Application Report
SWRA581B – December 2017 – Revised October 2018

Mmwave Radar Device ADC Raw Data Capture

ABSTRACT

This application report demonstrates how to interpret raw analog-to-digital converter (ADC) data that is captured using the Capture Demo or Mmwave Studio. The format of captured raw ADC data is discussed for different hardware setup respectively. Matlab snippet is provided for engineers who need to develop custom code for data processing.

Contents

List of Figures

Trademarks

1 Introduction

The raw data of the mmwave radar (xWRxxxx) can be captured using the Capture Demo provided in the mmWave SDK, or using Mmwave Studio provided in Mmwave Studio GUI tools. The primary topics covered include how to save raw data from the Capture Demo using Code Composer Studio™ (CCS), the raw data format of data captured using the Capture Demo, the raw data format of data captured using Mmwave Studio, and how to interpret the data in MATLAB. This document assumes that the user knows how to run the Capture Demo and Mmwave Studio. For instructions on how to run the Capture Demo, consult the *mmWave SDK Demo Setup Guide* found in the TI Resource Explorer mmWave Training Demos folder. For information on how to capture data using Mmwave Studio, consult the *Mmwave Studio User's Guide* in Mmwave Studio GUI tools.

Note that Mmwave Studio, previously known as Radar Studio, supports two different platforms. One is *DCA1000EVM Data Capture Card User's Guide*, which has interface with xWRxxxx EVM that enables users to stream the ADC data over Ethernet. Another platform is *TSW140x High Speed Data Capture/Pattern Generator Card* which requires Mmwave devPack EVM and *High Speed Data Converter Pro GUI User's Guide*. TSW1400 EVM transmits the captured data through Serial Peripheral Interface (SPI) to PC (through onboard USB to SPI converter). The Radar Studio (older version of Mmwave Studio) only supports TSW1400 EVM with HSDC Pro. For more information, see Device Firmware Package for the *Radar Studio User's Guide* and mmWave Sensor Raw Data Capture Using the TSW1400 Board for the training video.

For differences in the user interface of Mmwave Studio for DCA1000 EVM and TSW1400, see the *Mmwave Studio User Guide* . DCA1000 design is based on Lattice FPGA and LVDS data is captured through Ethernet streaming. TSW1400 EVM, based on Altera FPGA, capture the LVDS data in different format. Data format also depends on the type of mmwave devices. Therefore, the different format need to be discussed separately.

The application report is organized as follows - Section 2 lists the prerequisites. Section 3 describes how to run and save ADC data using capture demo and CCS. Section 4 discusses the data format saved by capture demo and CCS. Section 5,Section 6,Section 7 and Section 8 have detailed description of the data format saved by Mmwave Studio for different xWRxxxx devices using DCA1000 and TSW1400, respectively. Section 9 provides Matlab script example to process the data captured by Mmwave Studio.

2 Prerequisites

This document assumes that the user has installed the necessary prerequisites for the Capture Demo, Mmwave Studio and is familiar with the instructions on how to use this software. This document has been tested against SDK 1.01.00.02, CCSv7.3, Mmwave Studio 1.0.0.0.

For information on how to run the Capture Demo and any necessary prerequisites, install the mmWave SDK. Also see the *mmWave SDK User's Guide*, found at C:\ti\mmwave_sdk_<version_number>\docs.

For information on how to run Mmwave Studio and any necessary prerequisites, install Mmwave Studio GUI tools. Also see the *Mmwave Studio User's Guide*, found at C:\ti\mmwave_studio_<version_number>\docs

3 Viewing and Saving Raw Data Using Capture Demo and CCS

This section discusses how to view and save the raw ADC data captured using the Capture Demo provided in the mmWave SDK. This section assumes that the user is familiar with how to run the Capture Demo using CCS. For information on how to run the Capture Demo, consult the *mmWave SDK User's Guide* , found at C:\ti\mmwave_sdk_<version_number>\docs.

The ADC data is stored in L3 can be viewed and saved using the Memory Browser in CCS. To access the Memory Browser, open the View tab and select the Memory Browser option, shown in Figure 1.

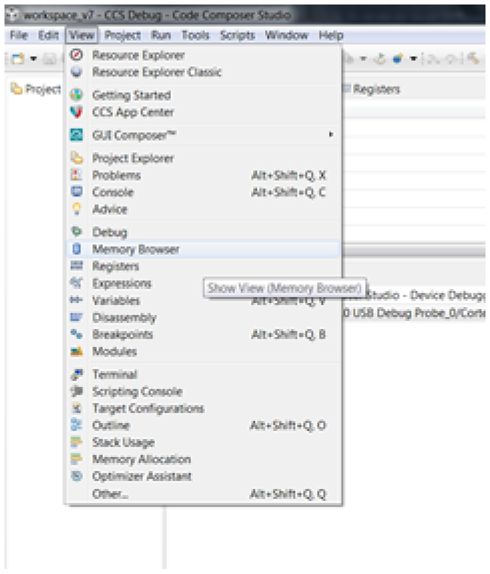
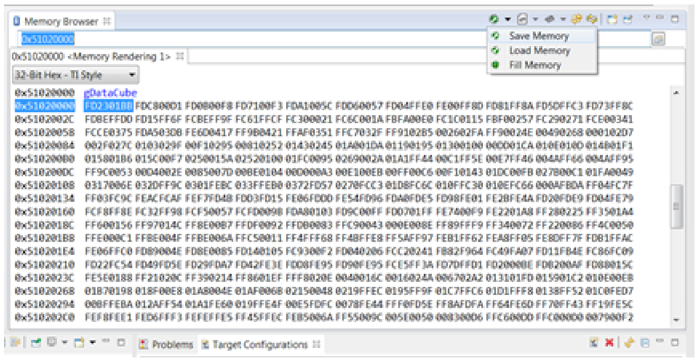


Figure 1. Opening Memory Browser in CCS

In the Memory Browser window, the contents of the L3 memory can be viewed by specifying the start address of the L3 memory. The start address is specified by CAPTURE_L3RAM_DATA_MEM_ADDRESS in the capture.h file and varies based on SDK version. The total size of memory reserved to store ADC data is specified by CAPTURE_L3RAM_DATA_MEM_SIZE in capture.h. Additionally, the L3 address and size are output in the last line of the CCS console before sending the configuration file to the device.

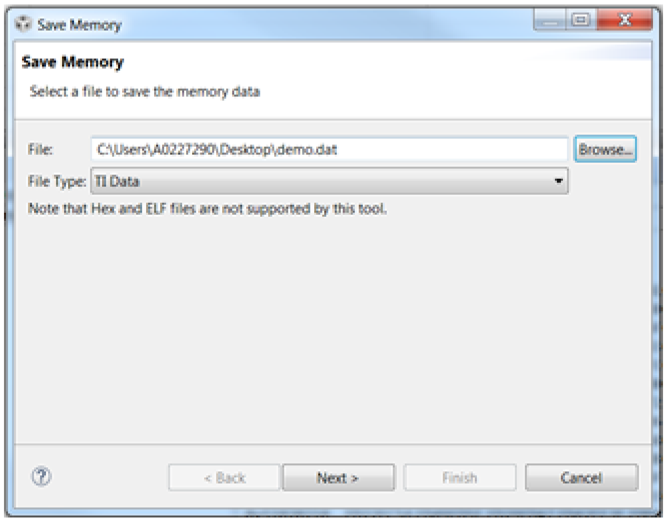
To save the raw ADC data click the Save Memory button at the top right corner of the Memory Browser window as shown in Figure 2.



```
0x51020000  gDataCube
0x51020000  FD230188 FDC800D1 FD0800F8 FD7100F3 FDA1005C FDD60057 FD04FFE0 FE00FF8D FD81FF8A FD5DFFC3 FD73FF8C
0x5102002C  FD8EFFD0 FD15FF6F FCBEFF9F FC61FFCF FC300021 FC6C001A FBFA00E0 FC1C0115 FBF00257 FC290271 FCE00341
0x51020058  FCCE0375 FDA503D8 FE6D0417 FF9B0421 FFAF0351 FFC7032F FF9102B5 002602FA FF90024E 00490268 000102D7
0x51020084  002F027C 0103029F 00F10295 00810252 01430245 01A001DA 01190195 01300100 00D001CA 010E010D 014B01F1
0x510200B0  015801B6 015C00F7 0250015A 02520100 01FC0095 0269002A 01A1FF44 00C1FF5E 00E7FF46 004AFF66 004AFF95
0x510200DC  FF9C0053 00D4002E 0085007D 00BE0104 00D000A3 00E100EB 00FF00C6 00F10143 01DC00FB 027800C1 01FA0049
0x51020108  0317006E 032DFF9C 0301FEBC 033FFEB0 0372FD57 0270FCC3 01D8FC6C 010FFC30 010EFC66 000AFBDA FF04FC7F
0x51020134  FF03FC9C FEACFCAF FEF7FD48 FD03FD15 FE06FD00 FE54FD96 FDA0FDE5 FD98FE01 FE28FE4A FD20FDE9 FD04FE79
0x51020160  FCF8FF8E FC32FF98 FCF50057 FCFD0098 FDA80103 FD9C00FF FD0701FF FE7400F9 FE2201A8 FF280225 FF3501A4
0x5102018C  FF600156 FF97014C FF8E0087 FFDF0092 FFD80083 FFC90043 000E008E FF89FFF9 FF340072 FF220086 FF4C0050
0x510201B8  FFE000C1 FFBE004F FFBE006A FFC50011 FF4FFF68 FF4BFFE8 FF5AFF97 FEB1FF62 FEA8FF05 FE8DFF7F FD81FFAC
0x510201E4  FE06FFC0 FD89004E FD8E0085 FD140105 FC9300F2 FD040206 FCC20241 FB82F964 FC49FA07 FD11FB4E FC86FC09
0x51020210  FD22FC54 FD49FD5E FD29FDA7 FD42FE3E FD08FE95 FD90FE95 FCE5FF3A FD7DFFD1 FD20008E FD8200AF FD88015C
0x5102023C  FE5E0188 FF21020C FF390214 FF8601EF FFF8020E 0040016C 0016024A 006702A2 013101FD 015901C2 010E00E8
0x51020268  01870198 018F00E8 01A8004E 01AF0068 02150048 0219FFEC 0195FF9F 01C7FFC6 01D1FFF8 0138FF52 01C0FED7
0x51020294  008FFEBA 012AFF54 01A1FE60 019FFE4F 00E5FDFC 0078FE44 FFF0FD5E FF8AFDFA FF64FE6D FF70FF43 FF19FE5C
0x510202C0  FEF8FEE1 FED6FFF3 FEFEFFE5 FF45FFEC FE85006A FF55009C 005E0050 008300D6 FFC600D0 FFC000D0 007900F2
```

Figure 2. Memory Browser *Save Memory* Button

In the Save Memory menu, specify the file path and file name and select TI Data as the file type as shown in Figure 3.



