1. 数据率问题

mmWave radar sensor 传输的数据率较高，需要使用高速接口实时的将数据送出。数据率举例如下：

128(ADC samples per chirp) * 128(Chirps per frame)*4(RX channel)*4(IQ data bytes)*25(Frames per second) = 50Mbps

在数据传输过程中，考虑数据传输的开销，需要大于 50Mbps 的高速接口才可实时采集 25Hz 刷新率的数据。

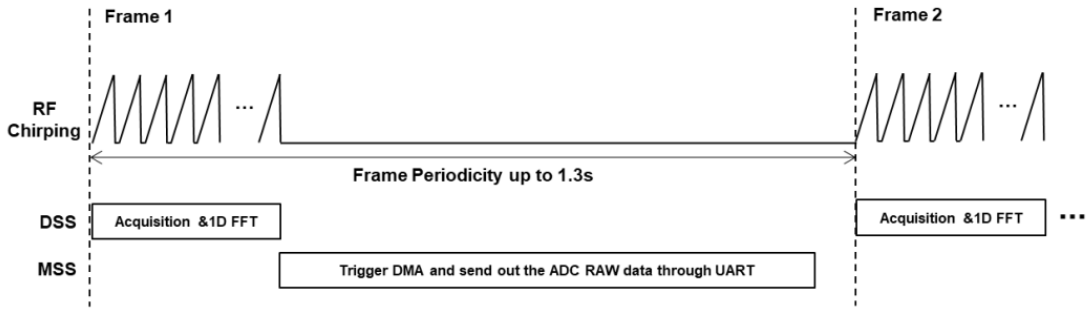
受限于 UART 口的传输速率，无法实时的获取 25Hz 刷新率的 ADC 数据，但可获得单个 frame 的 ADC 原始数据，也就是 1Hz 刷新率。对于 1Hz 刷新率，串口传输的数据获取的传输时间能力如下：

(256(ADC samples per chirp) * 64(Chirps per frame)*4(RX channel)*2(TX channel)*4(IQ data bytes)*1(Frames per second) * 8(8bits)) / (921600bps * 0.8(预估串口传输开销))= 5.69s

在使用 mmWave SDK 时，Frame 周期被限制为 300 us 到 1.342s，所以在使用 UART 进行数据获取时，需要注意控制数据总量，数据采集和传输时间不超过 RF Frame 的软件触发周期，即可实现连续的数据获取。

2. 处理时序

软件处理流程图如下图所示。



考虑到 L3 中的 ADC 数据可以一直保留到下一帧 chirp 开始，所以保证把 L3 的 ADC 数据需要在这个时间内发出即可。帧周期的选择，需要考虑预留足够的空闲时间给到数据传输，所以需要注意控制 frame 周期的设置。

3. 串口的工作模式

为满足数据传输速率的要求，串口的工作速率需要配置为 3125000bps，工作模式可以配置成 SDK 默认的普通模式，也可以打开 DMA，使用 DMA 进行数据搬移。工作模式的配置在 mss_main.c 文件 MmwDemo_initTask 函数中进行。

串口的普通模式配置代码如下：

```
/* Setup the default UART Parameters */
UART_Params_init(&uartParams);
uartParams.writeDataMode = UART_DATA_BINARY;
uartParams.readDataMode = UART_DATA_BINARY;
uartParams.clockFrequency = gMmwMssMCB.cfg.platformCfg.sysClockFrequency;
uartParams.baudRate       = gMmwMssMCB.cfg.platformCfg.loggingBaudRate;
uartParams.isPinMuxDone   = 1U;

/* Open the Logging UART Instance: */
gMmwMssMCB.loggingUartHandle = UART_open(1, &uartParams);
if (gMmwMssMCB.loggingUartHandle == NULL)
{
    System_printf("Error: Unable to open the Logging UART Instance\n");
    MmwDemo_debugAssert (0);
    return;
}
```

串口的 DMA 模式配置代码如下：

```
#define UART_DMA_TX_CHANNEL 1
#define UART_DMA_RX_CHANNEL 2
/* Open the DMA Instance */
DMA_Params_init(&dmaParams);
dmaHandle = DMA_open(0, &dmaParams, &errCode);
if (dmaHandle == NULL)
{
    printf ("Error: Unable to open the DMA Instance [Error code %d]\n", errCode);
    return;
}
/* Setup the default UART Parameters */
UART_Params_init(&uartParams);
uartParams.writeDataMode = UART_DATA_BINARY;
uartParams.readDataMode = UART_DATA_BINARY;
uartParams.clockFrequency = gMmwMssMCB.cfg.platformCfg.sysClockFrequency;
uartParams.baudRate       = gMmwMssMCB.cfg.platformCfg.loggingBaudRate;
uartParams.isPinMuxDone   = 1U;
uartParams.dmaHandle = dmaHandle;
uartParams.txDMAChannel = UART_DMA_TX_CHANNEL;
uartParams.rxDMAChannel = UART_DMA_RX_CHANNEL;

/* Open the Logging UART Instance: */
gMmwMssMCB.loggingUartHandle = UART_open(1, &uartParams);
if (gMmwMssMCB.loggingUartHandle == NULL)
{
    System_printf("Error: Unable to open the Logging UART Instance\n");
    MmwDemo_debugAssert (0);
    return;
}
```

