

Detailed Explanation of Frequency Hopping Configuration Based on Look-Up Table on AWRL1432



Ken Zhang

Central mmWave FAE

Abstract

This document was translated from a simplified Chinese source. (ZHCAGB2)

The AWRL1432 radar sensor is a low-cost and ultra-low-power product solution launched by TI. Benefiting from the excellent low-power design, the average power consumption of the AWRL1432 can be as low as the milliwatt level. Its RF configuration APIs are also more concise and streamlined, allowing developers to perform flexible custom configuration of RF using the look-up table method (Per-Chirp LUT). This paper details various waveform configuration methods of this product and provides a frequency hopping instance implemented based on the look-up table method, helping developers utilize the look-up table method to enhance and optimize the system parameter metrics of the radar.

Table of Contents

1 Introduction.....	2
2 Frequency Hopping Modulation of Millimeter-Wave Radar.....	3
3 Waveform Configuration Flow of AWRL1432.....	5
3.1 Conventional Configuration Method of AWRL1432.....	5
3.2 Look-Up Table Configuration Method of AWRL1432.....	5
3.3 Detailed Explanation of Look-Up Table RF APIs.....	6
4 Implementation of Frequency Hopping on AWRL1432.....	8
5 Verification of Frequency Hopping Results.....	10
5.1 Preparation Work Before Data Capture.....	10
5.2 Data Acquisition Using DCA1000EVM with mmw_demo.....	10
5.3 Data Post-Processing.....	11
6 Conclusion.....	13
7 References.....	13

List of Figures

Figure 2-1. Waveform Differences between FMCW and SFCW.....	3
Figure 2-2. Range-Doppler Heatmap Differences between FMCW and SFCW.....	4
Figure 5-1. Connection Diagram of AWRL1432 and DCA1000.....	11
Figure 5-2. Software Interface of rawDataCaptureGUI_DCA1000CLI.exe.....	11
Figure 5-3. Comparison of Range-Doppler Heatmaps Before and After Enabling Look-Up Table Frequency Hopping.....	12

List of Tables

Table 3-1. Parameter Types in AWRL1432 LUT.....	5
Table 3-2. AWRL1432 Look-Up Table Related RF APIs.....	6
Table 3-3. Meanings of h_ParamArrayLen Parameters.....	6
Table 3-4. Meanings of h_ParamRptCount Parameters.....	6
Table 3-5. Meanings of h_PerChirpParamCtrl Parameters.....	7
Table 3-6. Meanings of h_ParamArrayStartAdd Parameters.....	7
Table 4-1. LUT Values Corresponding to Frequency Hopping.....	8

Trademarks

All trademarks are the property of their respective owners.

1 Introduction

AWRL1432 is a 77GHz ultra-low-cost, ultra-low-power single-chip millimeter-wave radar SoC launched by TI, suitable for various applications such as blind spot detection (BSD) radar, two-wheeler rear warning radar, and hands-free trunk opener radar. The radar sensor uses FMCW (Frequency Modulated Continuous Wave) for RF modulation. The chirp and frame waveform parameters need to be adjusted accordingly based on different terminal applications. RF configuration determines the range, velocity, and angle metrics of the sensor; therefore, the setting of waveform parameters is crucial to the detection capability of the radar.

The built-in Radar Front-End Control System (FECSS) of the AWRL1432 can provide precise and real-time RF control, and opens up rich interface functions (APIs) to developers, helping them fully unleash the potential of the radar, flexibly configure and adjust various parameters, and cope with various design challenges. Flexible interface functions also help implement advanced modulation strategies to further enhance the actual performance of the radar. For example, the frequency hopping strategy helps increase the actual transmission bandwidth, achieving excellent range resolution.

2 Frequency Hopping Modulation of Millimeter-Wave Radar

Millimeter-wave radar sensors often use FMCW (Frequency Modulated Continuous Wave) for modulation. A typical radar signal frame consists of multiple continuous and repetitive chirps; therefore, the range resolution Δd is determined by the bandwidth B of a single chirp. However, the sweep slope and sweep duration of a single chirp are limited, the value of bandwidth B is relatively small, and the theoretically calculated range resolution will also be relatively ambiguous. If the bandwidth of a single chirp is simply increased to improve range resolution, the chirp duration T_c will increase, and the measurable maximum velocity metric will also decrease.

To expand the effective bandwidth, users can gradually accumulate a fixed frequency offset Δf on the sweep start frequency of each chirp. Although the bandwidth of each chirp remains unchanged, the equivalent bandwidth of the entire frame is significantly stretched to $B+n\Delta f$. This strategy is called Stepped Frequency Continuous Wave (SFCW). The differences between SFCW and FMCW are shown in Figure 2-1. While increasing the bandwidth, the effective period T_c of the chirp does not change.