Select a file to save the memory data

File: C:\Users\A0227290\Desktop\demo.dat

File Type: TI Data

Note that Hex and ELF files are not supported by this tool.

Figure 3. Save Memory and File Type

Click Next and specify the format, start address and length. For the format, select 16-Bit Hex – TI Style. The length of the data is based on the users chirp configuration. To calculate the length in words, the following formulas can be used. First, find the total size of ADC data generated in bytes.

$$\text{Total Size in Bytes} = \text{Num ADC Samples} \times \text{Num RX Channels} \times \text{Num Frames} \times \text{Num Chirps} \times \text{Num Bytes Per Sample} \quad (1)$$

The number of bytes per sample is based the output format specified by the adcCfg parameter in the sensor configuration. For a complex output format, there are 4 bytes per sample: 2 bytes for the real part and 2 bytes for the imaginary part. For a real output format, there are 2 bytes per sample. Once the total size of the data in bytes is determined the number of words can be found by dividing the total size by 16; this assumes that the user is saving in a **16-bit Hex – TI Style** format.

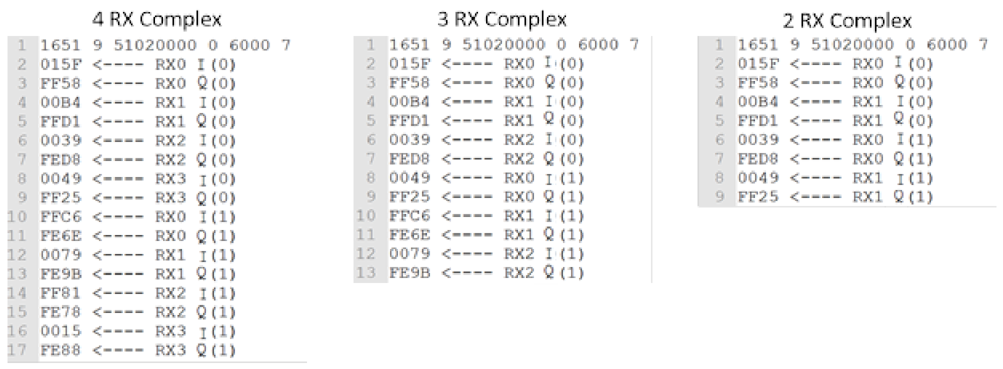
$$\text{Length in Words} = \text{Total Size} / 2 \text{ bytes per word} \quad (2)$$

Once the data is saved, it can be viewed in MATLAB using the capture_demo.m file located at C:\mmwave_sdk_<version_number>\packages\ti\demo\xwr14xx\capture\gui for the xWR1xx, and located at C:\ti\mmwave_sdk_01_00_00_05\packages\ti\demo\xwr16xx\capture\gui for the xWR16xx. This file plots the raw ADC data and the 1D FFT of the ADC data. This code can be modified to complete additional post processing on the data. To do further post processing, you must understand how the output data is formatted, as shown in Section 4.

4 Raw Data Format of Capture Demo Using CCS

Data captured using the Capture Demo can be stored in interleaved or non-interleaved mode, as specified by the adcbufCfg command in the sensor configuration. For xWR14xx devices, interleaved mode is recommended but non-interleaved mode can also be used. For xWR16xx/IWR6843 devices only non-interleaved mode can be used. By default the xWR14xx is configured for interleaved mode and the xWR16xx/IWR6843 is configured for non-interleaved mode. For more information about interleaving, see the *AWR1xxx Data Path Programmer's Guide* programmer’s guide. Data is stored in a 16-bit, two's-complement format.

Figure 4 shows the output data for interleaved mode with complex data for 4 RX, 3 RX, and 2 RX antennas..



4 RX Complex

```
1651 9 51020000 0 6000 7
015F <---- RX0 I(0)
FF58 <---- RX0 Q(0)
00B4 <---- RX1 I(0)
FFD1 <---- RX1 Q(0)
0039 <---- RX2 I(0)
FED8 <---- RX2 Q(0)
0049 <---- RX3 I(0)
FF25 <---- RX3 Q(0)
FFC6 <---- RX0 I(1)
FE6E <---- RX0 Q(1)
0079 <---- RX1 I(1)
FE9B <---- RX1 Q(1)
FF81 <---- RX2 I(1)
FE78 <---- RX2 Q(1)
0015 <---- RX3 I(1)
FE88 <---- RX3 Q(1)
```

3 RX Complex

```
1651 9 51020000 0 6000 7
015F <---- RX0 I(0)
FF58 <---- RX0 Q(0)
00B4 <---- RX1 I(0)
FFD1 <---- RX1 Q(0)
0039 <---- RX2 I(0)
FED8 <---- RX2 Q(0)
0049 <---- RX0 I(1)
FF25 <---- RX0 Q(1)
FFC6 <---- RX1 I(1)
FE6E <---- RX1 Q(1)
0079 <---- RX2 I(1)
FE9B <---- RX2 Q(1)
```

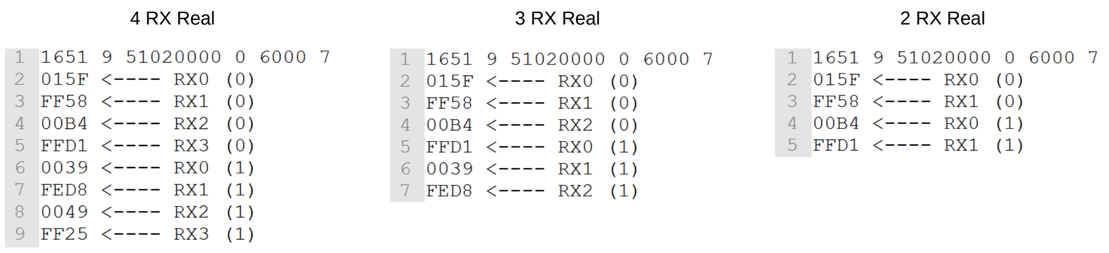
2 RX Complex

```
1651 9 51020000 0 6000 7
015F <---- RX0 I(0)
FF58 <---- RX0 Q(0)
00B4 <---- RX1 I(0)
FFD1 <---- RX1 Q(0)
0039 <---- RX0 I(1)
FED8 <---- RX0 Q(1)
0049 <---- RX1 I(1)
FF25 <---- RX1 Q(1)
```

Figure 4. Interleaved Complex Data Format Using Capture Demo

Figure 4 represents how the data is saved when using **16-bit Hex – TI Style** when the file is saved as a TI data format file. In Figure 4, RX0 I(0) represents the first sample of real data received at RX0, and RX0 Q(0) represents the first sample of imaginary data received at RX0. RX0 I(1) represents the second sample of real data received at RX0, and so on.

Data can also be captured as real data only. Figure 5 shows how the data is saved for interleaved mode with real data for 4 RX, 3 RX, and 2 RX antennas.



4 RX Real

```
1651 9 51020000 0 6000 7
015F <---- RX0 (0)
FF58 <---- RX1 (0)
00B4 <---- RX2 (0)
FFD1 <---- RX3 (0)
0039 <---- RX0 (1)
FED8 <---- RX1 (1)
0049 <---- RX2 (1)
FF25 <---- RX3 (1)
```

3 RX Real

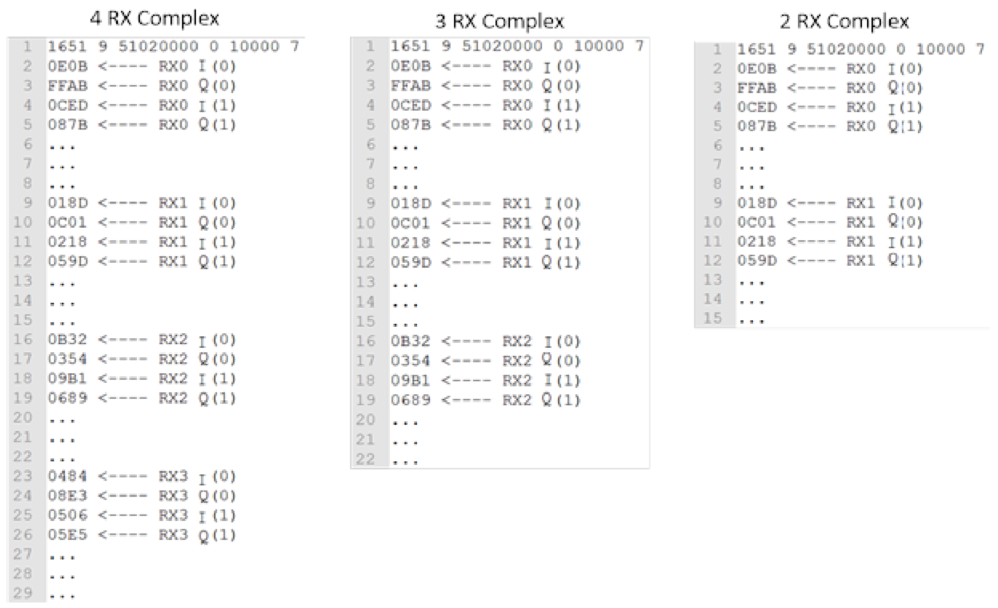
```
1651 9 51020000 0 6000 7
015F <---- RX0 (0)
FF58 <---- RX1 (0)
00B4 <---- RX2 (0)
FFD1 <---- RX0 (1)
0039 <---- RX1 (1)
FED8 <---- RX2 (1)
```

2 RX Real

```
1651 9 51020000 0 6000 7
015F <---- RX0 (0)
FF58 <---- RX1 (0)
00B4 <---- RX0 (1)
FFD1 <---- RX1 (1)
```

Figure 5. Interleaved Real Data Format Using Capture Demo

Data can also be stored in a non-interleaved format. For the xWR16xx/IWR6843, data can only be stored in non-interleaved format. However, for the xWR14xx, interleaved mode is recommended. Figure 6 shows the output format of the data for non-interleaved complex data.