4. 数据通信格式

新增加的 ADC 原始数据获取代码遵循与 SDK DEMO 一致的数据通信格式，以 TLV 的方式将 ADC 原始数据发出。如下表所示，SDK DEMO 中的 TLV type（包括新增） 在 MmwDemo_output_message_type 中定义。

TLV TYPE	Description
1	MMWDEMO_OUTPUT_MSG_DETECTED_POINTS
2	MMWDEMO_OUTPUT_MSG_RANGE_PROFILE
3	MMWDEMO_OUTPUT_MSG_NOISE_PROFILE
4	MMWDEMO_OUTPUT_MSG_AZIMUT_STATIC_HEAT_MAP
5	MMWDEMO_OUTPUT_MSG_RANGE_DOPPLER_HEAT_MAP
6	MMWDEMO_OUTPUT_MSG_STATS
7	MMWDEMO_OUTPUT_MSG_DETECTED_POINTS_SIDE_INFO
8	MMWDEMO_OUTPUT_MSG_AZIMUT_ELEVATION_STATIC_HEAT_MAP
9	MMWDEMO_OUTPUT_MSG_TEMPERATURE_STATS
10(New)	MMWDEMO_OUTPUT_MSG_L3_DATA
11	MMWDEMO_OUTPUT_MSG_MAX

新增 TLV type 如下图所示（见 mmwave_sdk_<ver>\packages\ti\demo\xwr16xx\mmw\include\mmw_output.h）。

```
    /*! @brief    temperature stats from Radar front end */
    MMWDEMO_OUTPUT_MSG_TEMPERATURE_STATS,

    MMWDEMO_OUTPUT_MSG_L3_DATA,

    MMWDEMO_OUTPUT_MSG_MAX
} MmwDemo_output_message_type;
```

r4f_linker.cmd 文件中可以看到，L3 的地址在 MSS 端的起始地址是 0x51000000。

```
/* Memory Map                                                              */
MEMORY{
    VECTORS  (X)  : origin=0x00000000 length=0x00000100
    PROG_RAM (RX) : origin=0x00000100 length=0x0003FF00+(MMWAVE_SHMEM_TCMA_NUM_BA
    DATA_RAM (RW) : origin=0x08000000 length=0x00030000+(MMWAVE_SHMEM_TCMB_NUM_BA
    L3_RAM (RW)   : origin=0x51000000 length=MMWAVE_L3RAM_NUM_BANK*MMWAVE_SHMEM_B
    HS_RAM (RW)   : origin=0x52080000 length=0x8000
}
```

L3 的大小 MMWAVE_L3RAM_SIZE=MMWAVE_L3RAM_NUM_BANK*MMWAVE_SHMEM_BANK_SIZE，后两者在 mmwave_sdk_<ver>\packages\ti\common\mmwave_sdk_xwr16xx.mak 文件中定义，如下图所示。

```
MMWAVE_SHMEM_BANK_SIZE  = 0x20000 #128KB

ifeq ("$(MMWAVE_SDK_SHMEM_ALLOC)", "")
SHMEM_ALLOC = 0x00000006                    # default case
MMWAVE_L3RAM_NUM_BANK = 6
MMWAVE_SHMEM_TCMA_NUM_BANK = 0
MMWAVE_SHMEM_TCMB_NUM_BANK = 0
else ifeq ($(MMWAVE_SDK_SHMEM_ALLOC), 0x00000006) # default case
SHMEM_ALLOC = $(MMWAVE_SDK_SHMEM_ALLOC)
MMWAVE_L3RAM_NUM_BANK = 6
MMWAVE_SHMEM_TCMA_NUM_BANK = 0
MMWAVE_SHMEM_TCMB_NUM_BANK = 0
```

在 mss_main.c 文件的“MmwDemo_transmitProcessedOutput”函数中增加如下代码来传送原始数据。

新增 TLV 代码如下：

```
tl[tlvIdx].type = MMWDEMO_OUTPUT_MSG_L3_DATA;
tl[tlvIdx].length = subFrameCfg->numRangeBins * subFrameCfg->numDopplerBins * subFrameCfg->numVirtualAntennas * 4;
packetLen += sizeof(MmwDemo_output_message_tl) + tl[tlvIdx].length;
tlvIdx++;
```

新增数据发送代码如下：

```
uint32_t *outdata;
outdata = 0x51000000;
UART_writePolling (uartHandle,(uint8_t*)&tl[tlvIdx],sizeof(MmwDemo_output_message_tl));
UART_writePolling(uartHandle, (uint8_t*)(&outdata[0]), tl[tlvIdx].length);
```