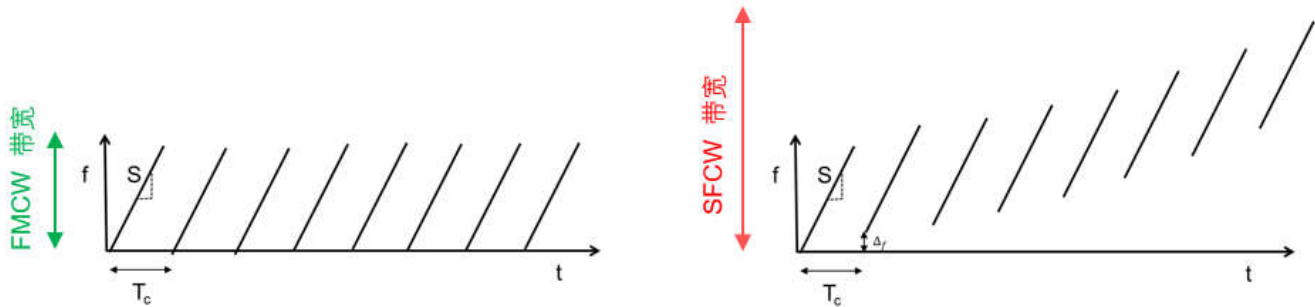


Figure 2-1. Waveform Differences between FMCW and SFCW

The frequency hopping strategy SFCW will cause the target to generate an offset along the Doppler direction on the Range-Doppler heatmap. The offset distance is related to the frequency hopping frequency Δf and the target distance R . When the offset is too large, folding may occur in the Doppler dimension, while the target position (Index) in the range direction remains unchanged. During subsequent processing, additional processing steps and processing time are required to correct this offset. Figure 2-2 This illustrates the Range-Doppler map differences between FMCW and SFCW. From the heatmap, we can not only intuitively see the improvement in range resolution brought by SFCW, but the Doppler-direction offset and folding brought by frequency hopping are also displayed in the figure.