4 RX Complex

```
1651 9 51020000 0 10000 7
0E0B <---- RX0 I(0)
FFAB <---- RX0 Q(0)
0CED <---- RX0 I(1)
087B <---- RX0 Q(1)
...
...
...
018D <---- RX1 I(0)
0C01 <---- RX1 Q(0)
0218 <---- RX1 I(1)
059D <---- RX1 Q(1)
...
...
...
0B32 <---- RX2 I(0)
0354 <---- RX2 Q(0)
09B1 <---- RX2 I(1)
0689 <---- RX2 Q(1)
...
...
...
0484 <---- RX3 I(0)
08E3 <---- RX3 Q(0)
0506 <---- RX3 I(1)
05E5 <---- RX3 Q(1)
...
...
...
```

3 RX Complex

```
1651 9 51020000 0 10000 7
0E0B <---- RX0 I(0)
FFAB <---- RX0 Q(0)
0CED <---- RX0 I(1)
087B <---- RX0 Q(1)
...
...
...
018D <---- RX1 I(0)
0C01 <---- RX1 Q(0)
0218 <---- RX1 I(1)
059D <---- RX1 Q(1)
...
...
...
0B32 <---- RX2 I(0)
0354 <---- RX2 Q(0)
09B1 <---- RX2 I(1)
0689 <---- RX2 Q(1)
...
...
...
```

2 RX Complex

```
1651 9 51020000 0 10000 7
0E0B <---- RX0 I(0)
FFAB <---- RX0 Q(0)
0CED <---- RX0 I(1)
087B <---- RX0 Q(1)
...
...
...
018D <---- RX1 I(0)
0C01 <---- RX1 Q(0)
0218 <---- RX1 I(1)
059D <---- RX1 Q(1)
...
...
...
```

Figure 6. Non-Interleaved Complex Data Format Using Capture Demo

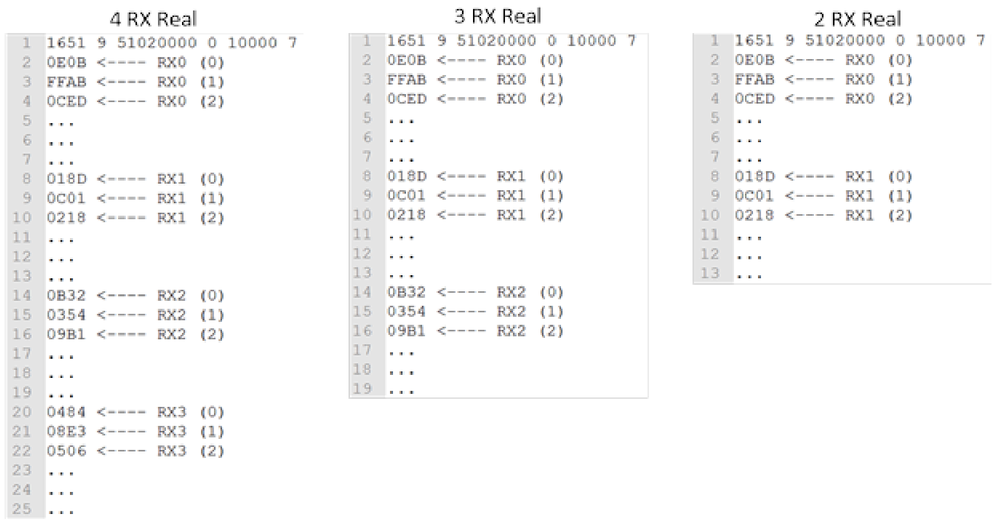
In non-interleaved format, all of the data for a given RX is stored in a single block of memory followed by all of the data for the next RX in another block of memory. For example, all of the RX0 data is stored sequentially, followed by all of the RX1 data. To determine the amount of memory occupied by each RX, use Equation 3.

$$\text{Bytes Per RX} = \text{Num ADC Samples} \times \text{Num Frames} \times \text{Num Chirps} \times \text{Num Bytes Per Sample} \quad (3)$$

For complex data, there are four bytes per sample. For real data, there are two bytes per sample. For non-interleaved data, the number of words per RX can be determined using Equation 4.

$$\text{Words per RX} = \text{Bytes per RX} / 2 \text{ bytes per word} \quad (4)$$

Figure 7 shows the output format of the data for non-interleaved real data.



4 RX Real

```
1651 9 51020000 0 10000 7
0E0B <---- RX0 (0)
FFAB <---- RX0 (1)
0CED <---- RX0 (2)
...
...
...
018D <---- RX1 (0)
0C01 <---- RX1 (1)
0218 <---- RX1 (2)
...
...
...
0B32 <---- RX2 (0)
0354 <---- RX2 (1)
09B1 <---- RX2 (2)
...
...
...
0484 <---- RX3 (0)
08E3 <---- RX3 (1)
0506 <---- RX3 (2)
...
...
...
```

3 RX Real

```
1651 9 51020000 0 10000 7
0E0B <---- RX0 (0)
FFAB <---- RX0 (1)
0CED <---- RX0 (2)
...
...
...
018D <---- RX1 (0)
0C01 <---- RX1 (1)
0218 <---- RX1 (2)
...
...
...
0B32 <---- RX2 (0)
0354 <---- RX2 (1)
09B1 <---- RX2 (2)
...
...
...
```

2 RX Real

```
1651 9 51020000 0 10000 7
0E0B <---- RX0 (0)
FFAB <---- RX0 (1)
0CED <---- RX0 (2)
...
...
...
018D <---- RX1 (0)
0C01 <---- RX1 (1)
0218 <---- RX1 (2)
...
...
...
```

Figure 7. Non-Interleaved Real Data Format Using Capture Demo

Similar to the non-interleaved mode with complex data, in non-interleaved mode with real data, all of the data for a given RX is stored in a single block of memory followed by all of the data for the next RX in the next block of memory. For example, if there are 1024 words per RX channel, all 1024 words for RX0 will be stored first, followed by all 1024 words for RX1, and so on.

5 xWR12xx and xWR14xx With DCA1000 Data Format

In addition to the Capture Demo, ADC data can also be saved using Mmwave Studio. This section discusses the data format of data saved using Mmwave Studio and DCA1000 EVM for the xWR12xx and xWR14xx.

For xWR12xx and xWR14xx devices, data is captured over four LVDS lanes and is stored in a binary file in an interleaved format. The DCA1000 captured data samples are two bytes long and in the two's-complement format. Each LVDS lane corresponds to a given receiver. For real data, one real sample from each LVDS lane will be stored. If there is a total of M number of chirps, the data is stored beginning with first chirp and ending with the Mth chirp. Figure 8 shows how data is stored in the binary file for real data of M number of chirps and N number of ADC samples per chirp when four receivers are enabled.

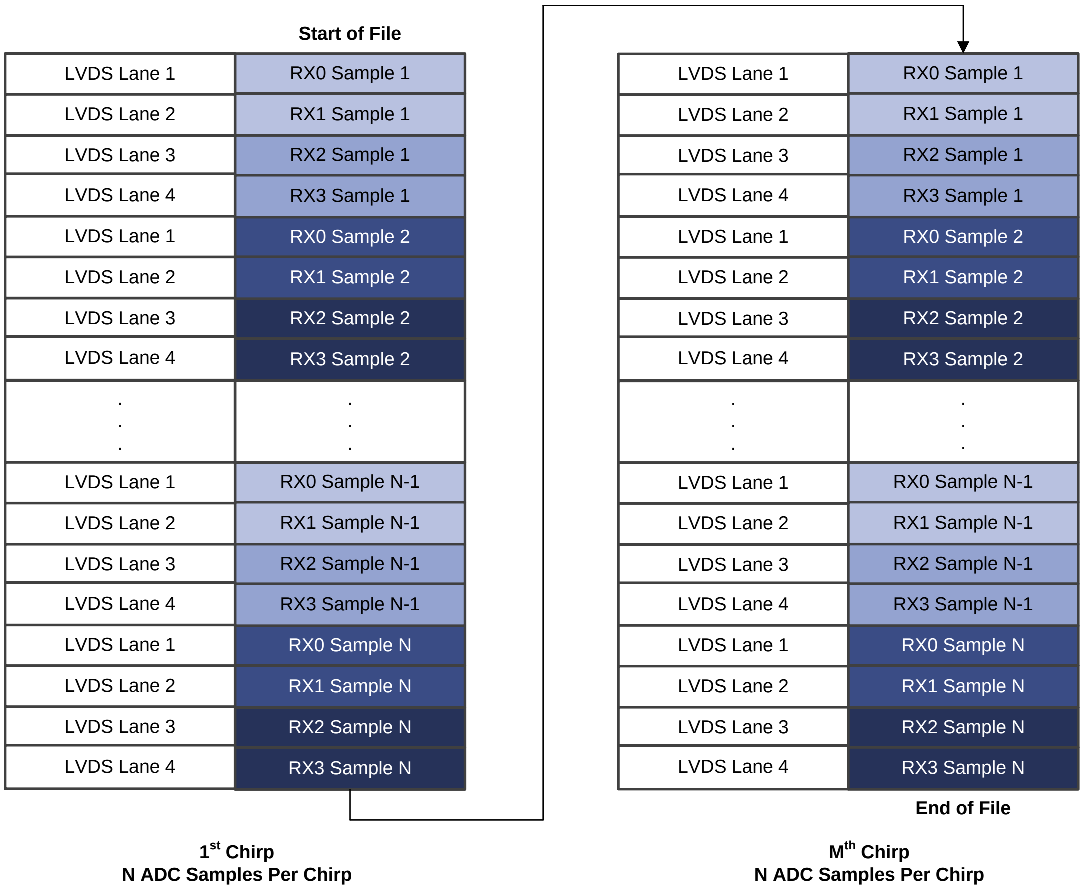

Start of File
LVDS Lane 1 | RX0 Sample 1
LVDS Lane 2 | RX1 Sample 1
LVDS Lane 3 | RX2 Sample 1
LVDS Lane 4 | RX3 Sample 1
LVDS Lane 1 | RX0 Sample 2
LVDS Lane 2 | RX1 Sample 2
LVDS Lane 3 | RX2 Sample 2
LVDS Lane 4 | RX3 Sample 2
. . . | . . .
LVDS Lane 1 | RX0 Sample N-1
LVDS Lane 2 | RX1 Sample N-1
LVDS Lane 3 | RX2 Sample N-1
LVDS Lane 4 | RX3 Sample N-1
LVDS Lane 1 | RX0 Sample N
LVDS Lane 2 | RX1 Sample N
LVDS Lane 3 | RX2 Sample N
LVDS Lane 4 | RX3 Sample N
End of File
1st Chirp
N ADC Samples Per Chirp
Mth Chirp
N ADC Samples Per Chirp

Figure 8. xWR14xx Real Data Format Using DCA1000

Any unused LVDS lanes will be populated with zeros. The number of enabled receivers should match the number of LVDS lanes to be enabled. The assignment of receiver to the LVDS lane is beginning with the lowest number receiver and ending with the highest number receiver. For example, if receivers 1 and 3 are enabled and LVDS 2 and 3 are enabled, then LVDS lane 2 will contain the data from receiver 1 and LVDS lane 3 will contain the data from receiver 3. Lanes 1 and 4 will be populated with zeros. Figure 9 shows the data storage format of the binary file for complex data.

