**雷达数据流处理流程**

**软件介绍**

目录

[1. 雷达波形参数，天线阵列及性能介绍 3](#_Toc101109822)

[1.1 波形简介 3](#_Toc101109823)

[1.2天线阵列 3](#_Toc101109824)

[1.3性能简介 4](#_Toc101109825)

[2 DSP软件架构 5](#_Toc101109826)

[3 DSP内存分配简介 6](#_Toc101109827)

[3.1芯片内存分布 6](#_Toc101109828)

[3.2 程序中内存分配情况 6](#_Toc101109829)

[4 数据流处理过程 7](#_Toc101109830)

[4.1正常数据处理流程 7](#_Toc101109831)

[4.2雷达标定数据处理流程 8](#_Toc101109832)

[5 DSP任务系统设计 9](#_Toc101109833)

[5.1任务系统简介 9](#_Toc101109834)

[5.2 任务frertos\_main 10](#_Toc101109835)

[5.3任务Task\_Chirp 12](#_Toc101109836)

[5.4任务Task\_Frame 14](#_Toc101109837)

[6 ADC采样配置 15](#_Toc101109838)

[7 FRAME数据处理 16](#_Toc101109839)

[7.1 frame数据处理中采用的基本算法实现介绍 16](#_Toc101109840)

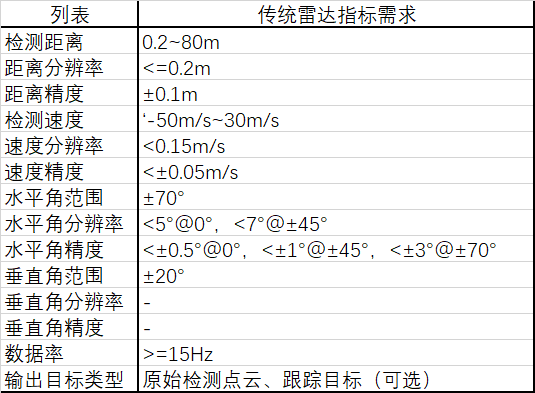
[7.2 frame数据处理介绍 21](#_Toc101109841)

[8速度解模糊 25](#_Toc101109842)

[9 DBF求角 26](#_Toc101109843)

# 雷达波形参数，天线阵列及性能介绍

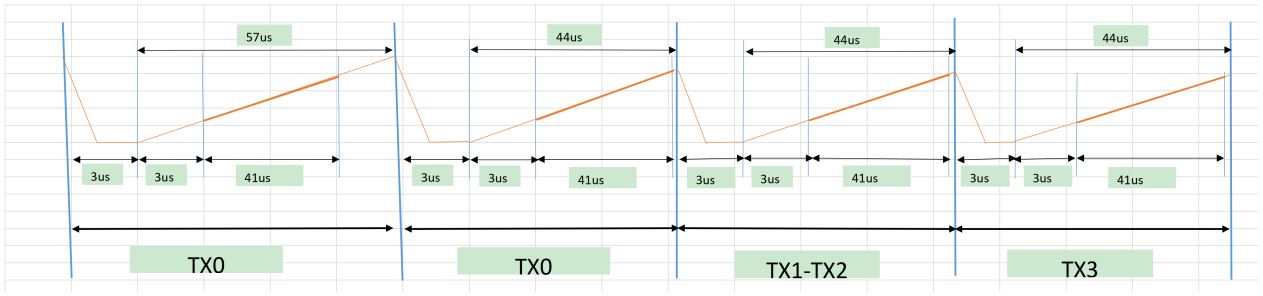
## 2944\_750M波形简介

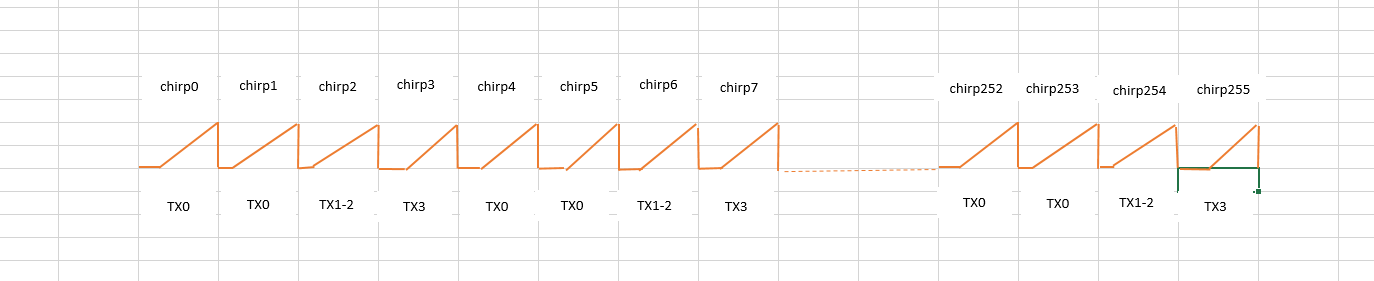


波形在arm核进行配置初始化，每次触发产生一个FRAME，每个FRAME产生256 chirp，做1D，然后触发下一个frame（发完256个chirp才能做2D）