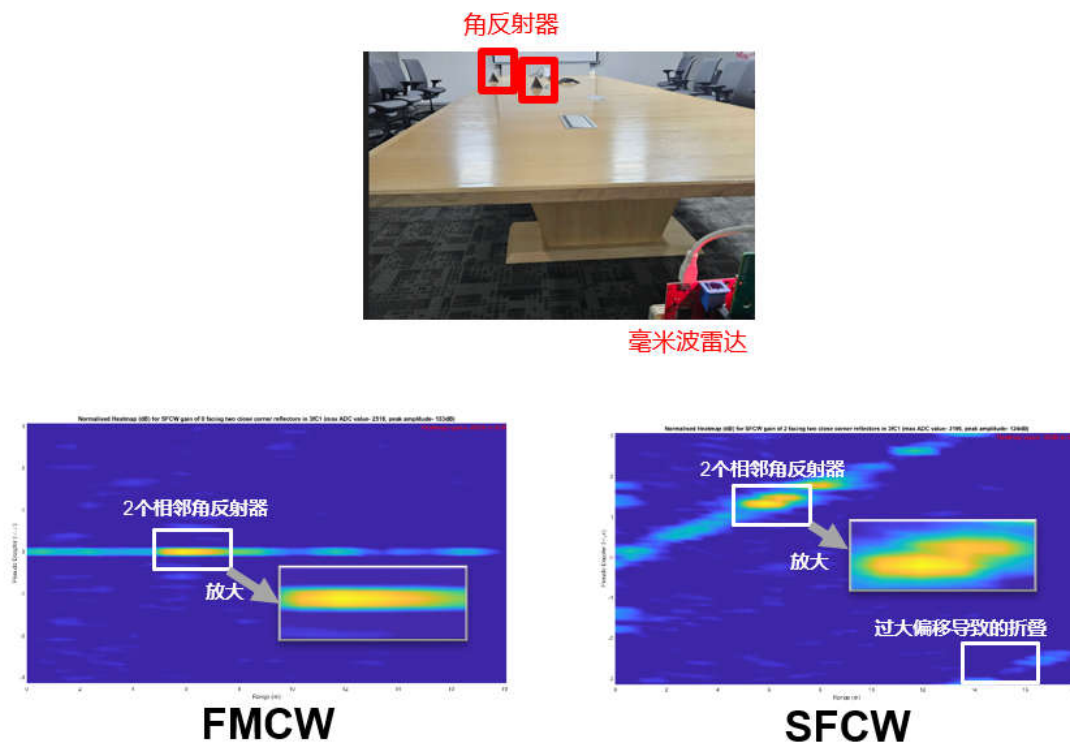


Figure 2-2. Range-Doppler Heatmap Differences between FMCW and SFCW

3 Waveform Configuration Flow of AWRL1432

3.1 Conventional Configuration Method of AWRL1432

Unlike other millimeter-wave products of TI, there is only one profile template inside the AWRL1432, and all chirps share this global template. This means that except for the differences introduced by MIMO modulation, there are no other differences between chirps. First, set the general parameters of chirp (such as sampling rate, number of sampling points, TDM (Time Division Multiplexing) / BPM (Binary Phase Modulation) modulation selection, high-pass filter selection, etc.) and sampling timing parameters (such as sweep slope, antenna enable, idle time, start frequency, etc.) through ① `rl_sensChirpProfComnCfg` and ② `rl_sensChirpProfTimeCfg` respectively. After completing the definition of chirp, continue to call ③ `rl_sensFrameCfg` to set the number of chirps in a burst, the period of the burst, the number of bursts in a frame, the period of the frame, etc., and finally call ④ `rl_sensSensorStart` to start wave transmission.

The conventional configuration method only requires three APIs to complete the RF definition, which is sufficient to meet the sensing requirements of most scenarios. For more complex configuration requirements, such as frequency hopping and DDMA, the AWRL1432 supports a more advanced look-up table method (Per-Chirp LUT). The internal look-up table mechanism can flexibly adjust and change various parameters of each chirp. However, due to hardware limitations, the internal phase shifter only supports 0° and 180° phase shifting, which needs to be considered when designing waveforms.

3.2 Look-Up Table Configuration Method of AWRL1432

For more complex configuration requirements, such as frequency hopping and DDMA, the AWRL/xWRLx432 also has a look-up table mechanism similar to Advance Chirp in the AWR2944 and AWR2243, called Per-Chirp LUT, to support independent control and modification of various parameters for each chirp. Users can refer to the `mmwave_dfp_interface_control_document` document in the `MMWAVE_L_SDK` for more detailed information.

Inside the AWRL1432, 8KB of `PER_CHIRP_RAM` is provided for storing the look-up table (LUT) used by the look-up table method, and its memory start address is `0x21880000`. When the AWRL1432 is not transmitting waves, user programs can directly read and write to access this memory address range. Before transmitting waves, the user writes the required waveform parameters into `PER_CHIRP_RAM` consecutively after 4-byte alignment according to a specific format. After wave transmission starts, the parameters in the LUT will overwrite and rewrite the corresponding parameters in the global profile. Note that unlike the advance chirp LUT table of the AWR2243, the chirp parameters here are absolute values rather than change amounts based on the profile. The chirp parameters that can be changed through the LUT are summarized in [Table 3-1](#). The data in `PER_CHIRP_RAM` will still remain during the deep sleep stage of the AWRL1432, but all APIs mentioned in [Table 3-2](#) below need to be called again.

Table 3-1. Parameter Types in AWRL1432 LUT

Serial Number	Parameter Name	Meaning	Width of LUT Unit	1 LSB Meaning
0	Start Frequency	Start Frequency	Unsigned 32-bit	High-resolution mode 6.103kHz Low-resolution mode 1.562MHz
1	Slope	Sweep Slope	Unsigned 16-bit	38.147kHz/us
2	Idle Time	Idle Time	Unsigned 16-bit	High-resolution mode 20us Low-resolution mode 100us
3	ADC Start Time	Sampling Start Time	Unsigned 16-bit	Skipped ADC Sampling Points
4	Tx Start Time	Tx Start Time	Signed 16-bit	20us
5	Tx Enable Mask	Tx Antenna Enable	4 bits	Each bit represents a transmit antenna
6	Tx BPM Enable Mask	Tx BPM Enable	4 bits	Each bit represents a transmit antenna

The APIs using the look-up table method are summarized in Table 2. First, according to the steps in Section 3.1, call ① `rl_sensChirpProfComnCfg`, ② `rl_sensChirpProfTimeCfg`, and ③ `rl_sensFrameCfg` in turn to specify the global profile template and frame timing. In addition, it is necessary to additionally call ④ `rl_sensPerChirpCfg` and ⑤ `rl_sensPerChirpCtrl`, and finally call ⑥ `rl_sensSensorStart` to start wave transmission.

Table 3-2. AWRL1432 Look-Up Table Related RF APIs

Step	API Name	Description
1	rl_sensChirpProfComnCfg	Configure the basic parameter template of chirp
2	rl_sensChirpProfTimeCfg	Configure the timing parameters of chirp
3	rl_sensFrameCfg	Configure the basic parameters of the frame, such as period, number of chirps, etc.
4	rl_sensPerChirpCfg	Configure the number of LUT units and the repetition count of each unit
5	rl_sensPerChirpCtrl	Configure which chirp parameters enable look-up tables and the offset address during look-up
6	rl_sensSensorStart	Start wave transmission

3.3 Detailed Explanation of Look-Up Table RF APIs

rl_sensPerChirpCfg is used to configure the number of LUT units and the repetition count for each unit.