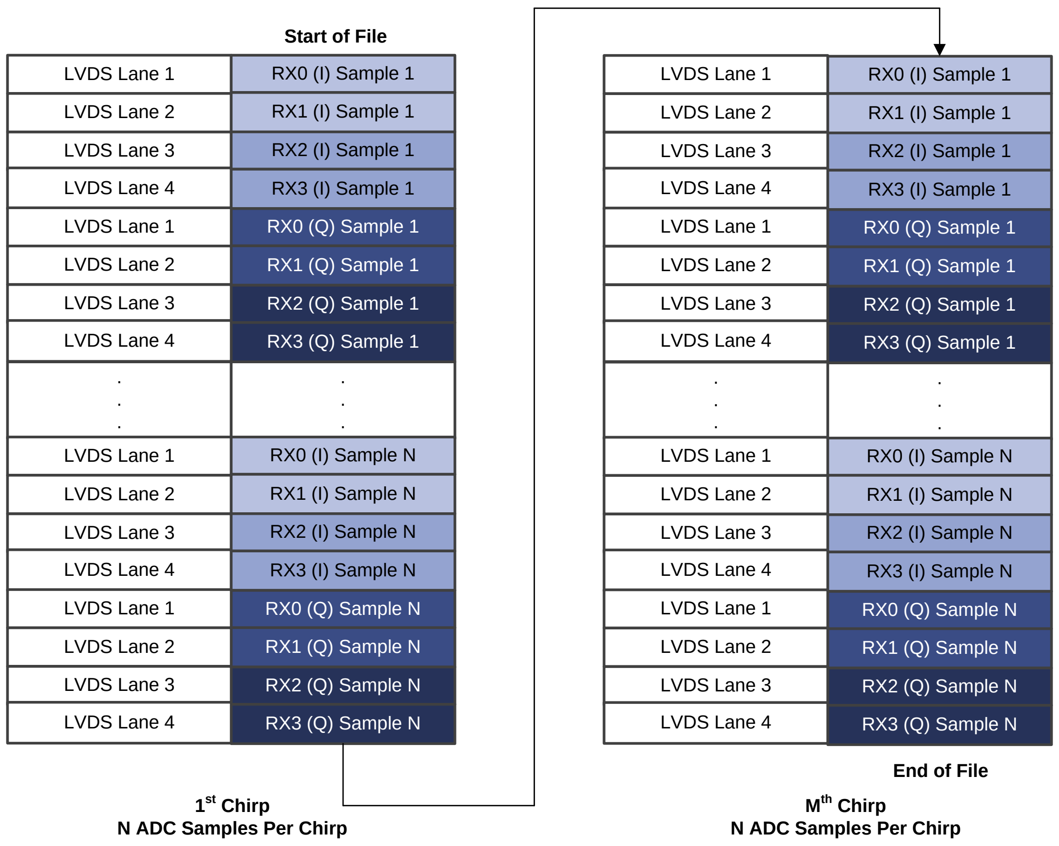


Figure 9. xWR14xx Complex Data Format Using DCA1000

For the complex data format, each sample consists of both a real and an imaginary part and each LVDS lane captures the complex samples per receiver. If "I first" option is selected in "IQ Swap" option of Mmwave Studio, real part is stored first for each receiver and then followed by the imaganry part from each receiver. Any unused LVDS lanes will be populated with zeros. For example, if receivers 1 and 3 are enabled and LVDS lane 1 and 2 are enabled, then LVDS lane 1 will contain the data from receiver 1 and LVDS lane 2 will contain the data from receiver 3. Lanes 3 and 4 will be populated with zeros.

6 xWR16xx and IWR6843 With DCA1000 Data Format

This section discusses the format of data saved from xWR16xx/IWR6843 using Mmwave Studio and DCA1000. For xWR16xx/IWR6843 devices, data is captured over two LVDS lanes and is stored in a binary file in non-inerleaved format. Data samples captured by DCA1000 are 2 bytes long and in two's-complement format. Because only two LVDS lanes are used, data capture using 3 receivers is not available with the xWR16xx/IWR6843. Only 1, 2, and 4 enabled receivers can be used.

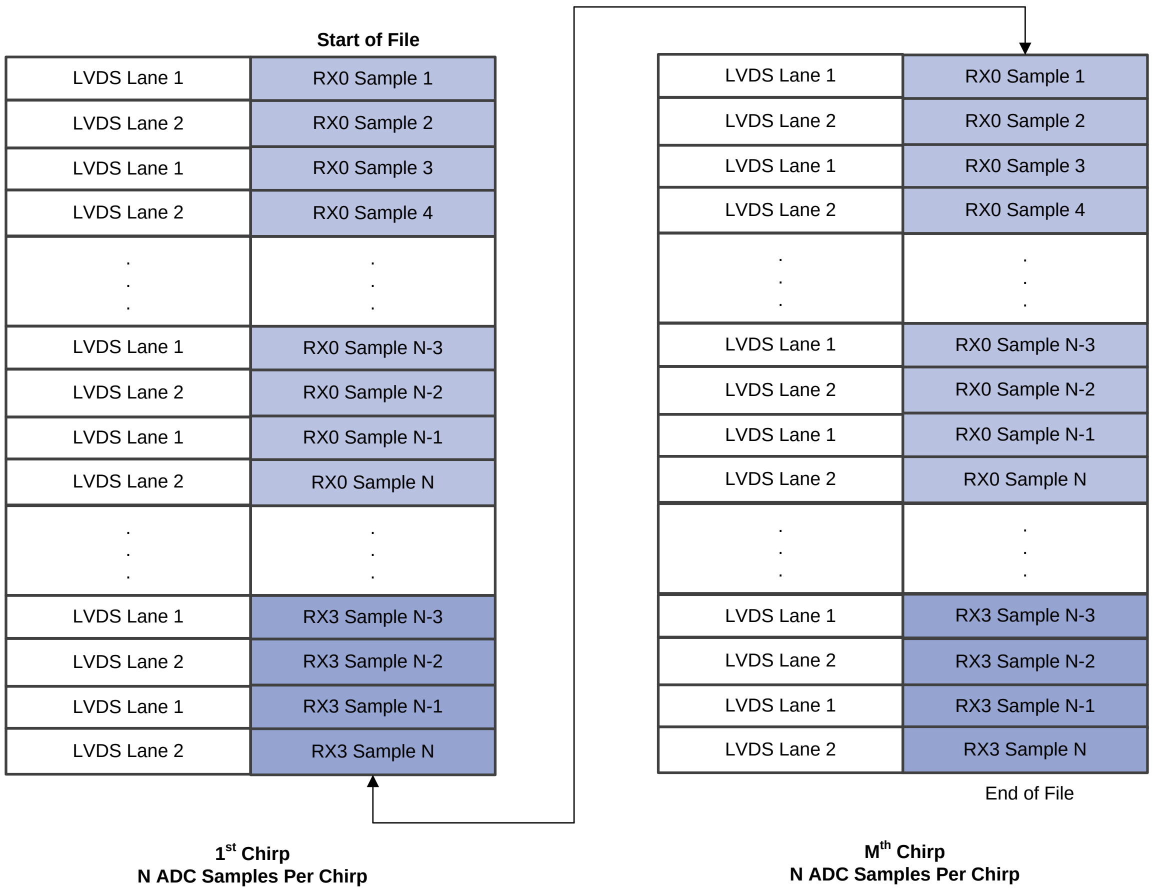


Figure 10. xWR16xx/IWR6843 Real Data Format Using DCA1000

Figure 10 shows how real data is captured for four receivers. Each chirp is stored in the order of the receiver, beginning with the lowest number receiver and ending with the highest number receiver. Lane 1 sends the odd samples from each receiver and lane 2 sends the even samples from each receiver. If there is a total of M number of chirps, the data is stored beginning with first chirp and ending with the Mth chirp. For any receivers that are disabled, the section corresponding to the disabled receivers can be removed from the diagram in Figure 10 to determine the total data output.

For complex data, lane 1 contains the real part of the sample and lane 2 contains the imaginary part of the sample. As shown in Figure 11, when four receivers are enabled, file is stored in the order of the receiver, beginning with the lowest number receiver and ending with the highest number receiver. The saved file has non-interleaved format beginning with the real part of every two samples and followed by the imaginary part of the every two samples. If there is a total of M number of chirps, the data is stored beginning with first chirp and ending with the Mth chirp. For any receivers that are disabled, the section corresponding to the disabled receivers can be removed from the diagram in to determine the total data output.

