Using the IWR1443 + DCA1000

Parts:

- IWR1443 (Note there are many newer alternatives to this product which a lot of the code online uses, I think x18 or something, might be worth looking into)
- DCA1000 (While you can get the processed data from the IWR1443 to do anything interesting need this)
- 1x 5V 2A power supply (The IWR1443 can power the DCA1000)

Documents to read

1. IWR1443BOOST Evaluation Module
2. MMWAVE SDK User Guide
    a. Product Release 3.6 LTS
3. DCA1000 Quick Start Guide
4. mmWave Studio User Guide

Required Software

1. **mmWave SDK:** https://www.ti.com/tool/MMWAVE-SDK
    a. Use version 2.1.0 since IWR1443 is an older board
2. **mmWave Studio:** https://www.ti.com/tool/MMWAVE-STUDIO
3. 32-bit Matlab Runtime Engine (Version 8.5.1)
    a. Be careful it's exactly this version or else the **Radar API** won't appear in mmWave studio
4. Microsoft Visual C++ 2013

Flashing the board

1. Ensure IWR1443 is powered off
2. Put jumpers on SOP2 and SOP0 to put it in **Flash programming** mode
3. Power up the board
4. Launch Uniflash
5. For the *Meta image 1* select your desired binary file
    a. For the demo it should be at:
    mmwave_sdk_<ver>\ti\demo\<platform>\mmw\<platform>_mmw_demo.bin
    b. Note you'll first need to install the SDK see Install mmWave SDK
6. Click load image

Verifying the IWR1443 with Demo Visualizer

1. Go to Demo Visualizer
2. Ensure only SOP0 has a jumper (i.e. put it in **functional mode** see pg14 of [1])
    a. See https://dev.ti.com/tirex/explore/node?node=A__AXAenV2u4woV.FhTlAk68Q__radar_toolbox__1AslXXD__LATEST
3. For visualizer settings:
    a. On launch pop-up will ask to redirect to SDK 2.1
    b. Select **xWR14xx** for platform
    c. Options > Serial Port and select your serial ports (lower port number is the **CFG_Port**)
    d. Click “SEND CONFIG TO MMWAVE DEVICE”
    e. Go to plots tab

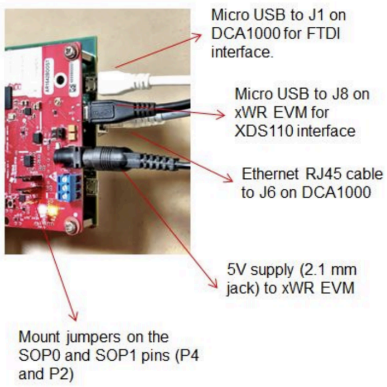
Note: This requires the board to have the demo firmware. See Flashing

For further details see *pg 9 [2]*

Aside: On **pg 8 [2]** under **mmWave demo with LVDS-based instrumentation** it talks about sending what I think is the raw data to the DCA1000 from the Demo application. We didn’t have time to investigate but I believe this would remove the need to have mmWave Studio at all involved and might simplify the options of configuring the board.

Getting raw data from DCA1000 with mmWave Studio

1. Ensure jumpers on SOP0 and SOP1 (i.e. put it into Dev mode)
2. Give your computer a static IP address of 192.168.33.30
    a. The DCA1000 is configured to send data by default to this IP address
    b. **Note:** We weren’t able to get this working on macOS only Windows
3. Start mmWave Studio in Administrator mode
4. Ensure all the wires are connected should be 1x Ethernet and 3x micro-USB



5. Go to the **Radar API** tab
6. Click **Set**
7. For the COM port select the **CFG** port (generally the lower number)
8. Click connect

Running the Demo application

For verification you can run a test script.

1. Under **Browse** in the bottom-right of the **Radar API** tab you can navigate to *ti/mmwave_studio_<ver>/mmWaveStudio/Scripts* and find **DataCaptureDemo_xWR**
2. Click start, change to the **Output** tab to see progress
3. Will open a new window with the results
4. Raw results are saved to **adc_data.bin**

Note: The script by default uses a test source which is why it will always look the same, you can disable this in the lua file

See **[4]** for more details and debugging steps

MSS Power Up async event

If you get the error:

```
[10:35:50]  Status: Failed, Error Type: RESP TIMEOUT
[10:35:55]  MSS Power Up async event was not received!
```