5. 数据格式

按照 4 所描述的方法，即可将原始数据送出，需要注意的是，在帧处理时送出的数据，是经过了 1D FFT 运算的数据，是 1D FFT 的结果，1D FFT 的运算发生在 chirp 处理阶段。所以，如需要获得 1D FFT 运算之前的 ADC 数据，需要将 FFTEN 及 windowEN 设置修改为 0，需要修改 range processing 的 DPC 代码（修改部分函数名 rangeProcHWA_ConfigHWA），示意如下，注意修改完成后需要重新编译 .\datapath\dpu\rangeproc 的库。

（见 mmwave_sdk_<ver>\packages\ti\datapath\dpu\rangeproc\src\rangeprochwa.c）

```
hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.fftEn = 0;//SDK default 1
hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.fftSize = mathUtils_ceilLog2(
hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.butterflyScaling =
                            (1 << pDPParams->numLastButterflyStagesToSca
hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.interfZeroOutEn = 0;
hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.windowEn = 0;//SDK default 1
hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.windowStart = rangeProcObj->h
hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.winSymm = rangeProcObj->hwaCf
hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.winInterpolateMode = 0;
```

6. TLV 配置

由上面分析可知，我们需要的原始数据是 tag 为 10 的 TLV，如果为了验证算法，还需要 tag 为 1 的 TLV(包括目标的位置和速度信息)。TLV 的配置代码如下图所示(见 mmw_cli.c)。它们的配置数将关系到收到的帧中含有哪些 TLV。logMagRange 为 1 将收到 tag 为 2 的 TLV，noiseProfile 为 1 将收到 tag 为 3 的 TLV，rangeAzimuthHeatMap 为 1 将收到 tag 为 4 的 TLV，rangeDopplerHeatMap 为 1 将收到 tag 为 5 的 TLV，statsInfo 为 1 将收到 tag 为 6 和 9 的 TLV。反之则不会收到对应的 TLV。

```
426     /* Populate configuration: */
427     guiMonSel.detectedObjects           = atoi (argv[2]);
428     guiMonSel.logMagRange               = atoi (argv[3]);
429     guiMonSel.noiseProfile              = atoi (argv[4]);
430     guiMonSel.rangeAzimuthHeatMap       = atoi (argv[5]);
431     guiMonSel.rangeDopplerHeatMap       = atoi (argv[6]);
432     guiMonSel.statsInfo                 = atoi (argv[7]);
```

需要注意的是 detectedObjects 为 0 不会收到对应的 TLV，为 1 将收到 tag 为 1 和 7 的 TLV，为 2 将收到 tag 为 1 的 TLV，相关代码（见 mmw_main.c）如下图所示。

```
if ((pGuiMonSel->detectedObjects == 1) || (pGuiMonSel->detectedObjects == 2) &&
     (result->numObjOut > 0))
{
    tl[tlvIdx].type = MMWDEMO_OUTPUT_MSG_DETECTED_POINTS;
    tl[tlvIdx].length = sizeof(DPIF_PointCloudCartesian) * result->numObjOut;
    packetLen += sizeof(MmwDemo_output_message_tl) + tl[tlvIdx].length;
    tlvIdx++;
}
/* Side info */
if ((pGuiMonSel->detectedObjects == 1) && (result->numObjOut > 0))
{
    tl[tlvIdx].type = MMWDEMO_OUTPUT_MSG_DETECTED_POINTS_SIDE_INFO;
    tl[tlvIdx].length = sizeof(DPIF_PointCloudSideInfo) * result->numObjOut;
    packetLen += sizeof(MmwDemo_output_message_tl) + tl[tlvIdx].length;
    tlvIdx++;
}
```

配置信息 argv 来自雷达配置文件的相应行。为了只收到 tag 为 1 和 10 的 TLV，guiMonitor 配置应为 2 0 0 0 0 0，如下图所示。

```
lowPower 0 1
guiMonitor -1 2 0 0 0 0 0
cfarCfg -1 0 2 8 4 3 0 15
```

7. 获取数据

获取原始数据的步骤如下。

(1) 识别控制串口(COM4)和数据串口(COM3)并链接，读取雷达配置文件。

```
delete(instrfind);                  % 删除串口
[controlSerialPort,dataSerialPort] = serial_port();      % 自动识别串口
configurationFileName_stop ='awr1642.cfg';
cliCfg = readCfg(configurationFileName_stop);           % 配置文件读取
hdataSerialPort = configureDataSport(dataSerialPort,8e6);%链接 COM串口
mmwDemoCliPrompt = char('mmwDemo:/>');
hControlSerialPort = configureControlPort(controlSerialPort);%链接 COM串口
```

(2) 将相应配置发送到控制串口，启动雷达。

```
for k=1:length(cliCfg)
    fprintf(hControlSerialPort, cliCfg{k});
    fprintf('%s\n', cliCfg{k});
    echo = fgetl(hControlSerialPort); % Get an echo of a command
    done = fgetl(hControlSerialPort); % Get "Done"
    prompt = fread(hControlSerialPort, size(mmwDemoCliPrompt,2)); % Get the prompt back
end
```