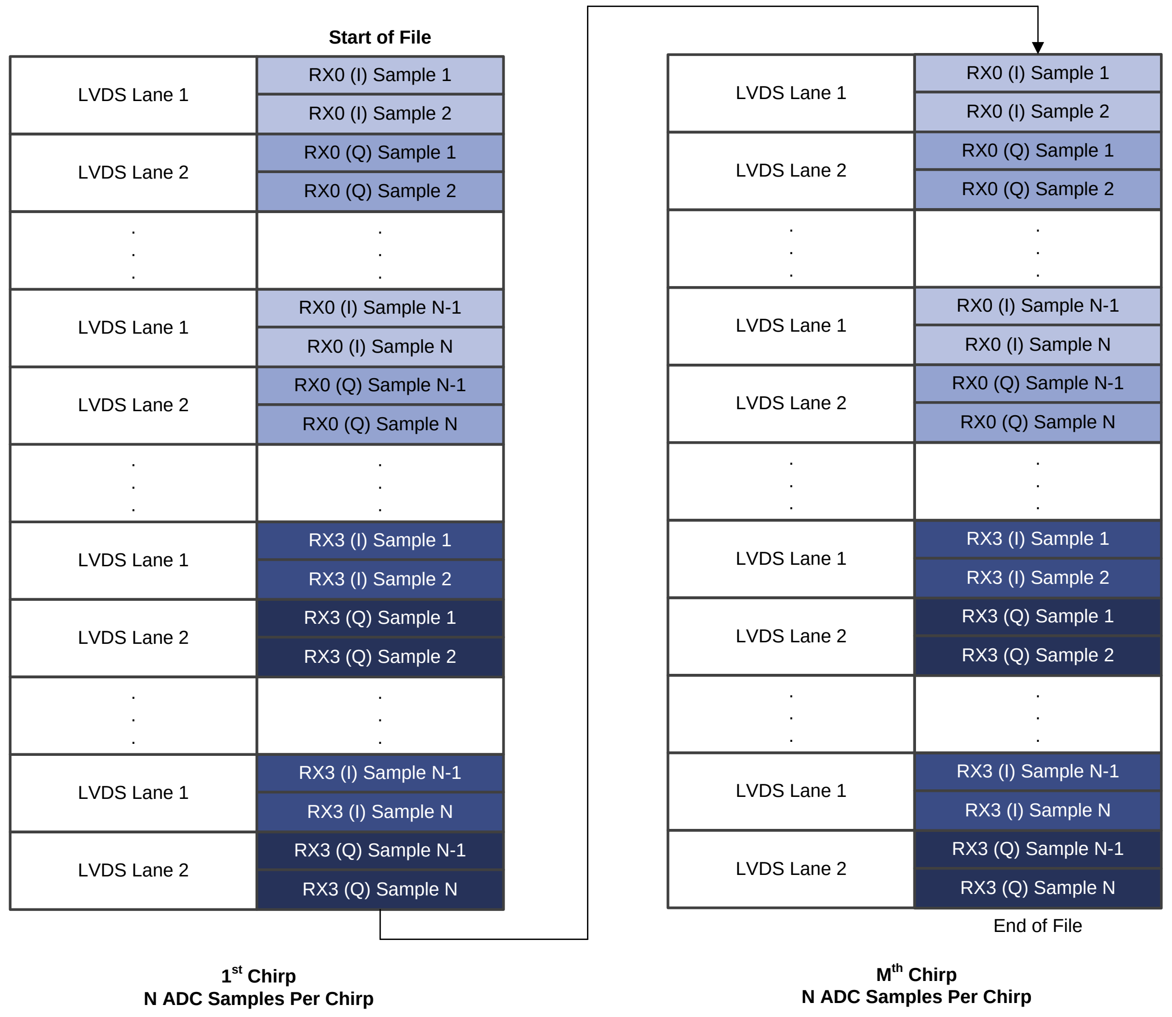


Figure 11. xWR16xx/IWR6843 Complex Data Format Using DCA1000

7 xWR12xx and xWR14xx With TSW1400 Data Format

ADC data can also be saved using Mmwave Studio and the TSW1400 EVM with HSDCPro. This section discusses the data format of data saved using Mmwave Studio for xWR12xx and xWR14xx.

For xWR12xx and xWR14xx devices, data is captured over four LVDS lanes and is stored in a binary file. The TSW1400 captured data samples are two bytes long and in the 'offset binary format'. This means that each sample has an extra 2^{15} added to it. In order to get the correct two's-complement number, 2^{15} must be subtracted from each sample. Each LVDS lane corresponds to a given receiver. For real data, each LVDS lane will capture two data samples per receiver and for complex data each LVDS lane will capture the real and imaginary part per receiver. Figure 12 shows how data is stored in the binary file for real data for M number of chirps and N number of ADC samples per chirp.

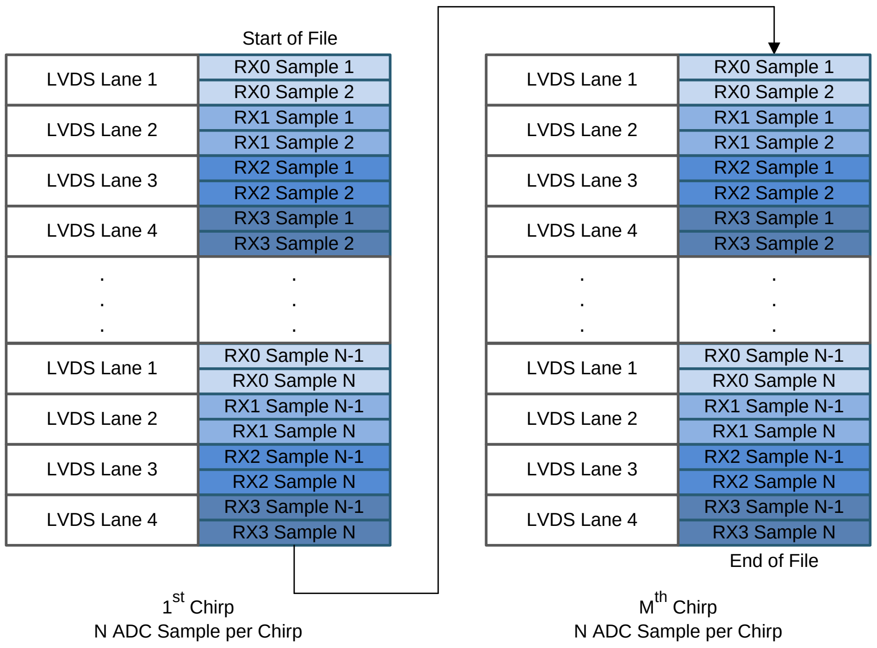


Figure 12. xWR14xx Real Data Format Using TSW1400

Eeach LVDS lane corresponds to a given receiver. If all four receivers are enabled, data will be stored as seen in Figure 12. Any unused LVDS lanes will be populated with zeros. For example, if receivers 1 and 3 are enabled then LVDS lane 1 will contain the data from receiver 1 and LVDS lane 2 will contain the data from receiver 3. Lanes 3 and 4 will be populated with zeros. Figure 13 shows the data storage format of the binary file for complex data.

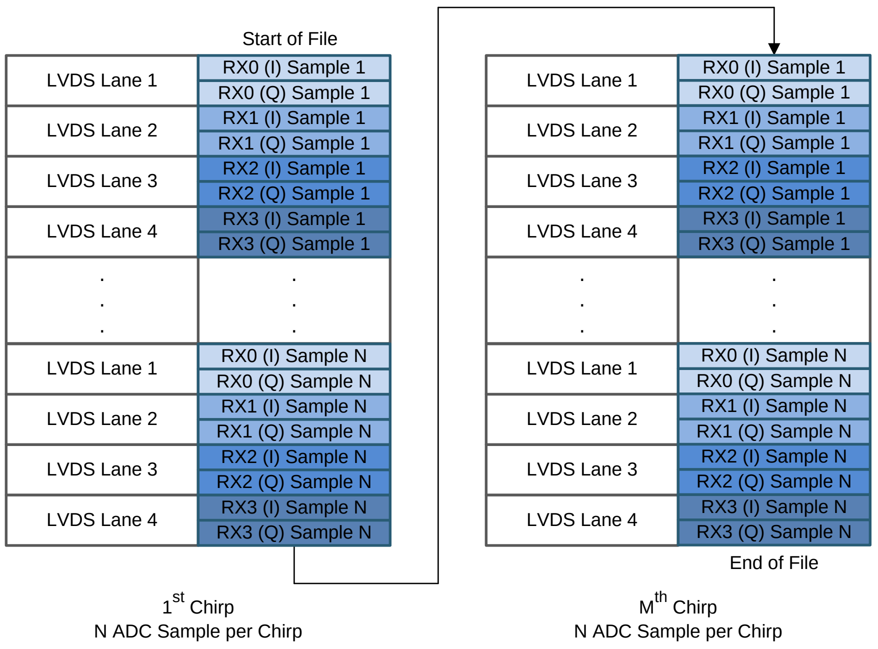


Figure 13. xWR14xx Complex Data Format Using TSW1400

The complex data format differs from the real data format in that each LVDS lane now only captures one sample per receiver. Each sample consists of both a real and an imaginary part. Similarly to the real data format, any unused LVDS lanes will be populated with zeros. For example, if receivers 1 and 3 are enabled then LVDS lane 1 will contain the data from receiver 1 and LVDS lane 2 will contain the data from receiver 3. Lanes 3 and 4 will be populated with zeros.

8 xWR16xx and IWR6843 With TSW1400 Data Format

This section discusses the data format of data saved using TSW1400 with xWR16xx/IWR6843. For xWR16xx/IWR6843 devices, data is captured over two LVDS lanes and is stored in a binary file. Data samples captured by TSW1400 via HSDCPro are 2 bytes long and in the 'offset binary format'. This means that each sample has an extra 2^{15} added to it. In order to get the correct two's-complement number, 2^{15} must be subtracted from each sample. Because only two LVDS lanes are used, data capture using 3 receivers is not available with the xWR16xx/IWR6843. Only 1, 2, and 4 enabled receivers can be used. Figure 14 shows how real data is captured for one receiver.