Chirp时间及chirp之间的相关关系如下图图1所示。实际每个FRAME所发波形是图2的64次循环。

图 1

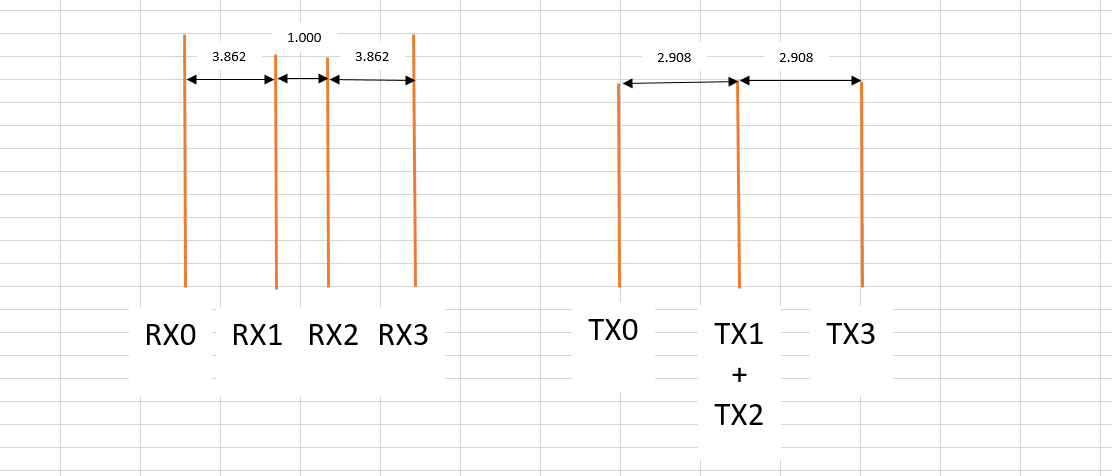




## 1.2天线阵列

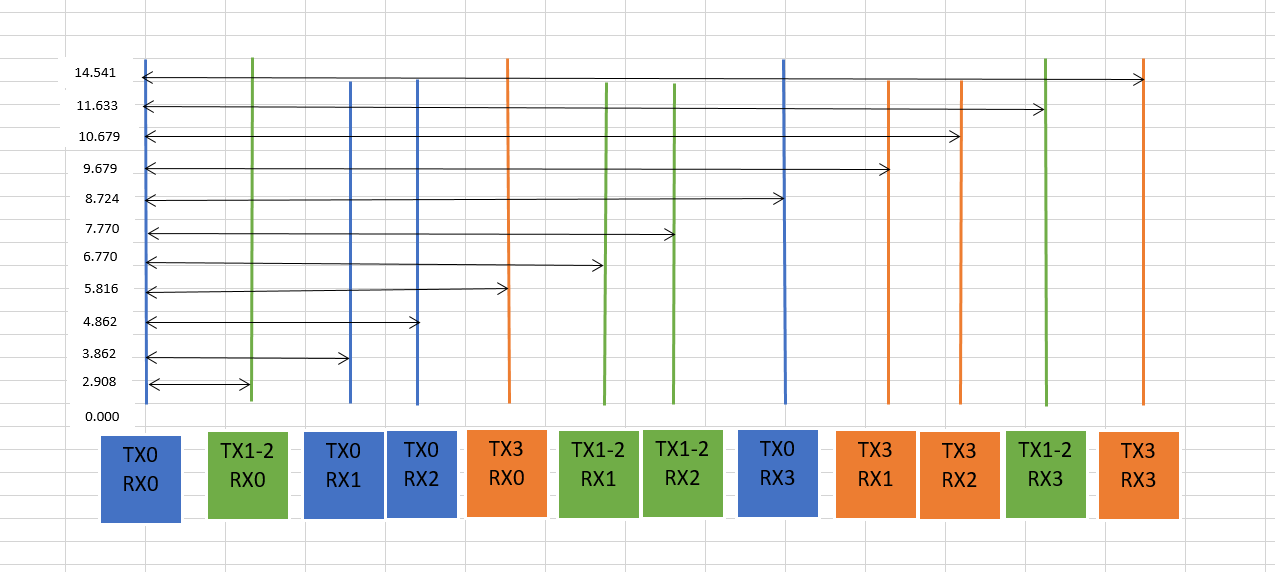
(1)实际天线阵列如下图2，图中天线间距数值为0.5λ(电磁波中心频率为76.5GHZ), λ实际值为3.9215mm。

图 2



(2)虚拟天线阵列如下图3，图中天线间距数值为0.5λ。

图 3



## 1.3性能简介

|  |  |
| --- | --- |
| **性能项目** | **值** |
| 数据更新周期 | 50ms |
| 最大跟踪点数 | 64个 |
| 最大输出点云数 | 120个 |
| 探测距离范围 | 0.2—80米 |
| 距离分辨率 | 0.2m |
| 测速范围 | -50m/s-30m/s |
| 速度分辨率 | 0.15m/s |
| 角度范围 | -75°-75° |
| 角度分辨率 | 6° |
|  |  |