You first should use **Uniflash** to format your device. I believe this happens when the Demo has been flashed and then you try use mmWave Studio

Lua API

The DCA1000 is configured using LUA scripts. The documentation for the various commands can be found by going to the **Shell** tab of mmWave Studio and running

```
help ar1.DataPathConfig
```

For example

```
## DataPathConfig

```lua
>help ar1.DataPathConfig
Int32 ar1.DataPathConfig(UInt32 intfSel, UInt32 transferFmtPkt0, UInt32
transferFmtPkt1) - DataPathConfig API Defines the used to configure the device data
path
_I_ UInt32  intfSel  - Data path interface select(0:7)+ CQ config(b8:15)
_I_ UInt32  transferFmtPkt0  - Data output format(b0:7)+ CQ0TransSize(b8:15)+
CQ1TransSize(b16:23)+ CQ2TransSize(b24:31)
_I_ UInt32  transferFmtPkt1  - Supress packet 1 transmission
```
```

DCA1000 Data format

The IWR1443 and the DCA1000 provide complex-valued samples of the IF signal which is the result of mixing the RX and TX signal and putting it through a lower-pass filter i.e. it's the beat frequency.

The data is formatted as so (from 24.6 of **[4]**)

24.6 ***DCA1000 EVM capture format (xWR12xx/xWR14xx complex, 4 channel, 4 lanes [Interleaved])***

Chirp 1	Rx0I0	Rx1I0	Rx2I0	Rx3I0	Rx0Q0	Rx1Q0	Rx2Q0	Rx3Q0
	Rx0I1	Rx1I1	Rx2I1	Rx3I1	Rx0Q1	Rx1Q1	Rx2Q1	Rx3Q1
	…		…		…		…	…
	Rx0IN-1	Rx1IN-1	Rx2IN-1	Rx3IN-1	Rx0QN-1	Rx1QN-1	Rx2QN-1	Rx3QN-1
Chirp 2	Rx0I0	Rx1I0	Rx2I0	Rx3I0	Rx0Q0	Rx1Q0	Rx2Q0	Rx3Q0
	Rx0I1	Rx1I1	Rx2I1	Rx3I1	Rx0Q1	Rx1Q1	Rx2Q1	Rx3Q1
	…		…		…		…	…
	Rx0IN-1	Rx1IN-1	Rx2IN-1	Rx3IN-1	Rx0QN-1	Rx1QN-1	Rx2QN-1	Rx3QN-1

S

Related to our config.lua file (see below) this frame consists of every **PERIODICITY ms** we transmit **CHRIP_LOOPS** chirps. Each chirp linearly sweeps from **START_FREQ** onwards. We capture
ADC_SAMPLES samples during this chirp across **NUM_RX** RX antennas.

Note that there is interleaving if you use more than one TX antenna, we were not sure how this works but it's included in the code. If you use multiple antennas is decided by:

```
START_CHIRP_TX = 0
END_CHIRP_TX = 0 -- 2 for 1843
```

In the lua config, since both of ours were 0 we only used 1 TX antenna.

Interfacing with Python

You can open up a socket to the DCA1000 and receive the data over that, however you'll need to configure the DCA1000 correctly (put it into continuous mode, correct # of LVDS channels) so we'd
recommend just using our code implementation at:
https://github.com/r-bt/MakeyMakey/tree/main/src/xwr.

Note that you also need to run the lua config scripts to setup the DCA1000 located at:
https://github.com/r-bt/MakeyMakey/blob/main/scripts/1443_mmwavestudio_config_old.lua

Our code was based off the work at
https://github.com/ConnectedSystemsLab/xwr_raw_ros/tree/main/src/xwr_raw but modified to work with the IWR1443 since they used other boards which have different data formats.

See **xwr/dsp.py** for how we shape the frames

On a high level this works as follows:

1. We parse the config.lua file used to configure the DCA1000 to get the number of adc samples, frame size, etc
2. We open up a socket to the DCA1000 and create a frame buffer to store the data in
3. We continuously read from the DCA1000 and append to the frame buffer, whenever we have enough data we return the frame
4. We shape the frame so that it's the shape (n_chirps_per_frame, n_samples_per_chrip, n_receivers)
5. We process the frames

Side notes

1. The Matlab package for TI radars doesn't work with the IWR1443
2. We found that in continuous mode the DCA1000 would just stop sometimes – it wasn't consistent so we think it was to do with our python code. Would be interesting to know why and fix this