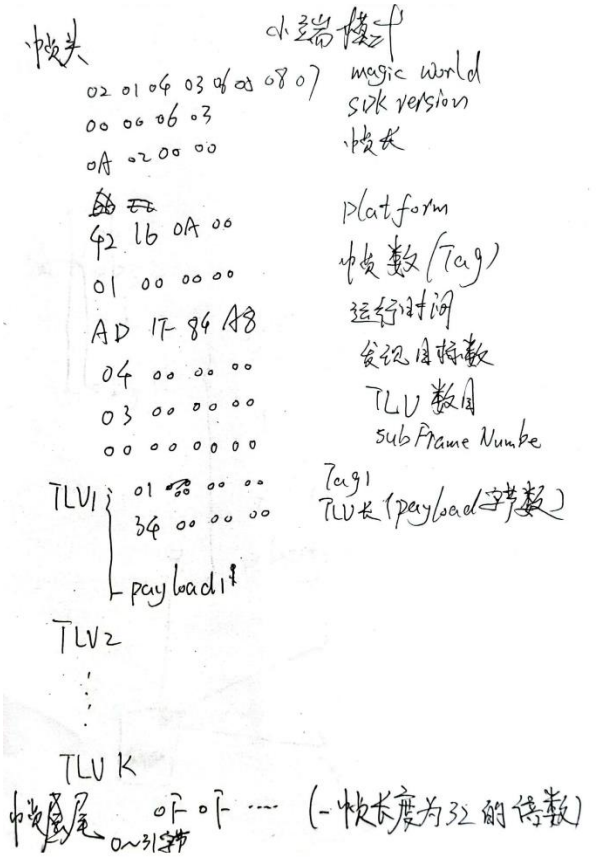
(3) 从数据串口读取数据，获得包含原始数据的帧，为方便后续研究，可对其进行保存，并关闭控制串口和数据串口。

```
Data_dec= fread(hdataSerialPort); % Get the prc
prompt_1=[prompt_1;Data_dec];
save(filename,'prompt_1');
fprintf(hControlSerialPort, 'sensorStop');
fclose(hControlSerialPort);
delete(hControlSerialPort);
fclose(hdataSerialPort);
delete(hdataSerialPort);
```

8. 数据处理

(1) 从帧中获取原始数据

帧结构如下图所示。



获取的数据 udata 可能含有多个帧，处理代码如下图所示。

```
while cur<len
    framelenhex=[udata(cur+16,:),udata(cur+15,:),udata(cur+14,:),u
    framelen=hex2dec(framelenhex);
    next=cur+framelen;
    if next>len
        break
    end
    fdata=udata((cur+1):(cur+framelen),:);
    frame_info=FrameProcess(fdata);
    range=[range,frame_info(1)];
    doppler=[doppler,frame_info(2)];
    angle=[angle,frame_info(3)];
    cur=next;
end
```

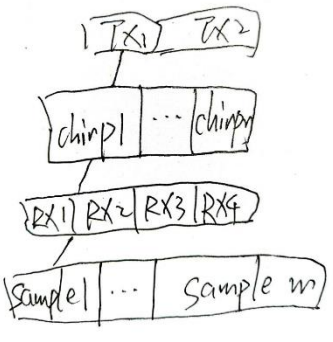
每次先获取当前帧的帧长，再对当前帧的数据进行处理，处理代码如下图所示。

```
TLV_num=zeros(10,1);
cur=40;
for k=1:TLV_nums
    TLV_lenhex=[fdata(cur+8,:),fdata(cur+7,:),fdata(cur+6,:),fdata
    TLV_len=hex2dec(TLV_lenhex);
    TLV_tag=hex2dec(fdata(cur+1,:));
    payload=fdata((cur+9):(cur+8+TLV_len),:);
    cur=cur+8+TLV_len;
    switch TLV_tag
        case 1
            [dst,velocity,angle]=DobjProcess(payload,TLV_len);
            TLV_num(1)=TLV_num(1)+1;
        case 10
            L3Process(payload,TLV_len);
            TLV_num(10)=TLV_num(10)+1;
```

首先获取帧中的 TLV 数目（根据 6 所述，应该只包含 tag 为 1 和 10 的 TLV)。每次先获取当前 TLV 的长度，再根据相应的 tag 对当前 TLV 的数据进行处理。当读取的 TLV 的 tag 为 10 时，说明成功获取到当前帧的原始数据。

(2) 原始数据结构

原始数据结构如下图所示。



由下图(见 mss_main.c)可知，每个采样点数据(sample)由虚部和实部各两字节(uint_16)合计 4 字节组成，且虚部数据在前，实部数据在后。

```
if (subFrameCfg->adcBufCfg.iqSwapSel == 1)
{
    staticCfg->ADCBufData.dataProperty.dataFmt = DPIF_DATAFORMAT_COMPLEX16_IMRE;
}
else
```