# 2 DSP软件架构

图 4



# DSP内存分配简介

## 3.1芯片内存分布

芯片内存分布如下图图5，对于DSS\_L3表格中显示有2.5MB可用，但是DSP寻址中只能寻地址2MB范围，如图6所示。

图 5

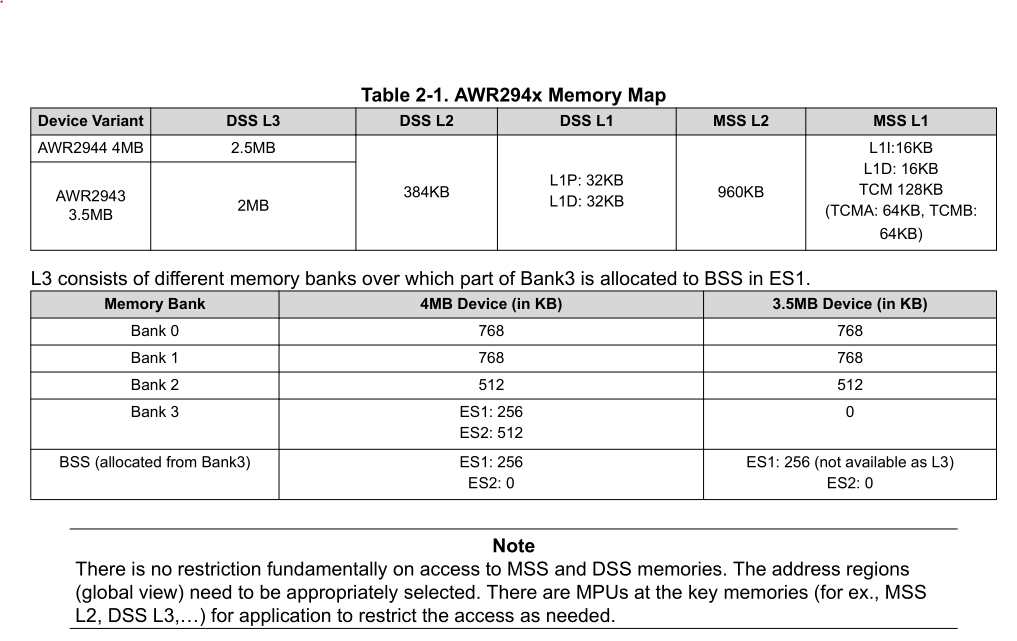
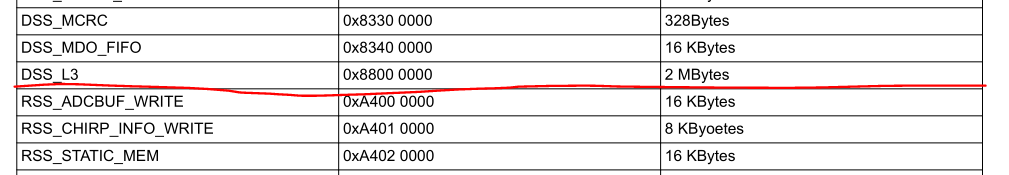