Its input parameters include UINT16 h_ParamArrayLen[12] and UINT16 h_ParamRptCount[12]. Every two bytes in h_ParamArrayLen (i.e., one element of h_ParamArrayLen) specify the number of LUT units corresponding to each RF parameter in PER_CHIRP_RAM, with the specific meanings shown in Table 3-3. Every two bytes in h_ParamRptCount (i.e., one element of h_ParamRptCount) specify the repetition count of each LUT unit, that is, how many consecutive chirps will adopt the same LUT unit, with the specific meanings shown in Table 3-4.

Table 3-3. Meanings of h_ParamArrayLen Parameters

Bytes	[1:0]	[3:2]	[5:4]	[7:6]
Descripti on	Start Frequency Number of LUT Units	Sweep Slope Number of LUT Units	Idle Time Number of LUT Units	Sampling Start Time Number of LUT Units
Bytes	[9:8]	[11:10]	[13:12]	[23:14]
Descripti on	Tx Start Time Number of LUT Units	Tx Enable Mask Number of LUT Units	Tx BPM Enable Number of LUT Units	Reserved Padding

Table 3-4. Meanings of h_ParamRptCount Parameters

Bytes	[1:0]	[3:2]	[5:4]	[7:6]
Descripti on	Start Frequency Repetition Count per LUT Unit	Sweep Slope Repetition Count per LUT Unit	Idle Time Repetition Count per LUT Unit	Sampling Start Time Repetition Count per LUT Unit
Bytes	[9:8]	[11:10]	[13:12]	[23:14]
Descripti on	Tx Start Time Repetition Count per LUT Unit	Tx Enable Mask Repetition Count per LUT Unit	Tx BPM Enable Repetition Count per LUT Unit	Reserved Padding

Assuming that four LUT units A, B, C, and D for a certain RF parameter are pre-written into PER_CHIRP_RAM, and the number of LUT units for this RF parameter in h_ParamArrayLen is set to 4, the repetition count of this RF parameter in h_ParamRptCount is set to 2, and the number of chirps in a single frame is 16, then in the final actual transmission of one frame, the transmission sequence for this RF parameter is A, A, B, B, C, C, D, D, A, A, B, B, C, C, D, D. When h_ParamArrayLen is 2 and h_ParamRptCount is 4, the transmission sequence for this RF parameter is A, A, A, A, B, B, B, B, A, A, A, A, B, B, B, B.

rl_sensPerChirpCtrl configures which chirp parameters enable look-up tables and the offset addresses during look-up.

The main input parameters include UINT16 h_PerChirpParamCtrl and UINT16 h_ParamArrayStartAdd[12]. Each bit in h_PerChirpParamCtrl represents whether a certain chirp RF parameter enables the look-up table method. A bit of 0 represents disabled, and a bit of 1 represents enabled. By default, the look-up table enable status is disabled. See Table 3-5 for the specific meaning of each bit. Every two bytes in h_ParamArrayStartAdd (i.e., one h_ParamArrayStartAdd element) specify the offset address of the LUT table for various RF front-end parameters in PER_CHIRP_RAM, with the specific meanings shown in Table 3-6. In actual operation, attention should be paid to 4-byte address alignment and avoiding overlap between LUT tables of different parameters.

Table 3-5. Meanings of h_PerChirpParamCtrl Parameters

Bit	[0]	[1]	[2]	[3]
Descripti on	Start Frequency Look-Up Table Enable	Sweep Slope Look-Up Table Enable	Idle Time Look-Up Table Enable	Sampling Start Time Look-Up Table Enable
Bit	[4]	[5]	[6]	[15:7]
Descripti on	Tx Start Time Look-Up Table Enable	Tx Enable Mask Look-Up Table Enable	Tx BPM Enable Look-Up Table Enable	Reserved Padding

Table 3-6. Meanings of h_ParamArrayStartAdd Parameters

Bytes	[1:0]	[3:2]	[5:4]	[7:6]
Descripti on	Start Frequency Offset Address in LUT	Sweep Slope Offset Address in LUT	Idle Time Offset Address in LUT	Sampling Start Time Offset Address in LUT
Bytes	[9:8]	[11:10]	[13:12]	[23:14]
Descripti on	Tx Start Time Offset Address in LUT	Tx Enable Mask Offset Address in LUT	Tx BPM Enable Offset Address in LUT	Reserved Padding

4 Implementation of Frequency Hopping on AWRL1432

Frequency hopping essentially means that the sweep start frequency of each chirp accumulates upward step by step, and the start frequency of each chirp is different. Therefore, on the AWRL1432, the start frequency of each chirp requires a LUT unit.