取 m=256，n=32，则原始数据的长度为 256(ADC samples per chirp) * 16(Chirps per frame)*4(RX channel)*2(TX channel)*4(IQ data bytes)=262144 字节。

(3) 数据拆分、组合

由于每个数据的大小是两字节，因此首先将原始数据 Data_dec 的每两个字节合为一个数，考虑到数据存储采用小端模式，因此两字节中的后一字节为高位，合成后的数转化为 16 字节的有符号数。代码如下图所示。

```
Data_zuhe=zeros(1,Tx_Number*Rx_Number*Doppler_Number*Range_Number*2);

for i=1:Tx_Number*Rx_Number*Doppler_Number*Range_Number*2

    Data_zuhe(i) = Data_dec((i-1)*2+1)+Data_dec((i-1)*2+2)*256;%两个字
    if(Data_zuhe(i)>32767)
        Data_zuhe(i) = Data_zuhe(i) - 65536;  %限制幅度
    end

end
```

对数据的实部和虚部进行分离并重组得到复信号。代码如下图所示。

```
Re_Data_All=zeros(1,Range_Number*Doppler_Number*Tx_Number*Rx_Number); %
Im_Data_All=zeros(1,Range_Number*Doppler_Number*Tx_Number*Rx_Number); %

% 虚部实部分解
for i=1:Tx_Number*Rx_Number*Doppler_Number*Range_Number
    Im_Data_All(1,i) = Data_zuhe(1,(i-1)*2+1);
    Re_Data_All(1,i) = Data_zuhe(1,(i-1)*2+2);
end

%% 虚部+实部数据重组得到复信号
ReIm_Data_All =complex(Re_Data_All,Im_Data_All);
```

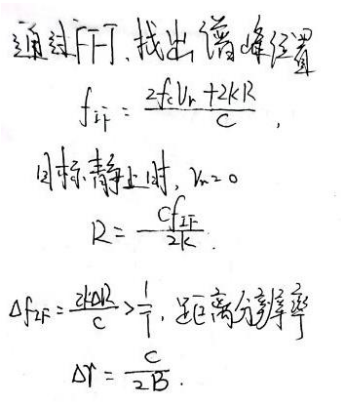
根据前文原始数据结构的分析，我们将回波数据排列成 3 维矩阵(n_chirps*n_samples*n_X)，分别是速度维、距离维和天线维。分别用 ADC_Data1 和 ADC_Data2 存储两根发射天线的数据，最后拼接在一起。代码如下图所示。

```
ADC_Data1=zeros(Doppler_Number,Range_Number,Rx_Number); %建立计算存储数据的空矩阵
ADC_Data2=zeros(Doppler_Number,Range_Number,Rx_Number); %建立计算存储数据的空矩阵

RX_num=Range_Number;
chirp_num=RX_num*Rx_Number;
TX_num=chirp_num*Doppler_Number;
for i=1:Doppler_Number
    for j=1:Range_Number
        for k=1:Rx_Number
            ADC_Data1(i,j,k) = ReIm_Data_All((i-1)*chirp_num+(k-1)*RX_num+j);
            ADC_Data2(i,j,k) = ReIm_Data_All(TX_num+(i-1)*chirp_num+(k-1)*RX_num+j);
        end
    end
end
ADC_Data = cat(3,ADC_Data1(:,:,1:Rx_Number), ADC_Data2(:,:,1:Rx_Number));
```

(4) 距离 FFT

对单个脉冲采样数据（距离维）进行 1D FFT 得到矩阵 fft1d，对各脉冲信息整合，得到整体的距离变化信息。采样频率 fs，采样点数 N，分辨率 df=fs/N，第 k 个采样点的频率 f=(k-1)*df。距离信息的获取原理如下图所示。



距离由谱峰确定，可见采样点数代表了距离信息。

距离 FFT 的代码如下图所示。

```
fft1d= zeros(Doppler_Number,Range_Number,Tx_Number*Rx_Number);
for qq =1:Tx_Number*Rx_Number
    for chirp_fft=1:Doppler_Number
        fft1d(chirp_fft,:,qq) = fft(ADC_Data(chirp_fft,:,qq));
    end
end
```

(5) 静态杂波滤除（相量均值相消法）

静态杂波滤除可采用相量均值相消法。其原理为，对所有接收脉冲求平均得出参考接收脉冲，接着利用每一束接收脉冲减去参考接收脉冲就可以得到目标回波信号，参考接收脉冲的表达式为