图 6

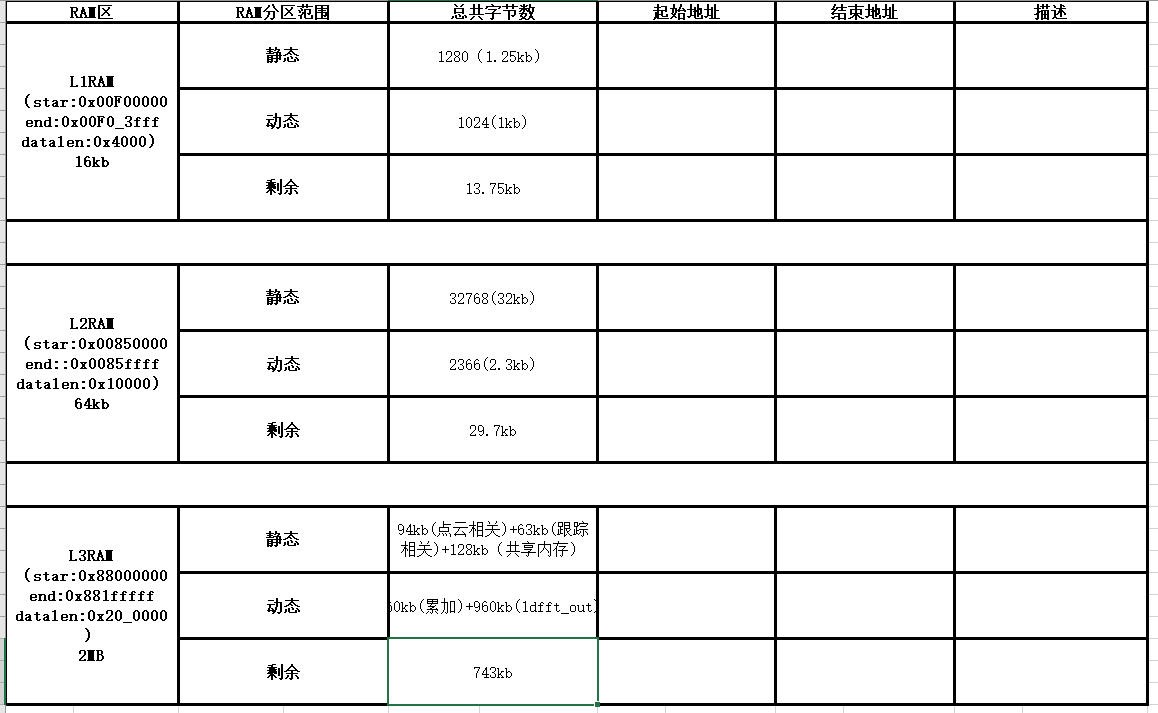


## 3.2 程序中内存分配情况

下图中的L1RAM是从L1D中分配出来的，其余用给程序使用。下图中的L2RAM是从

DSS\_L2中分配了64kb,剩余的320kb留给程序使用。L2RAM使用了DSS\_L3中的2MB。

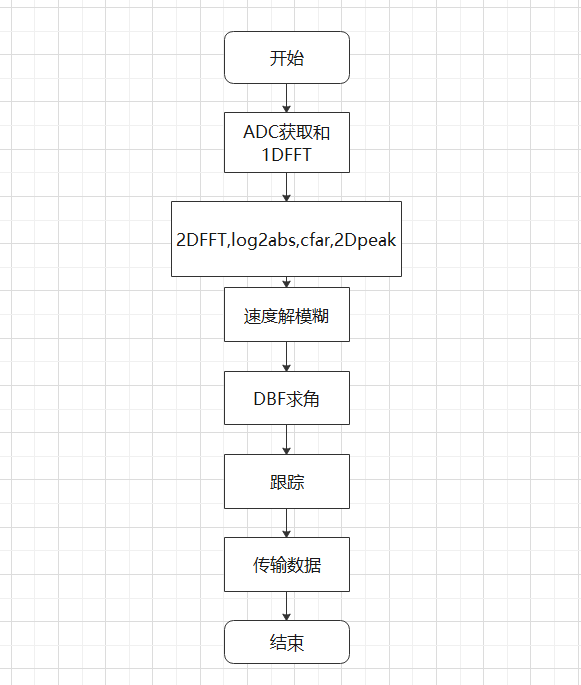
图 7



# 数据流处理过程

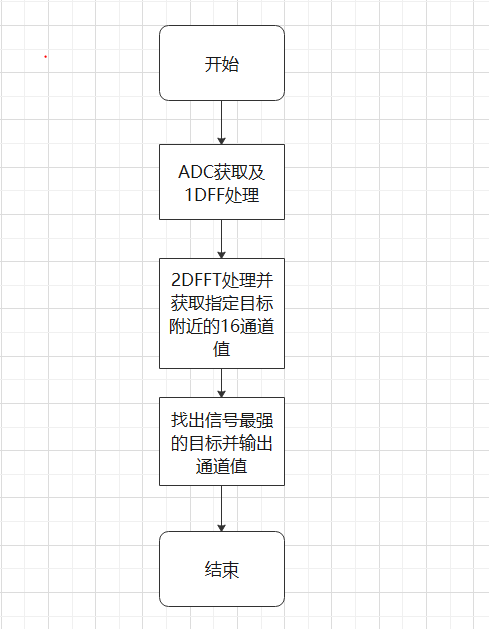
## 4.1正常数据处理流程

图 8



## 4.2雷达标定数据处理流程

图 9

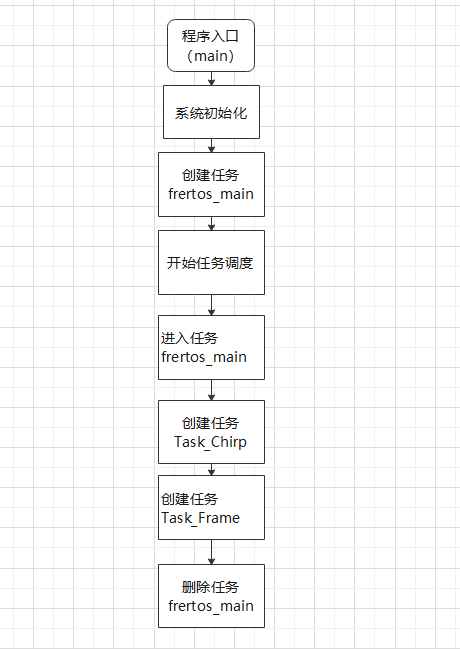


# DSP任务系统设计

## 5.1任务系统简介

DSP核中使用的操作系统为FREE\_RTOS,程序中建立了3个任务，任务分别为frertos\_main,Task\_Chirp,Task\_Frame。任务创建流程如下图10。