Assuming there are a total of 64 chirps in each frame, the start frequency of the first chirp is 76GHz, and each chirp increments by 15.62MHz, then we need to pre-write 64 start frequency LUT units into PER_CHIRP_RAM. Except for the start frequency, the other RF parameters of these 64 chirps are completely identical. Therefore, other parameters do not need to be written into PER_CHIRP_RAM, nor is it necessary to enable the look-up table method.

At SDK startup, the default chirp time resolution is the low-resolution mode (configured by the input parameter `c_ChirpTimerResol` of the `rl_fecssDevPwrOn` function), and each LSB of the start frequency LUT unit represents 1.562MHz. From this, we can derive the contents of the LUT, as shown in Table 4-1. Since Δf cannot be perfectly divided by the LSB resolution, this leads to a certain error between the actual frequency hopping frequency and the theoretical frequency hopping frequency. Users can further optimize the LUT values based on the following table to control the error within 6.103kHz. After calculating and acquiring all contents, we can directly modify the values of PER_CHIRP_RAM using pointers.

Table 4-1. LUT Values Corresponding to Frequency Hopping

Chirp ID	0	1	2	...	63
Start Frequency	76GHz	76.015GHz	76.031GHz	...	76.984GHz
Theoretical Δf	0MHz	15.62MHz	31.24MHz	...	984.06MHz
LUT Value (HEX)	0x00BE00U	0x00BE0A	0x00BE14	...	0x00C076

Given that the number of chirps in a frame is 64 and 64 different chirp start frequencies are required, the LUT needs to be updated whenever a new chirp is transmitted. Thus, we can confirm that the number of start frequency LUT units in `h_ParamArrayLen` is 64 and the repetition count of a single start frequency LUT unit in `h_ParamRptCount` is 1. Furthermore, given that only the start frequency look-up table method needs to be enabled, and the storage address offset of the start frequency LUT relative to PER_CHIRP_RAM is 0, we also obtain that the start frequency look-up table enable in `h_PerChirpParamCtrl` is set to 1 and the offset address of the start frequency LUT in `h_ParamArrayStartAdd` is 0.