$$C[m]=\frac{1}{N}\sum_{i=1}^{N}R[m,i]$$

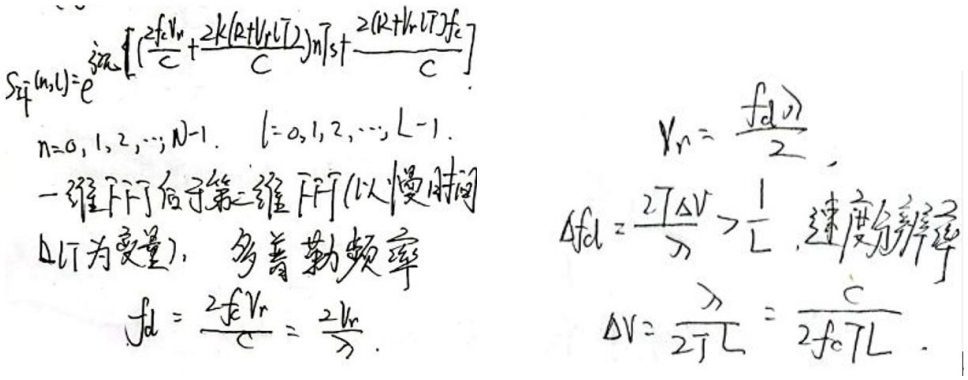
其中，m 为距离维采样点，i 为速度维采样点，相量均值相消算法的公式为 R[m,n]=R[m,n]-C[m]。

静态杂波滤除的代码如下图所示。

```
fft1d_jingtai =zeros(Doppler_Number,Range_Number,Tx_Number*Rx_Number);
for n=1:Tx_Number*Rx_Number
    avg = sum(fft1d(:,:,n))/Doppler_Number;
    for m=1:Doppler_Number
        fft1d_jingtai(m,:,n) = fft1d(m,:,n)-avg;
    end
end
fft1d =fft1d_jingtai;
```

(6) 速度 FFT

对距离 FFT 的结果在速度维作 1D FFT 得到矩阵 fft2d。设信号采样周期为 Ts，脉冲重复间隔为 T，单脉冲采样点数为 N，接收 L 个脉冲。速度信息的获取原理如下图所示。



速度 FFT 的代码如下图所示。

```
fft2d= zeros(Doppler_Number,Range_Number,Tx_Number*Rx_Number);
for kk=1:Tx_Number*Rx_Number
    for chirp_fft=1:Range_Number
        fft2d(:,chirp_fft,kk) =fftshift( fft(fft1d(:,chirp_fft,kk)));
    end
end
```

(7) 直流分量抑制

基于 77GHz/79GHz FMCW 体制的毫米波雷达，在 RD(Ranger-Doppler)阈检测时，我们会遇到一个直流分量的问题，进而导致近距离（2~3 个采样点）无法处理，从而造成一个较大的盲区。

这里采取的方法是将第一个采样点的数据置 0，代码如下图所示。

```
fft2d(:,1,:)=0;
```

(8) 角度 FFT

对速度 FFT 结果在天线维作 1D FFT 得到矩阵 fft3d。毫米波波长为 λ，天线阵列间距为 d，设定 Q=180 返回 Q 点 DFT。

角度计算方法如下，峰值天线维索引为 pag。得到空间频率 fw = (pag-Q/2-1)/Q，角度$\theta = sin^{-1}(fw * \lambda/d)$。

角度 FFT 的代码如下图所示。

```
fft3d = zeros(Doppler_Number,Range_Number,Q);
for qq =1:Doppler_Number
    for kk=1:Range_Number
        fft3d(qq,kk,:) = fftshift(fft(fft2d(qq,kk,:),Q));
    end
end
```

(9) 绘制距离 FFT 结果

fft1d 的结果。

```
FFT1_mag=abs(fft1d(:,:,1));
figure();
mesh(FFT1_mag);
xlabel('采样点数');ylabel('脉冲数');zlabel('幅度');
xlim([0 NSample]);ylim([0 NChirp]);
title('距离维FFT结果');
```

(10) 绘制 2 维 FFT 结果

fft2d 的结果。需要注意的是距离和速度刻度，由前文推导可以知道快时间维的采样点数（每个 chirp 的采样数）代表距离信息，慢时间维的采样点数（每帧的 chirp 数）代表速度信息，由推导公式可以得到三维图的 X 和 Y 坐标刻度。

```
FFT2_mag=abs(fft2d(:,:,1));
[X,Y] = meshgrid(c*(0:NSample-1)*Fs/2/freqSlope/NSample,...
    (-NChirp/2:NChirp/2 - 1)*lambda/Tc/NChirp/2);
figure();
mesh(X,Y,FFT2_mag);
xlim([0 c*(NSample-1)*Fs/2/freqSlope/NSample]);
ylim([(-NChirp/2)*lambda/Tc/NChirp/2 (NChirp/2 - 1)*lambda/Tc/NChirp/2]);
xlabel('距离(m)');ylabel('速度(m/s)');zlabel('幅度');
title('2D-FFT结果');
```

(11) 绘制角度维 FFT 结果

fft3d 的结果。需要注意的是距离和角度刻度，由推导公式可以得到三维图的 X 和 Y 坐标刻度。

```
FFT3_mag=reshape(abs(fft3d(1,:,:)),Range_Number,Q).';
[R,Z] = meshgrid(c*(0:NSample-1)*Fs/2/freqSlope/NSample,...
    asin((-Q/2:Q/2 - 1)*lambda/d/Q)*180/pi);
figure();
mesh(R,Z,FFT3_mag);
xlim([0 c*(NSample-1)*Fs/2/freqSlope/NSample]);
ylim([asin((-Q/2)*lambda/d/Q)*180/pi asin((Q/2 - 1)*lambda/d/Q)*180/pi]);
xlabel('距离维(m)');ylabel('角度维(° )');zlabel('幅度');
title('角度维FFT结果');
```