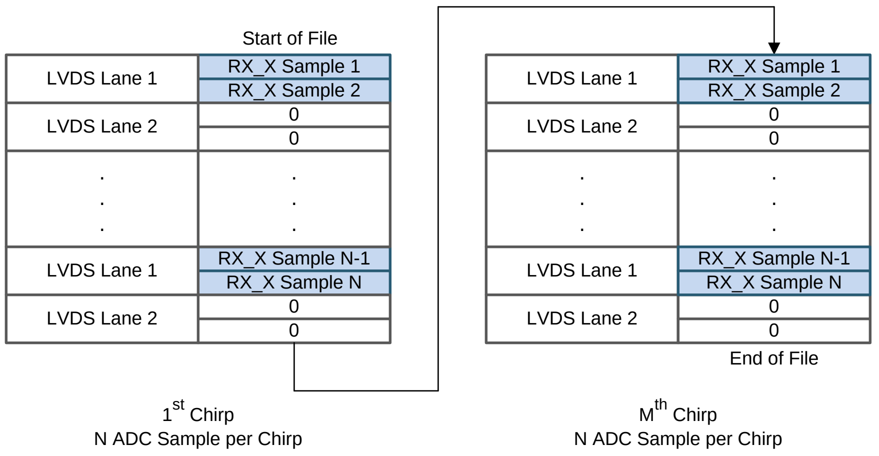


Figure 14. xWR16xx/IWR6843 Real Data Format for One Receiver Using TSW1400

When only one receiver is enabled only the first LVDS lane is used. Data sent from the second LVDS lane is populated with zeros. If there are N number of ADC samples and M number of chirps, the data is stored beginning with samples 1 through N of the first chirp through to samples 1 through N of the Mth chirp. If multiple receivers are enabled the data is stored in a different format, shown in Figure 15.

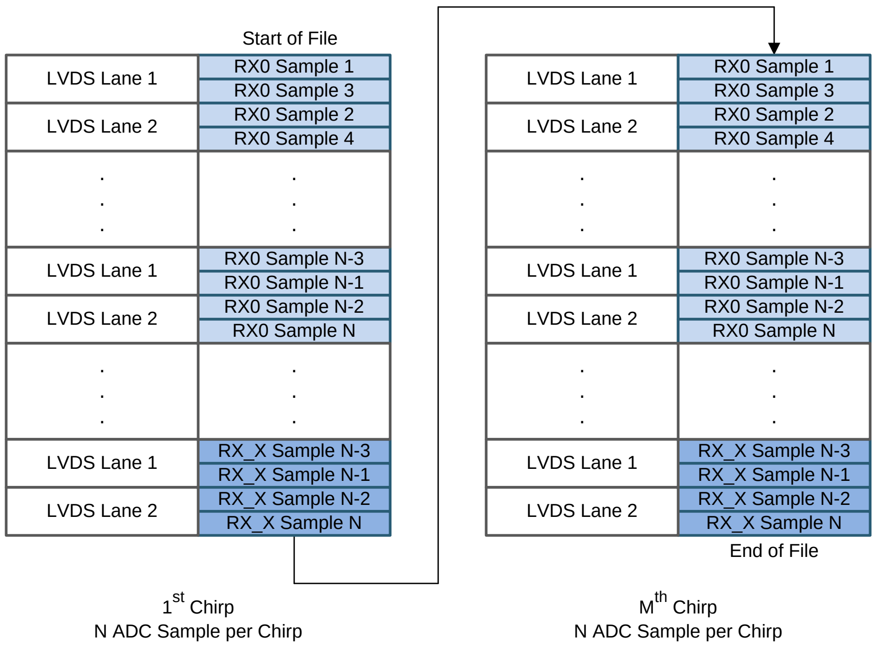


Figure 15. xWR16xx/IWR6843 Real Data Format for Multiple Receivers Using TSW1400

When multiple receivers are enabled both LVDS lanes are used, with lane 1 sending the odd samples from each receiver and lane 2 sending the even samples from each receiver. If there is a total of M number of chirps, the data is stored beginning with first chirp and ending with the Mth chirp. Each chirp is stored in the order of the receiver, beginning with the lowest number receiver and ending with the highest number receiver. For complex data, each LVDS lane sends a real part and an imaginary part. The format for complex data is shown in Figure 16.

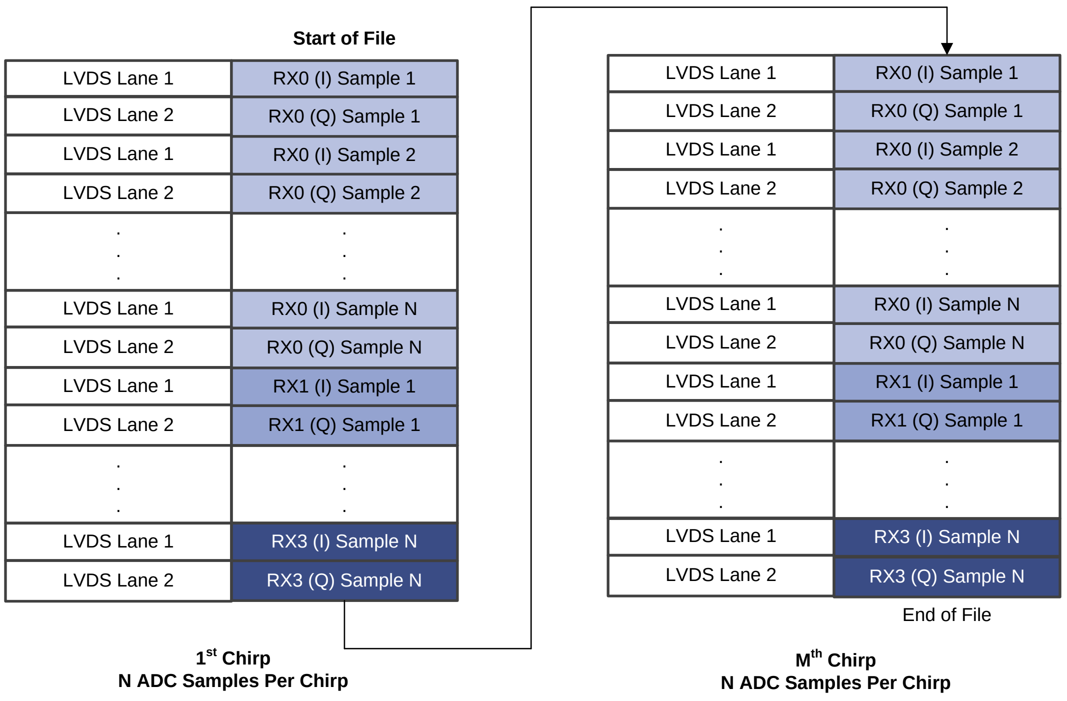


Figure 16. xWR16xx/IWR6843 Complex Data Format Using TSW1400

When multiple receivers are enabled, the complex data is stored in a non-interleaved format beginning with the real part followed by the imaginary part as shown in Figure 16. For each chirp, data is saved in the order of receiver, beginning with the lowest number receiver and ending with the highest number receiver. If there is a total of M number of chirps, the data is stored beginning with first chirp and ending with the Mth chirp. For any receivers that are disabled, the section corresponding to the disabled receivers can be removed from the diagram in Figure 16 to determine the total data output.

9 Interpreting Binary File in MATLAB

The data captured in Mmwave Studio can be processed using the Mmwave Studio GUI. However, any custom post processing will require the user to create a custom script. This section contains sample MATLAB code snippet to help the user to interpret binary files captured using the xWR14xx and xWR16xx to allow for custom post processing.

The data file stored by Mmwave Studio is located under the PostProc folder under the installation path, by default it is adc_data.bin. For TSW1400, data is saved directly as adc_data.bin. For DCA1000, after pressing "DCA1000 ARM" and "Trigger Frame", raw data file called adc_data_RAW_0.bin is saved first. Due to the Ethernet protocol, the file received by the DCA1000 EVM may not contain the UDP packets in correct order. There could be scenarios when the data is missing and not captured in the file. The packet reorder and zero fill utility needs to be run to scan through the raw captured file and re-arrange the packets in correct order and in case of any missing packets; it will fill an equal amount of zeros in the file. This utility also removes the meta data and the output now contains the raw ADC data as transferred by the xWRxxxx device. By pressing "PostProc" button, adc_data_RAW_0.bin will go through packet reorder and zero fill and is saved as adc_data.bin. Same result can be obtained by running the Packet_Reorder_Zerofill.exe, as instructed in the *Mmwave Studio User's Guide*. All following matlab scripts are for decoding data file adc_data.bin -- the raw ADC data tranferred by the xWRxxxx device.

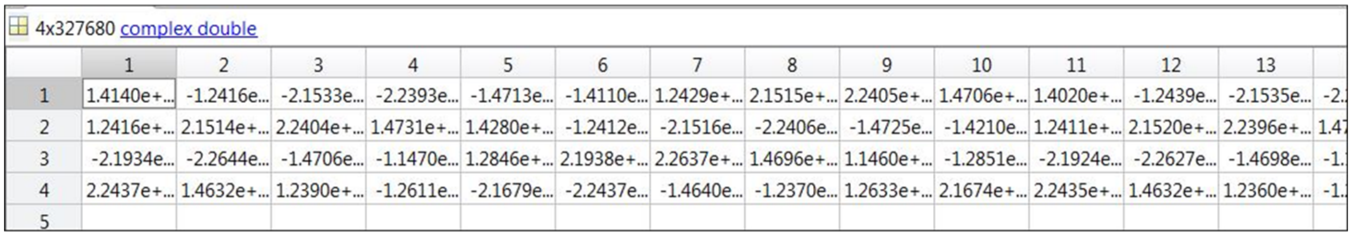
9.1 DCA1000 With xWR12xx and xWR14xx MATLAB Example

The function in the following MATLAB code example can be used to read files saved with the Mmwave Studio and xWR14xx using DCA1000.

```
%%% This script is used to read the binary file produced by the DCA1000
%%% and Mmwave Studio
%%% Command to run in Matlab GUI - readDCA1000('<ADC capture bin file>')

function [retVal] = readDCA1000(fileName)
%% global variables
% change based on sensor config
numADCBits = 16; % number of ADC bits per sample
numLanes = 4; % do not change. number of lanes is always 4 even if only 1 lane is used. unused
lanes
isReal = 0;  % set to 1 if real only data, 0 if complex dataare populated with 0 %% read file and
convert to signed number

% read .bin file
fid = fopen(fileName,'r');
% DCA1000 should read in two's complement data
adcData = fread(fid, 'int16');
% if 12 or 14 bits ADC per sample compensate for sign extension
if numADCBits ~= 16
    l_max = 2^(numADCBits-1)-1;
    adcData(adcData > l_max) = adcData(adcData > l_max) - 2^numADCBits;
end
fclose(fid);

%% organize data by LVDS lane
% for real only data
if isReal
    % reshape data based on one samples per LVDS lane
    adcData = reshape(adcData, numLanes, []);
%for complex data
else
    % reshape and combine real and imaginary parts of complex number
    adcData = reshape(adcData, numLanes*2, []);
    adcData = adcData([1,2,3,4],:) + sqrt(-1)*adcData([5,6,7,8],:);
end
%% return receiver data
retVal = adcData;
```

The function in the previous code snippet will return a matrix with all of the receiver data stored in rows. The values for numADCBits and isReal should be modified based on the sensor configuration used. The value for numLanes should not be modified as there will always be four LVDS lanes used. Unused lanes will be populated with zeros.