Based on the above analysis, we can write the following code, add it to the function `MMWave_startLink`, and then recompile the corresponding library files and application files, thereby implementing the frequency hopping mechanism on the AWRL1432. In the following code, although other RF parameters are also configured, only the lowest bit of `h_PerChirpParamCtrl` is set to 1; therefore, the look-up table method for other RF parameters does not take effect.

```
typedef struct
{
    UINT32 w_StartFreqHighRes[64U];
    SINT16 xh_ChirpSlope[4U];
    UINT16 h_ChirpIdleTime[4U];
    UINT16 h_ChirpAdcStartTime[4U];
    SINT16 xh_ChirpTxStartTime[4U];
    UINT8 c_ChirpTxEn[4U];
    UINT8 c_ChirpBpmEn[4U];
} T_SENS_PER_CHIRP_LUT;
T_SENS_PER_CHIRP_LUT *p_sensPerChirpLUT = \
    (T_SENS_PER_CHIRP_LUT*)M_RL_SENS_PER_CHIRP_LUT_HW_START_ADDRESS;

UINT8 k_lutIndex = 0U;
for(k_lutIndex=0; k_lutIndex<64; k_lutIndex++)
{
    p_sensPerChirpLUT->w_StartFreqHighRes[k_lutIndex] = \
        M_RL_SENS_CHIRP_RFFREQ_LR_76G + k_lutIndex* 10U;//frequency step 15.62MHz for each chirp
}
T_RL_API_SENS_PER_CHIRP_CFG z_sensPerChirpCfg = {
    .h_ParamArrayLen[M_RL_SENS_PER_CHIRP_FREQ_START] = 64U,
    .h_ParamArrayLen[M_RL_SENS_PER_CHIRP_FREQ_SLOPE] = 4U,
    .h_ParamArrayLen[M_RL_SENS_PER_CHIRP_IDLE_TIME] = 4U,
```

```

        .h_ParamArrayLen[M_RL_SENS_PER_CHIRP_ADC_START_TIME] = 4U,
        .h_ParamArrayLen[M_RL_SENS_PER_CHIRP_TX_START_TIME] = 4U,
        .h_ParamArrayLen[M_RL_SENS_PER_CHIRP_TX_ENABLE] = 4U,
        .h_ParamArrayLen[M_RL_SENS_PER_CHIRP_BPM_ENABLE] = 4U,
        .h_ParamRptCount[M_RL_SENS_PER_CHIRP_FREQ_START] = 1U,
        .h_ParamRptCount[M_RL_SENS_PER_CHIRP_FREQ_SLOPE] = 1U,
        .h_ParamRptCount[M_RL_SENS_PER_CHIRP_IDLE_TIME] = 1U,
        .h_ParamRptCount[M_RL_SENS_PER_CHIRP_ADC_START_TIME] = 1U,
        .h_ParamRptCount[M_RL_SENS_PER_CHIRP_TX_START_TIME] = 1U,
        .h_ParamRptCount[M_RL_SENS_PER_CHIRP_TX_ENABLE] = 1U,
        .h_ParamRptCount[M_RL_SENS_PER_CHIRP_BPM_ENABLE] = 1U
    };
    T_RL_API_SENS_PER_CHIRP_CTRL z_sensPerChirpCtrl = {
        .h_PerChirpParamCtrl = 0x1, //only enable start
frequency LUT update
        .h_ParamArrayStartAdd[M_RL_SENS_PER_CHIRP_FREQ_START] =
            ((UINT32)(&p_sensPerChirpLUT->w_StartFreqHighRes[0]) &
M_RL_SENS_PER_CHIRP_LUT_ADD_MASK),
        .h_ParamArrayStartAdd[M_RL_SENS_PER_CHIRP_FREQ_SLOPE] =
            ((UINT32)(&p_sensPerChirpLUT->xh_ChirpSlope[0]) & M_RL_SENS_PER_CHIRP_LUT_ADD_MASK),
        .h_ParamArrayStartAdd[M_RL_SENS_PER_CHIRP_IDLE_TIME] =
            ((UINT32)(&p_sensPerChirpLUT->h_ChirpIdleTime[0]) & M_RL_SENS_PER_CHIRP_LUT_ADD_MASK),
        .h_ParamArrayStartAdd[M_RL_SENS_PER_CHIRP_ADC_START_TIME] =
            ((UINT32)(&p_sensPerChirpLUT->h_ChirpAdcStartTime[0]) &
M_RL_SENS_PER_CHIRP_LUT_ADD_MASK),
        .h_ParamArrayStartAdd[M_RL_SENS_PER_CHIRP_TX_START_TIME] =
            ((UINT32)(&p_sensPerChirpLUT->xh_ChirpTxStartTime[0]) &
M_RL_SENS_PER_CHIRP_LUT_ADD_MASK),
        .h_ParamArrayStartAdd[M_RL_SENS_PER_CHIRP_TX_ENABLE] =
            ((UINT32)(&p_sensPerChirpLUT->c_ChirpTxEn[0]) & M_RL_SENS_PER_CHIRP_LUT_ADD_MASK),
        .h_ParamArrayStartAdd[M_RL_SENS_PER_CHIRP_BPM_ENABLE] =
            ((UINT32)(&p_sensPerChirpLUT->c_ChirpBpmEn[0]) & M_RL_SENS_PER_CHIRP_LUT_ADD_MASK)
    };
    int32_t retVal;
    retVal = rl_sensPerChirpCfg(M_DFP_DEVICE_INDEX_0, &z_sensPerChirpCfg);
    retVal = rl_sensPerChirpCtrl(M_DFP_DEVICE_INDEX_0, &z_sensPerChirpCtrl);

```

5 Verification of Frequency Hopping Results

To verify the frequency hopping performance, this paper acquires the raw ADC data of the AWRL1432 sensor via the DCA1000EVM. The experiment implements data acquisition based on the mmw_demo sample program in the mmWave SDK, and utilizes MATLAB algorithms to perform Range-Doppler processing on the acquired data, quantitatively analyzing and verifying the frequency hopping results through the generated Range-Doppler heatmaps.

5.1 Preparation Work Before Data Capture

Flash the updated mmw_demo appimage into the AWRL1432BOOST. This paper uses radar_toolbox_4_00_00_05 and MMWAVE_L_SDK_05_05_04_02 for subsequent testing work. During the data acquisition process, we still need to rely on the serial CLI of mmw_demo to issue the cfg file. After receiving the CLI command issued by the serial port, mmw_demo will normally execute RF initialization and other necessary RF APIs, such as rl_sensChirpProfComnCfG.