(12) 谱峰搜索

通过谱峰搜索找到峰值下标，即可换算成对应的距离、速度信息。

对于单目标的情况，计算最大峰值即可，其代码如下图所示。

```
fft3d=abs(fft3d);
pink=max(fft3d(:));
[row,col,pag]=ind2sub(size(fft3d),find(fft3d==pink));
```

得到峰值对应的索引后，由距离、速度和角度的推导公式可以得到相应信息，其代码如下图所示。

```
fb = ((col-1)*Fs)/NSample;            %差拍频率
fd = (row-NChirp/2-1)/(NChirp*Tc);    %多普勒频率
fw = (pag-Q/2-1)/Q;                   %空间频率
R = c*fb/2/freqSlope;                 %距离公式
v =lambda*fd/2;                       %速度公式
theta = asin(fw*lambda/d);            %角度公式
angle = theta*180/pi;
fprintf('目标距离：  %f m\n',R);
fprintf('目标速度：  %f m/s\n',v);
fprintf('目标角度：  %f° \n',angle);
```

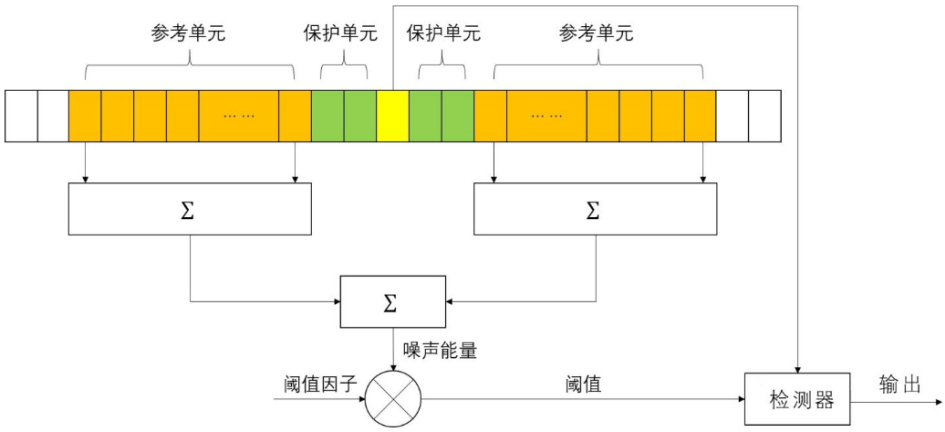
对于多目标的情况，一般采用 CFAR（Constant False Alarm Rate，恒虚警率）检测。这里采用 CA-CFAR (平均单元恒虚警率)，根据想要的虚警概率 PFA 去计算阈值因子∂，再利用周围参考单元的值去估计待测单元的噪声功率，进而就能得到门限=$\partial * P_n$（其中P_n为噪声功率），若待测单元的值大于门限值，则判断为有目标，反之为无目标。

阈值因子∂和噪声功率P_n的的计算公式如下，

$$\partial = N * (PFA^{-\frac{1}{N}} - 1)$$

$$P_n = \frac{1}{N} * sum(X)$$

其中 N 表示 X 的长度，为参考单元的总数，而 X 的值为参考单元的值。CFAR 检测器一般情况下起始和结尾的参考单元数都是一样的，守护单元被放在待测单元的两侧。一维 CFAR 检测原理模型如下图所示。



一维 CFAR 的实现代码如下图所示。其中 inputArr 为输入矩阵，NTrain 为单侧的参考单元数，NGuard 为单侧的保护单元数，PFA 为设定的虚警率。返回值 cfar1D_Arr 用于存储判决结果（是否有目标），threshold 用于存储检测单元的判决门限。

```
function [cfar1D_Arr,threshold] = ac_cfar1D(NTrain,NGuard,PFA,inputArr)
    cfar1D_Arr = zeros(size(inputArr));
    threshold = zeros(size(inputArr));

    totalNTrain = 2*(NTrain);
    a = totalNTrain*((PFA^(-1/totalNTrain))-1);
    %求平均值
    for i = NTrain+NGuard+1:length(inputArr)-NTrain-NGuard
        avg = mean([inputArr((i-NTrain-NGuard):(i-NGuard-1))...
            inputArr((i+NGuard+1):(i+NTrain+NGuard))]);
        threshold(1,i) = a.*avg;
        %根据threshold比较
        if(inputArr(i) < threshold(i))
            cfar1D_Arr(i) = 0;
        else
            cfar1D_Arr(i) = 1;
        end
    end
end
```

