~/fuerte\_workspace /opt/ros/fuerte

sudo apt-get install python-rosinstall

mkdir ~/fuerte\_workspace/sandbox

$ printenv | grep ROS

$ source /opt/ros/<distro>/setup.bash

$ source /opt/ros/kinetic/setup.bash

rosws init ~/fuerte\_workspace /opt/ros/fuerte

sudo apt-get install python-rosinstall

mkdir ~/fuerte\_workspace/sandbox

rosws set ~/fuerte\_workspace/sandbox

$ echo $ROS\_PACKAGE\_PATH

/home/jesudara/fuerte\_workspace/sandbox:/opt/ros/fuerte/share:/opt/ros/fuerte/stacks

REFERENCES

[1] Zeldovich, Lina. “These Rescue Robots Can Tackle Almost Any Disaster, Part 2.” *ASME.org*, Feb. 2019, www.asme.org/engineering-topics/articles/robotics/rescue-robots-can-tackle-almost-any-disaster-part.

[2] Zeldovich, Lina. “The 7 Coolest Rescue Robots Tackle Almost Any Disaster, Part 1.” *ASME.org*, Feb. 2019, www.asme.org/engineering-topics/articles/robotics/7-coolest-rescue-robots-tackle-almost-any.

[3] Underwood, Kimberly. “DHS, NIST Standards for Rescue Robots Gain Momentum.” *SIGNAL Magazine*, 3 Oct. 2018, www.afcea.org/content/dhs-nist-standards-rescue-robots-gain-momentum.

[4] Faggella, Daniel. “11 Robotic Applications for Search and Rescue.” *HuffPost*, HuffPost, 23 Nov. 2017, www.huffpost.com/entry/11-robotic-applications-for-search-and-rescue\_b\_5a173c9ae4b0bf1467a845c4.

[5] http://www.ti.com/product/IWR1443