图 10

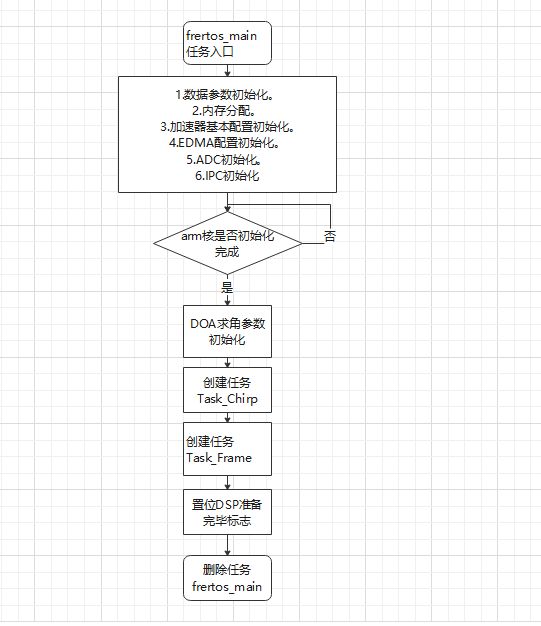


## 5.2 任务frertos\_main

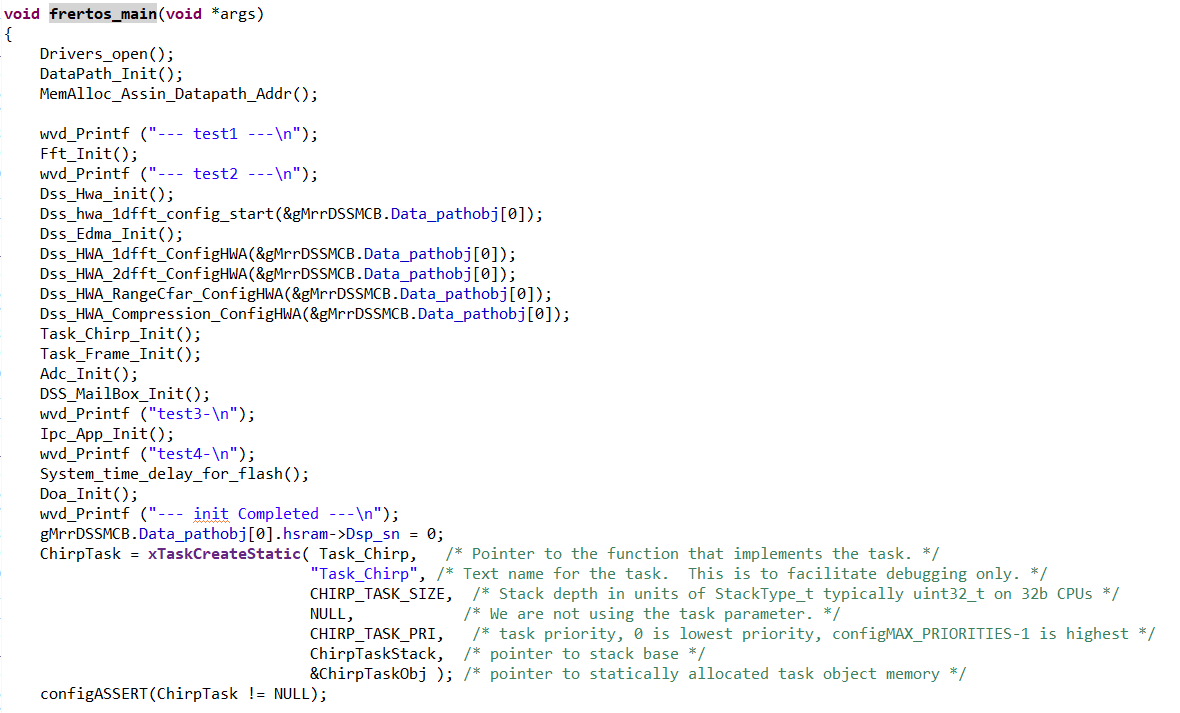
(1) 任务frertos\_main的功能：初始化波形参数，初始化内存分配，初始化ADC，初始化EDMA，初始化硬件加速器，初始化IPC核间通讯，创建任务Task\_Chirp,Task\_Frame。

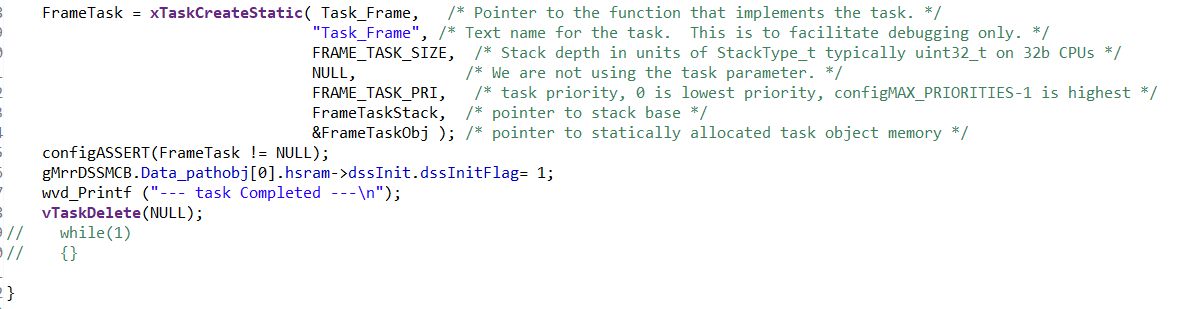
（2）任务逻辑如下图11所示：

（**注**：对于初始化阶段中插入等待arm核初始化完成判断是因为雷达数据处理参数是存储在FLASH中，所以部分参数初始化需要等待程序从FALSH读出完整参数后才能进行）