沿速度维进行 1D CFAR 的代码如下图所示。其中 FFT2_mag 是进行 2D FFT 后第一个通道的结果。

```
Tv=6;Pv=4;PFAv = 0.001;

dopplerDimCfarThresholdMap = zeros(size(FFT2_mag));  %创建一个
dopplerDimCfarResultMap    = zeros(size(FFT2_mag));

for i=1:Range_Number
    dopplerDim =  reshape(FFT2_mag(:,i),1,Doppler_Number);
    [cfar1D_Arr,threshold] = ac_cfar1D(Tv,Pv,PFAv,dopplerDim);
    dopplerDimCfarResultMap(:,i) = cfar1D_Arr.';
    dopplerDimCfarThresholdMap(:,i) = threshold.';
end
```

沿着 doppler 维度方向寻找在 doppler 维 cfar 判决后为 1 的结果，并保存有物体的 doppler 坐标，其代码如下图所示。

```
saveMat = zeros(size(FFT2_mag));
for range = 1:Range_Number
    indexArr = find(dopplerDimCfarResultMap(:,range));
    objDopplerArr = [indexArr;zeros(Doppler_Number - length(indexArr),1)];
    saveMat(:,range) = objDopplerArr; %保存doppler下标
end
% 保存有物体的doppler坐标
objDopplerIndex = unique(saveMat);  % unqiue是不重复的返回数组中的数
objDopplerIndex(objDopplerIndex==0)=[]; %排除数组中的0
```

根据之前 doppler 维的 cfar 结果对应的下标 objDopplerIndex，对相应的速度进行 range 维度的 CFAR，其代码如下图所示。

```
Tr=6;Pr=4;PFAr = 0.003;

rangeDimCfarThresholdMap = zeros(size(FFT2_mag));  %创建一个
rangeDimCfarResultMap = zeros(size(FFT2_mag));

for i = 1:length(objDopplerIndex)
    %根据速度下标进行range CFAR
    j = objDopplerIndex(i);      % 获得物体所在的行
    rangeDim = reshape(FFT2_mag(j, :),1,Range_Number);  %变成
    [cfar1D_Avv,threshold] = ac_cfar1D(Tr,Pr,PFAr,rangeDim);
    rangeDimCfarResultMap(j,:) = cfar1D_Avv;
    rangeDimCfarThresholdMap(j,:) = threshold;
end
```

绘制 CFAR 判决结果的代码如下图所示。

```
figure();
mesh(X,Y,(rangeDimCfarResultMap));
xlabel('距离(m)');ylabel('速度(m/s)');zlabel('信号幅值');
title('rangeCFAR之后判决结果(峰值聚集前)');
```

对于判决为有目标的位置和速度索引，检测其是否为峰值，并对确定为峰值的目标确定其天线维最大值的索引，其代码如下图所示。

```
[objDprIdx,objRagIdx] = peakFocus(rangeDimCfarResultMap,FFT2_mag);%
objAglIdx=zeros(1,length(objDprIdx));%角度的ID号
for i=1:length(objDprIdx)
    row=objDprIdx(i);
    col=objRagIdx(i);
    [~,id]=max(fft3d(row,col,:));
    objAglIdx(i)=id;
end
```

峰值检索函数 peakFocus 的代码如下图所示。其思路是对判决为有目标的坐标，检测其是否是附近 3 * 3 矩阵中的最大值，若是则判断为峰值，记录其距离和速度索引。

```
function [row,column] = peakFocus(inputCfarResMat,FFT2_mag)
    j = 1;
    row = zeros([1 256]);
    column = zeros([1 256]);
    [d,r] = find(inputCfarResMat==1);   %寻找进行range维cfar后的判决为1的坐标
    for i = 1 : length(d)
        peakRow = d(i);
        peakColumn = r(i);
        peak = FFT2_mag(peakRow,peakColumn);  %待验证的峰值
        % 在附近的3*3矩阵中的数进行比较,如果中间的数是最大值,就判定为1
        % 根据之前进行的2次cfar,因为有TrainCell和GuardCell，所以不会碰到边缘
        tempArr =[FFT2_mag(peakRow-1,peakColumn-1) , FFT2_mag(peakRow-1,peakColum
                  FFT2_mag(peakRow,peakColumn-1)   ,                      peak
                  FFT2_mag(peakRow+1,peakColumn-1) , FFT2_mag(peakRow+1,peakColum
        truePeak = max(tempArr);     % 寻找最大值
        if(truePeak == peak)         %如果中间的是最大值就保存当前的坐标
            row(j) = peakRow;
            column(j) = peakColumn;
            j = j+1;
        end
    end
    row(row==0)=[]; %去掉后面的0
    column(column==0)=[];
end
```

经过峰值检索后，获取到目标的索引，利用推导公式计算目标距离、速度、角度，其代码如下图所示。

```
objSpeed = ( objDprIdx - NChirp/2 - 1)*lambda/Tc/NChirp/2;
objRange = single(c*(objRagIdx-1)*Fs/2/freqSlope/NSample);
objAngle = asin((objAglIdx - Q/2 - 1)*lambda/d/Q)*180/pi;
```