An example of a cfg file verified through DCA1000 data capture tests is shown below. To adapt to data acquisition, the main modifications include: limiting the number of frames to 10 frames in frameCfG; enabling LVDS output of raw ADC data in adcLogging; disabling the low-power mode in lowPowerCfG to prevent data loss during acquisition; and seeing from chirpComnCfG that TDM modulation is used. Before testing, we also need to modify the CLI_MMWaveChirpCommonCfG function to prevent the CLI from abnormally exiting after detecting the use of TDM modulation (gMmwMssMCB.profileComnCfG.c_ChirpTxMimoPatSel == 1).

```
sensorStop 0
channelCfG 7 3 0
chirpComnCfG 20 0 0 128 1 33 2
chirpTimingCfG 6 34 0.00 117.1875 76.5
frameCfG 2 0 1200 32 80 10
antGeometryCfG 0 0 1 1 0 2 0 1 1 2 0 3
guiMonitor 2 1 0 0 0 1 0 0 0 0 0
sigProcChainCfG 64 32 1 0 0 0 0 0
sensorPosition 0 0 0.3 0 0
cfarCfG 2 8 4 3 0 12.0 0 0.95 0 1 1 1
aoaFovCfG -80 80 -20 20
rangeSelCfG 0.1 10.0
clutterRemoval 0
compRangeBiasAndRxChanPhase 0.0 1.00000 0.00000 -1.00000 0.00000 1.00000 0.00000 -1.00000 0.00000
1.00000 0.00000 -1.00000 0.00000
adcLogging 1
lowPowerCfG 0
factoryCalibCfG 0 0 40 0 0x1ff000
sensorStart 0 0 0 0
```

5.2 Data Acquisition Using DCA1000EVM with mmw_demo

Connect the DCA1000EVM, computer, and AWRL1432BOOST as shown in [Figure 5-1](#), configure the computer's IP address to 192.168.33.30, and then run radar_toolbox_4_00_00_05\tools\Adc_Data_Capture_Tool_DCA1000_CLI\rawDataCaptureGUI_DCA1000CLI.exe. As shown in [Figure 5-2](#), select the serial port number, cfg file, capture file path, capture duration, etc., in the GUI interface, and click Go to start data capture. The software will automatically issue the cfg file and configure the DCA1000EVM for data capture. Upon completion of data capture, the bin file is stored in the data folder under the current path.