图11

（3）代码如下：



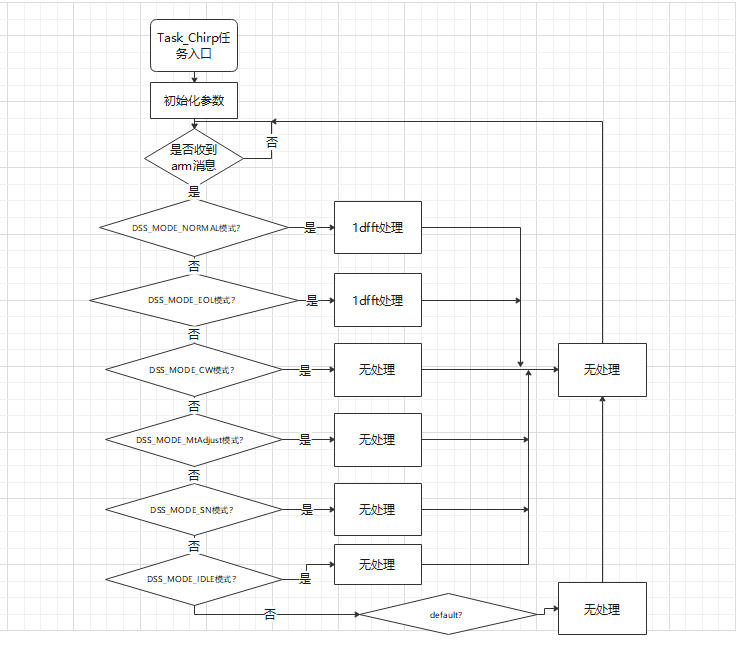


## 5.3任务Task\_Chirp

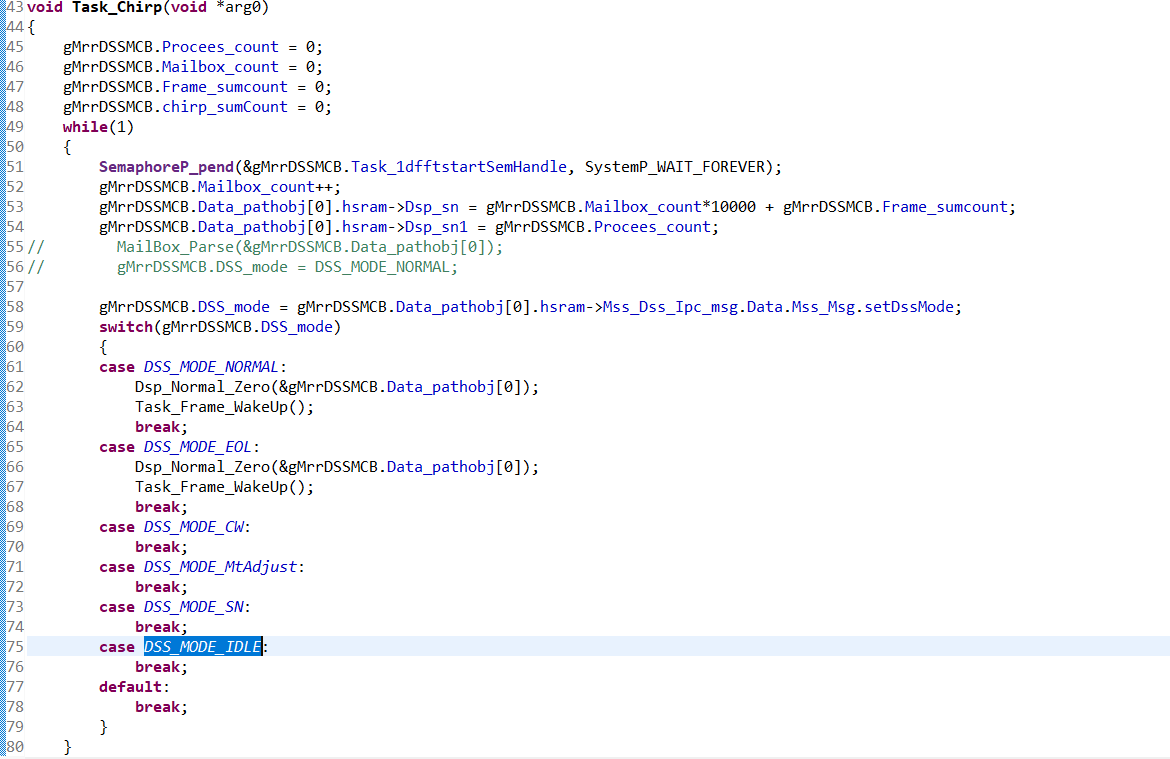
（1）任务Task\_Chirp的功能：根据雷达模式处理数据，雷达一共有6种模式，在本任务中，每个模式实现功能如下表：

|  |  |
| --- | --- |
| **模式** | **实现功能** |
| DSS\_MODE\_NORMAL | 实现256chirp的1dFFT运算 |
| DSS\_MODE\_EOL | 实现256chirp的1dFFT运算 |
| DSS\_MODE\_CW | 暂无功能 |
| DSS\_MODE\_MtAdjust | 暂无功能 |
| DSS\_MODE\_SN | 暂无功能 |
| DSS\_MODE\_IDLE | 暂无功能 |