The returned matrix of the function should look similar toFigure 17.


4x327680 complex double

	1	2	3	4	5	6	7	8	9	10	11	12	13	
1	1.4140e+...	-1.2416e...	-2.1533e...	-2.2393e...	-1.4713e...	-1.4110e...	1.2429e+...	2.1515e+...	2.2405e+...	1.4706e+...	1.4020e+...	-1.2439e...	-2.1535e...	-2.
2	1.2416e+...	2.1514e+...	2.2404e+...	1.4731e+...	1.4280e+...	-1.2412e...	-2.1516e...	-2.2406e...	-1.4725e...	-1.4210e...	1.2411e+...	2.1520e+...	2.2396e+...	1.47
3	-2.1934e...	-2.2644e...	-1.4706e...	-1.1470e...	1.2846e+...	2.1938e+...	2.2637e+...	1.4696e+...	1.1460e+...	-1.2851e...	-2.1924e...	-2.2627e...	-1.4698e...	-1.
4	2.2437e+...	1.4632e+...	1.2390e+...	-1.2611e...	-2.1679e...	-2.2437e...	-1.4640e...	-1.2370e...	1.2633e+...	2.1674e+...	2.2435e+...	1.4632e+...	1.2360e+...	-1.
5														

Figure 17. MATLAB Script Output -- xWR14xx With DCA1000

Row 1 contains all of the data from the first receiver, row 2 from the second receiver, row 3 from the third receiver, and row 4 from the fourth receiver. In cases where certain receivers are disabled, the corresponding rows will be populated with zeros. Each row will contain a number of columns equal to the number of ADC samples per chirp multiplied by the total number of chirps. The columns are organized by chirps. For example, if there are 256 ADC samples and a total of 1280 chirps (128 chirps/frame x 10 frames), each row will contain 327680 columns. The first 256 columns correspond to the first chirp, the next 256 columns to the second chirp, and so on. The first 32768 columns correspond to the chirps of the first frame, and so on. The data can then be processed as the user desires.

9.2 DCA1000 With x16xx and IWR6843 MATLAB Example

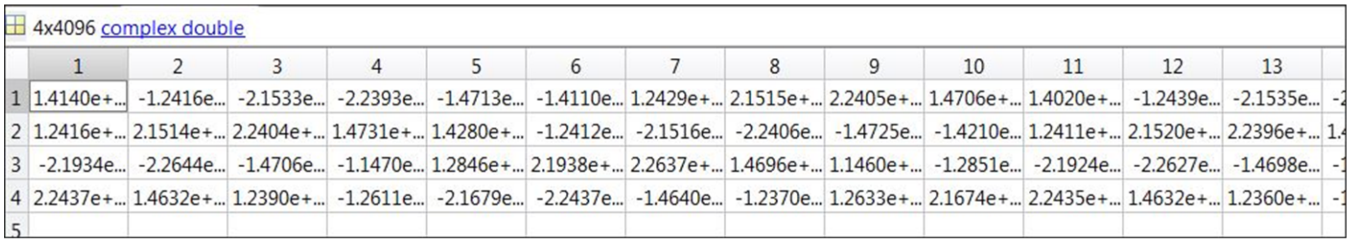
The function in the following MATLAB code example can be used to read files saved with the Mmwave Studio and xWR16xx with DCA1000. IWR6843 has the same dataformat as xWR16xx.

```
%%% This script is used to read the binary file produced by the DCA1000
%%% and Mmwave Studio
%%% Command to run in Matlab GUI -
 readDCA1000('<ADC capture bin file>')  function [retVal] = readDCA1000(fileName)
%% global variables
% change based on sensor config
numADCSamples = 256; % number of ADC samples per chirp
numADCBits = 16; % number of ADC bits per sample
numRX = 4; % number of receivers
numLanes = 2; % do not change. number of lanes is always 2
isReal = 0;  % set to 1 if real only data, 0 if complex data0
%% read file
% read .bin file
fid = fopen(fileName,'r');
adcData = fread(fid, 'int16');
% if 12 or 14 bits ADC per sample compensate for sign extension
if numADCBits ~= 16
    l_max = 2^(numADCBits-1)-1;
    adcData(adcData > l_max) = adcData(adcData > l_max) - 2^numADCBits;
end
fclose(fid);
fileSize = size(adcData, 1);
% real data reshape, filesize = numADCSamples*numChirps
if isReal
    numChirps = fileSize/numADCSamples/numRX;
    LVDS = zeros(1, fileSize);
    %create column for each chirp
    LVDS = reshape(adcData, numADCSamples*numRX, numChirps);
    %each row is data from one chirp
    LVDS = LVDS.';
else
    % for complex data
    % filesize = 2 * numADCSamples*numChirps
```

```
        numChirps = fileSize/2/numADCSamples/numRX;
        LVDS = zeros(1, fileSize/2);
        %combine real and imaginary part into complex data
        %read in file: 2I is followed by 2Q
        counter = 1;
        for i=1:4:fileSize-1
            LVDS(1,counter)     = adcData(i) + sqrt(-
1)*adcData(i+2);                       LVDS(1,counter+1)   = adcData(i+1)+sqrt(-
1)*adcData(i+3);                       counter = counter + 2;
        end
        % create column for each chirp
        LVDS = reshape(LVDS, numADCSamples*numRX, numChirps);
        %each row is data from one chirp
        LVDS = LVDS.';
end
%organize data per RX
adcData = zeros(numRX,numChirps*numADCSamples);
for row = 1:numRX
    for i = 1: numChirps
        adcData(row, (i-1)*numADCSamples+1:i*numADCSamples) = LVDS(i, (row-
1)*numADCSamples+1:row*numADCSamples);
    end
end
% return receiver data
retVal = adcData;
```

The function in the previous code snippet will return a matrix with all of the receiver data stored in rows. The values for numADCBits and isReal should be modified based on the sensor configuration used. The value for numLanes should not be modified as there will always be two LVDS lanes used.

The returned matrix of the function should look similar to Figure 18.



4x4096 complex double

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	1.4140e+...	-1.2416e...	-2.1533e...	-2.2393e...	-1.4713e...	-1.4110e...	1.2429e+...	2.1515e+...	2.2405e+...	1.4706e+...	1.4020e+...	-1.2439e...	-2.1535e...
2	1.2416e+...	2.1514e+...	2.2404e+...	1.4731e+...	1.4280e+...	-1.2412e...	-2.1516e...	-2.2406e...	-1.4725e...	-1.4210e...	1.2411e+...	2.1520e+...	2.2396e+...
3	-2.1934e...	-2.2644e...	-1.4706e...	-1.1470e...	1.2846e+...	2.1938e+...	2.2637e+...	1.4696e+...	1.1460e+...	-1.2851e...	-2.1924e...	-2.2627e...	-1.4698e...
4	2.2437e+...	1.4632e+...	1.2390e+...	-1.2611e...	-2.1679e...	-2.2437e...	-1.4640e...	-1.2370e...	1.2633e+...	2.1674e+...	2.2435e+...	1.4632e+...	1.2360e+...
5													

Figure 18. MATLAB Script Output -- xWR16xx With DCA1000

Row 1 contains all of the data from the first receiver, row 2 from the second receiver, row 3 from the third receiver, and row 4 from the fourth receiver. In cases where certain receivers are disabled, the corresponding rows will be removed. Each row will contain a number of columns equal to the number of ADC samples per chirp multiplied by the total number of chirps. The columns are organized by chirps. For example, if there are 256 ADC samples and a total of 16 chirps (8 chirps/frame x 2 frames), each row will contain 4096 columns. The first 256 columns correspond to the first chirp, the next 256 columns to the second chirp, and so on. The first 2048 columns correspond to the chirps of the first frame, and so on. The data can then be processed as the user desires.

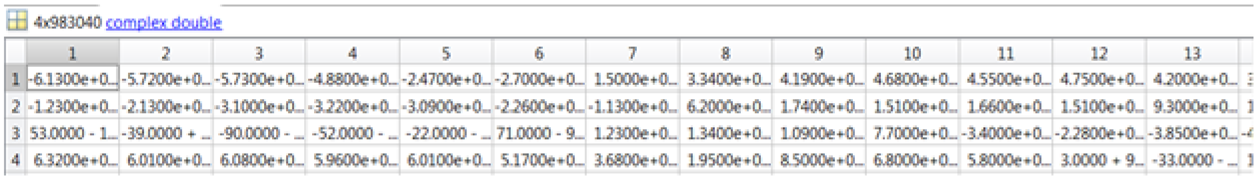
9.3 TSW1400 With xWR12xx and xWR14xx MATLAB Example

The function in the following MATLAB code example can be used to read files saved with the xWR14xx and TSW1400.

```
%%% This script is used to read the binary file produced by the TSW1400
%%% and Mmwave Studio
%%% Command to run in Matlab GUI - readTSW14xx('<ADC capture bin file>')

function [retVal] = readTSW14xx(fileName)
%% global variables

% change based on sensor config
%----------------------------------------------------------------------
numADCBits = 16; % number of ADC bits per sample
numLanes = 4; % do not change. number of lanes is always 4 even if only 1 lane is used. unused
lanes
isReal = 0;  % set to 1 if real only data, 0 if complex dataare populated
% with 0
%----------------------------------------------------------------------

%% read file and convert to signed number

% read .bin file
fid = fopen(fileName,'r');
adcData = fread(fid, 'uint16');
% compensate for offset binary format
adcData = adcData - 2^15;
% if 12 or 14 bits ADC per sample compensate for sign extension
if numADCBits ~= 16
    l_max = 2^(numADCBits-1)-1;
                adcData(adcData > l_max) = adcData(adcData > l_max) - 2^numADCBits;
end
fclose(fid);
%% organize data by LVDS lane

% reshape data based on two samples per LVDS lane
adcData = reshape(adcData, numLanes*2, []);
% for real only data
if isReal
    %each LVDS lane contains two samples from each RX
    rxSample1 = adcData([1,3,5,7],:);
    rxSample2 = adcData([2,4,6,8],:);
    % interleave the first sample set and the second sample set
    adcData = reshape([rxSample1;rxSample2], size(rxSample1,1), []);
%for complex data
else
    % combine real and imaginary parts of complex number
    adcData = adcData([1,3,5,7],:) + sqrt(-1)*adcData([2,4,6,8],:);
end
%% return receiver data

retVal = adcData;
```

The function in the previous code snippet will return a matrix with all of the receiver data stored in rows. The values for numADCBits and isReal should be modified based on the sensor configuration used. The value for numLanes should not be modified as there will always be four LVDS lanes used. Unused lanes will be populated with zeros.