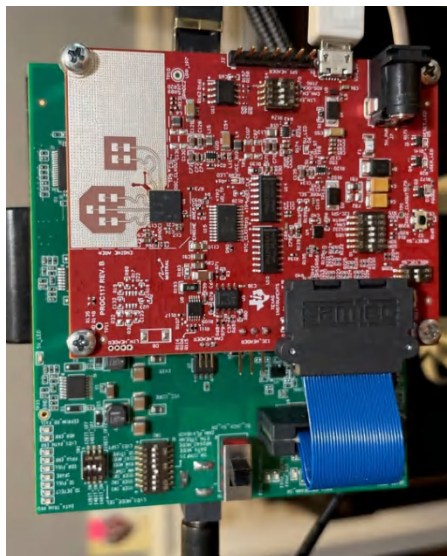


Figure 5-1. Connection Diagram of AWRL1432 and DCA1000

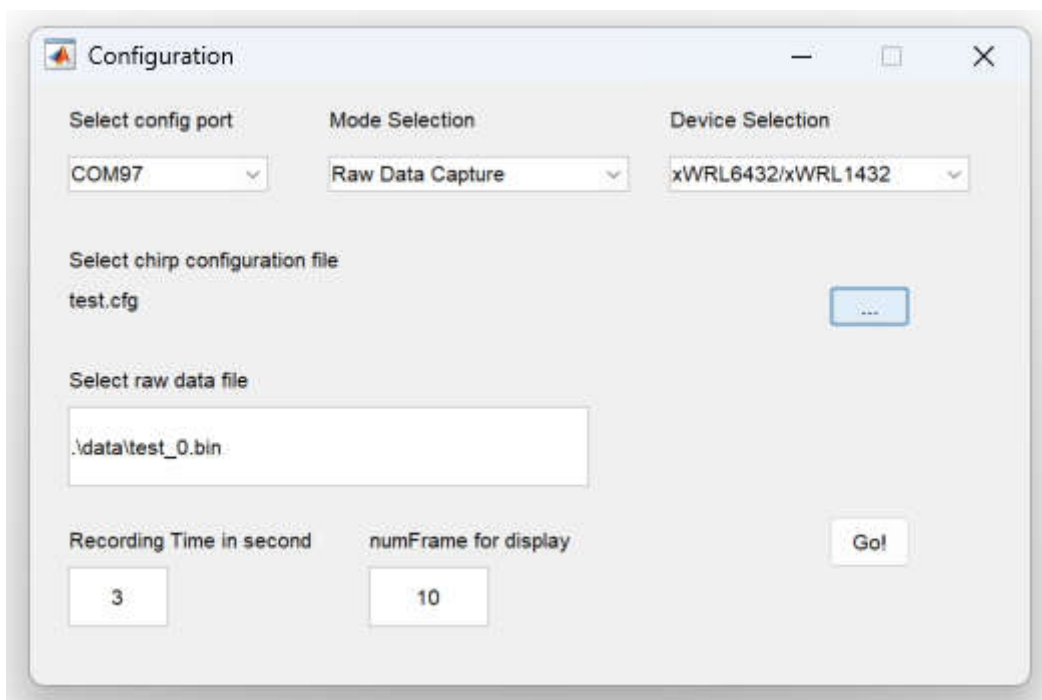


Figure 5-2. Software Interface of rawDataCaptureGUI_DCA1000CLI.exe

5.3 Data Post-Processing

After acquiring the raw ADC data, users can refer to the MATLAB code in MMWAVE_L_SDK_05_05_04_02\tools\ADC_parser for data post-processing. After performing 2DFFT calculation and processing on the data, the Range-Doppler heatmap is obtained. As shown in [Figure 5-3](#), they are the heatmaps before (left) and after (right) enabling the look-up table frequency hopping. After enabling frequency hopping, it can be seen in the figure that the stationary target in the middle of the image generates an offset in the Doppler dimension. This characteristic tilt verifies that we have successfully enabled frequency hopping on the AWRL1432 chip.

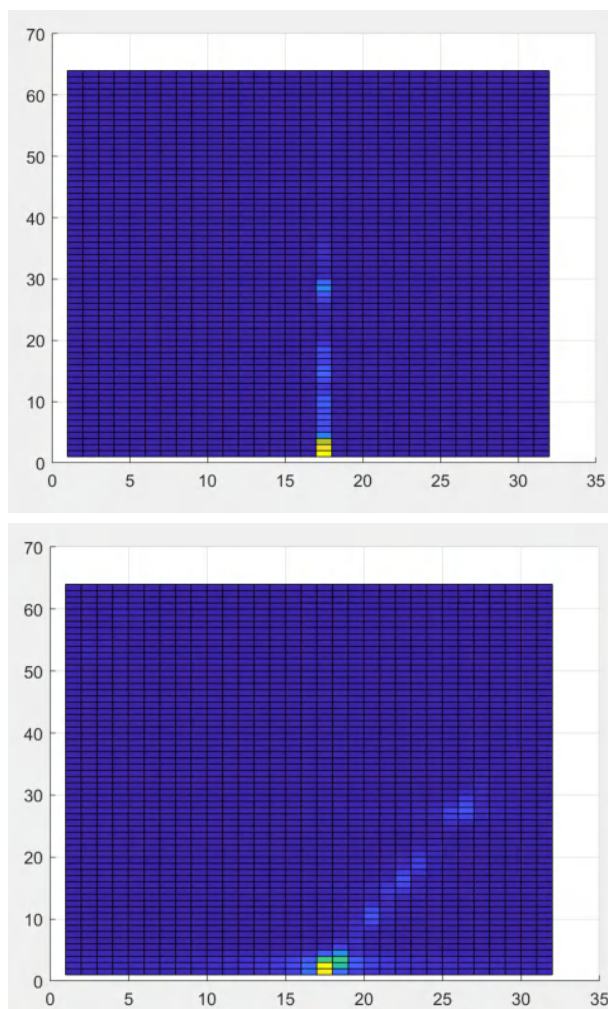


Figure 5-3. Comparison of Range-Doppler Heatmaps Before and After Enabling Look-Up Table Frequency Hopping

6 Conclusion

This paper introduces the configuration mechanism of the Per-Chirp LUT method and provides a repeatable and verifiable frequency hopping configuration instance based on this mechanism, thereby improving the range resolution of the system. This look-up table mechanism can help developers implement various advanced customized modulation strategies on the AWRL1432, such as AB-wave velocity extension, random timing parameter anti-interference, and custom BPM modulation. Developers can debug and verify by themselves based on the examples in this paper.

7 References

1. [AWRL1432 Datasheet](#)
2. AWRL1432 Interface Control Document (ICD)

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES "AS IS" AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you fully indemnify TI and its representatives against any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI's products are provided subject to [TI's Terms of Sale](#), [TI's General Quality Guidelines](#), or other applicable terms available either on [ti.com](#) or provided in conjunction with such TI products. TI's provision of these resources does not expand or otherwise alter TI's applicable warranties or warranty disclaimers for TI products. Unless TI explicitly designates a product as custom or customer-specified, TI products are standard, catalog, general purpose devices.

TI objects to and rejects any additional or different terms you may propose.

Copyright © 2026, Texas Instruments Incorporated

Last updated 10/2025