（2）任务逻辑



（3）任务代码



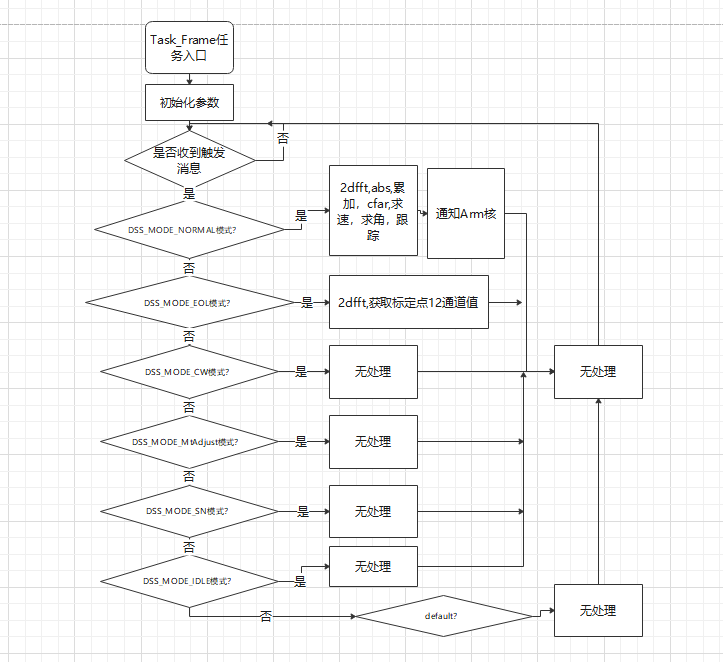


## 5.4任务Task\_Frame

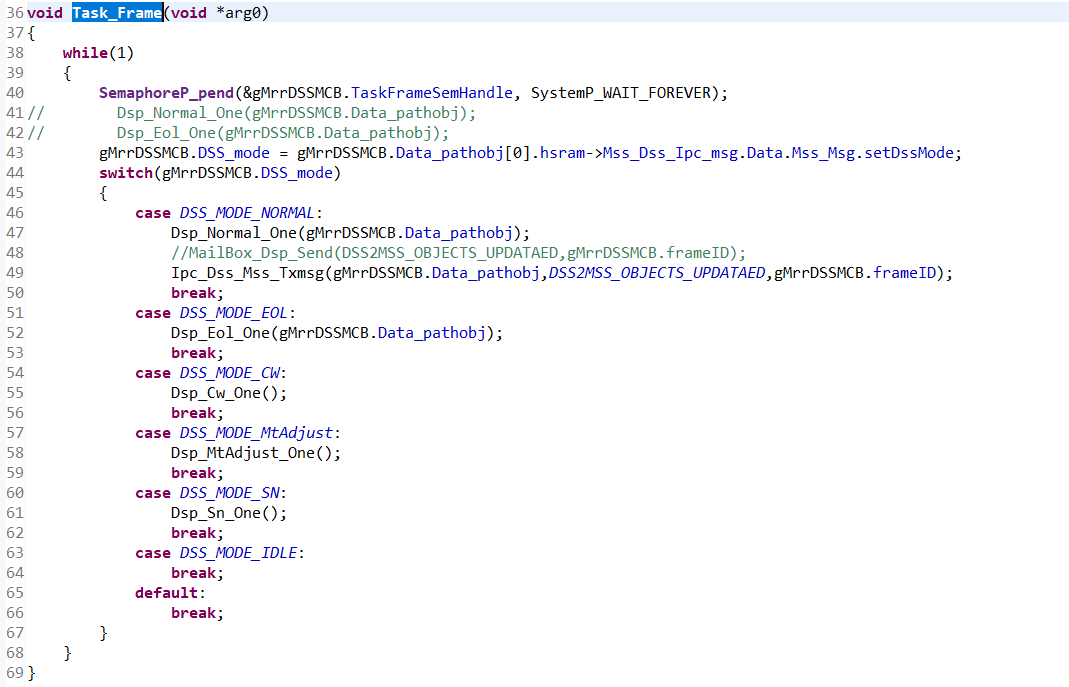
（1）任务Task\_Frame的功能：根据雷达模式处理数据，雷达一共有6种模式，在本任务中，每个模式实现功能如下表：

|  |  |
| --- | --- |
| **模式** | **实现功能** |
| DSS\_MODE\_NORMAL | 实现2dfft,abs,累加，CFAR，求速，求角，跟踪 |
| DSS\_MODE\_EOL | 实现2dfft,abs,获得标定点的12通道数据 |
| DSS\_MODE\_CW | 暂无功能 |
| DSS\_MODE\_MtAdjust | 暂无功能 |
| DSS\_MODE\_SN | 暂无功能 |
| DSS\_MODE\_IDLE | 暂无功能 |