The returned matrix of the function should look similar to Figure 19.


4x983040 complex double

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	-6.1300e+0...	-5.7200e+0...	-5.7300e+0...	-4.8800e+0...	-2.4700e+0...	-2.7000e+0...	1.5000e+0...	3.3400e+0...	4.1900e+0...	4.6800e+0...	4.5500e+0...	4.7500e+0...	4.2000e+0...
2	-1.2300e+0...	-2.1300e+0...	-3.1000e+0...	-3.2200e+0...	-3.0900e+0...	-2.2600e+0...	-1.1300e+0...	6.2000e+0...	1.7400e+0...	1.5100e+0...	1.6600e+0...	1.5100e+0...	9.3000e+0...
3	53.0000 - 1...	-39.0000 + ...	-90.0000 - ...	-52.0000 - ...	-22.0000 - ...	71.0000 - 9...	1.2300e+0...	1.3400e+0...	1.0900e+0...	7.7000e+0...	-3.4000e+0...	-2.2800e+0...	-3.8500e+0...
4	6.3200e+0...	6.0100e+0...	6.0800e+0...	5.9600e+0...	6.0100e+0...	5.1700e+0...	3.6800e+0...	1.9500e+0...	8.5000e+0...	6.8000e+0...	5.8000e+0...	3.0000 + 9...	-33.0000 - ...

Figure 19. MATLAB Script Output -- xWR14xx With TSW1400

In Figure 19, row 1 contains all of the data from the first receiver, row 2 from the second receiver, row 3 from the third receiver, and row 4 from the fourth receiver. In cases where certain receivers are disabled, the corresponding rows will be populated with zeros. Each row will contain a number of columns equal to the number of ADC samples per chirp multiplied by the total number of chirps. The columns are organized by chirps. For example, if there are 256 ADC samples and a total of 10 chirps, each row will contain 2560 columns. The first 256 columns will correspond to the first chirp, the next 256 columns to the second chirp, and so on. The data can then be processed as the user desires.

9.4 *TSW1400 With xWR16xx MATLAB Example*

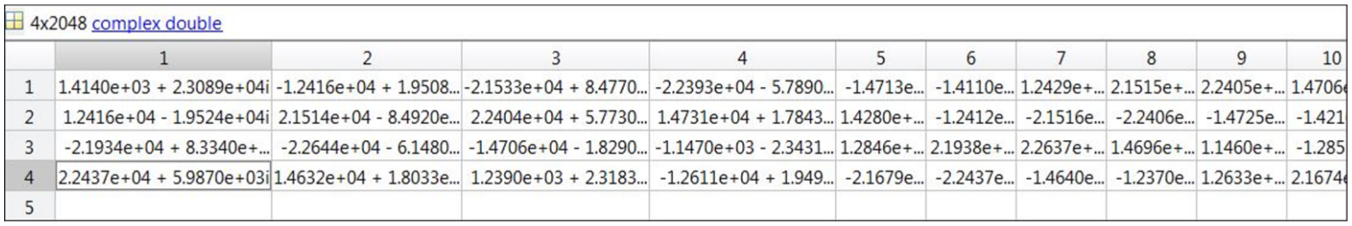
The function in the following MATLAB code example can be used to read files saved with the xWR16xx and TSW1400.

```
%%% This script is used to read the binary file produced by the TSW1400
%%% and Mmwave Studio
%%% Command to run in Matlab GUI - readTSW16xx('<ADC capture bin file>')

function [retVal] = readTSW16xx(fileName)
%% global variables

% change based on sensor config
%----------------------------------------------------------------------
numADCSamples = 256; % number of ADC samples per chirp
numADCBits = 16; % number of ADC bits per sample
numRX = 1; % number of receivers
numLanes = 2; % do not change. number of lanes is always 2
isReal = 1;  % set to 1 if real only data, 0 if complex data0
%-----------------------------------------------------------------------
%% read file

% read .bin file
fid = fopen(fileName,'r');
adcData = fread(fid, 'uint16');
% compensate for offset binary format
adcData = adcData - 2^15;
% if 12 or 14 bits ADC per sample compensate for sign extension
if numADCBits ~= 16
    l_max = 2^(numADCBits-1)-1;
    adcData(adcData > 1_max) = adcData(adcData > 1_max) - 2^numADCBits;
end
fclose(fid);
% get total file
fileSize = size(adcData, 1);
test = adcData;
%% organize data by LVDS lane


% for real data
if isReal
    adcData = reshape(adcData, numLanes*2, []);
    % seperate each LVDS lane into rows
    LVDS = zeros(2, length(adcData(1,:))*2);
    % interleave the two sample sets from each lane
    LVDS(1,1:2:end-1) = adcData(1,:);
```

```
        LVDS(1,2:2:end) = adcData(2,:);
        numChirps = fileSize/2/numADCSamples/numRX;
        if numRX > 1
            LVDS(2,1:2:end-1) = adcData(3,:);
            LVDS(2,2:2:end) = adcData(4,:);
            LVDS = reshape([LVDS(1,:);LVDS(2,:)], size(LVDS(1,:),1), []);
            numChirps = fileSize/2/numADCSamples/numRX;
        end
        LVDS = LVDS(1, :);
    % for complex data
    else
        adcData = reshape(adcData, numLanes, []);
        % seperate each LVDS lane into rows
        LVDS = zeros(1, fileSize/2);
        LVDS(1,:) = adcData(1, :) + sqrt(-1)*adcData(2,:);
        numChirps = fileSize/2/numADCSamples/numRX;
    end

    %% organize data by receiver
    adcData = zeros(numRX,numChirps*numADCSamples);
    LVDS = reshape(LVDS, numADCSamples*numRX, numChirps);
    LVDS = LVDS.';
    for row = 1:numRX
        for i = 1: numChirps
           adcData(row, (i-1)*numADCSamples+1:i*numADCSamples) = LVDS(i, (row-
    1)*numADCSamples+1:row*numADCSamples);
        end
    end
    %% return receiver data

    retVal = adcData;
```

The function in the above code snippet will return a matrix with all of the receiver data stored in rows. The values for numADCSamples, numADCBits, numRX and isReal should be modified based on the sensor configuration used. The value for numLanes should not be modified as there will always be two LVDS lanes used. Unused lanes will be populated with zeros.

The returned matrix of the function should look similar to Figure 20.



4x2048 complex double

	1	2	3	4	5	6	7	8	9	10
1	1.4140e+03 + 2.3089e+04i	-1.2416e+04 + 1.9508...	-2.1533e+04 + 8.4770...	-2.2393e+04 - 5.7890...	-1.4713e...	-1.4110e...	1.2429e+...	2.1515e+...	2.2405e+...	1.4706
2	1.2416e+04 - 1.9524e+04i	2.1514e+04 - 8.4920e...	2.2404e+04 + 5.7730...	1.4731e+04 + 1.7843...	1.4280e+...	-1.2412e...	-2.1516e...	-2.2406e...	-1.4725e...	-1.421
3	-2.1934e+04 + 8.3340e+...	-2.2644e+04 - 6.1480...	-1.4706e+04 - 1.8290...	-1.1470e+03 - 2.3431...	1.2846e+...	2.1938e+...	2.2637e+...	1.4696e+...	1.1460e+...	-1.285
4	2.2437e+04 + 5.9870e+03i	1.4632e+04 + 1.8033e...	1.2390e+03 + 2.3183...	-1.2611e+04 + 1.949...	-2.1679e...	-2.2437e...	-1.4640e...	-1.2370e...	1.2633e+...	2.1674
5										

Figure 20. MATLAB Script Output -- xWR16xx With TSW1400

In Figure 20, row 1 contains all of the data from the first receiver, row 2 from the second receiver, row 3 from the third receiver, and row 4 from the fourth receiver. In cases where certain receivers are disabled, the corresponding rows will be removed. Each row will contain a number of columns equal to the number of ADC samples per chirp multiplied by the total number of chirps. The columns are organized by chirps. For example, if there are 256 ADC samples and a total of 8 chirps, each row will contain 2048 columns. The first 256 columns will correspond to the first chirp, the next 256 columns to the second chirp, and so on. The data can then be processed as the user desires.

10 Summary

This document outlines how to interpret raw ADC data that is captured using the Capture Demo or Mmwave Studio. For data captured using the Capture Demo the capture_demo.m file, provided in the mmWave SDK, can be used and modified to process the data. For data captured in Mmwave Studio, the provided MATLAB code snippet can be used to read the binary file and organize the data into rows by receiver. You can then add to the code to process the data as needed.

11 References

- *DCA1000EVM Data Capture Card User's Guide*
- *TSW140x High Speed Data Capture/Pattern Generator Card*
- *High Speed Data Converter Pro GUI User's Guide*
- *AWR1xxx Data Path Programmer's Guide*
- *Industrial Radar Family Technical Reference Manual*
- mmWave Sensor Raw Data Capture Using the TSW1400 Board
- mmWave Device Firmware Package
- mmWave Software Development Kit
- mmWave Studio

Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from A Revision (August 2018) to B Revision	Page
• Updates were made in Section 5.	7
• Updates were made in Section 6.	9
• Updates were made in Section 7.	11
• Updates were made in Section 8.	13