（2）任务逻辑



（3）任务代码



# 6 ADC采样配置

1. ADC采样率配置为25Mhz,采样点数为1024个。
2. 实数采样，16位采样。
3. 每个FRAME一共256个chirp，每个chirp的采样数据放在ADCBUFF中的相同位置，所每个chirp采样完毕要及时处理ADC数据。
4. ADC数据摆放如下表，每个CHIRP产生4个通道的数据。

|  |  |
| --- | --- |
| **Start\_address:** 0xA500\_0000U | RX0\_data(512 samples,int16:2bytes) |
| **Start\_address:** 0xA500\_1000U | RX1\_data(512 samples, int16:2bytes) |
| **Start\_address:** 0xA500\_2000U | RX2\_data(512 samples, int16:2bytes) |
| **Start\_address:** 0xA500\_3000U | RX3\_data(512 samples, int16:2bytes) |

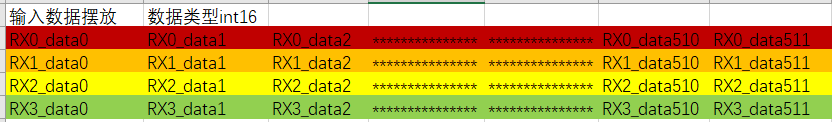
# FRAME数据处理

## frame数据处理中采用的基本算法实现介绍

1. FFT算法（1D）

1DFFT的计算长度为512个实数点采样长度，具体计算采用硬件加速器进行计算，所以不用关注具体实现，只需配置好输入，输出数据参数及格式，使能硬件加速的FFT模式并配置好FFT计算长度即可。

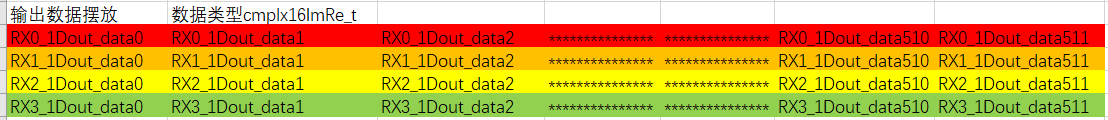
**输入数据描述**：输入数据类型为16位实数，数据总量为4个RX的ADC通道数据，每个通道数据包含512个实数采样点。数据摆放格式如下图：



硬件加速器输入参数配置如下：

|  |  |
| --- | --- |
| 参数 | 配置 |
| source.srcRealComplex | HWA\_SAMPLES\_FORMAT\_REAL（实数） |
| source.srcWidth | HWA\_SAMPLES\_WIDTH\_16BIT（16位宽度） |
| source.srcSign | HWA\_SAMPLES\_SIGNED（有符号） |
| .source.srcScale | 5（输入数据点右移5位） |
| source.srcAcnt | obj->Range\_Parameter.numAdcSamples – 1（512 - 1） |
| source.srcAIdx | sizeof(int16\_t) （2） |
| source.srcBcnt | obj->Antennas\_Parameter.numRxAntennas-1（4 - 1） |
| source.srcBIdx | obj->Range\_Parameter.numAdcSamples \* sizeof(int16\_t)（512\*2） |

**输出数据描述**：输出数据类型为16位复数（cmplx16ImRe\_t,实部类型为int16，虚部类型为int16,虚部在前实部在后），数据总量为4个RX的1DFFT结果数据，每个通道数据包含512个复数点。数据摆放格式如下：



硬件加速器输出参数配置如下：

|  |  |
| --- | --- |
| 参数 | 配置 |
| dest.dstRealComplex | HWA\_SAMPLES\_FORMAT\_COMPLEX（复数） |
| dest.dsWidth | HWA\_SAMPLES\_WIDTH\_16BIT（16位宽度） |
| dest.dsSign | HWA\_SAMPLES\_SIGNED（有符号） |
| dest.dstScale | 1（输出数据点右移1位） |
| dest.dstAcnt | obj->Range\_Parameter.numAdcSamples – 1（512 - 1） |
| dest.dstAIdx | sizeof(int32\_t) （4） |
| dest.dstBIdx | obj->Range\_Parameter.numAdcSamples \* sizeof(int32\_t)（512\*2） |

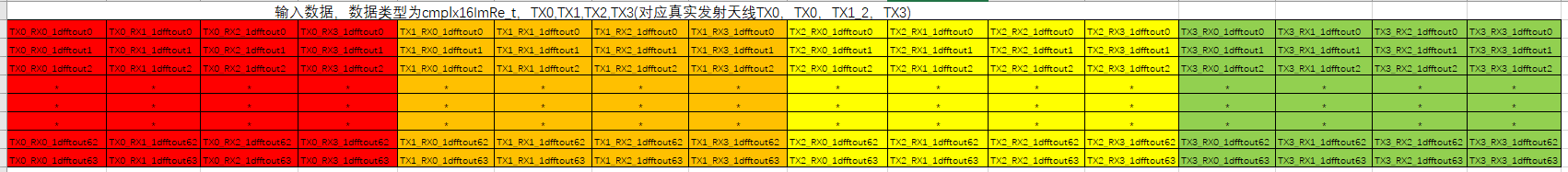
**硬件加速器模式配置如下**：

|  |  |
| --- | --- |
| 参数 | 配置 |
| fftMode.fftEn | FFT1D\_FFT\_ENABLE（使能FFT） |
| fftMode.fftSize | 9（log2(512),512为采样点长度） |
| fftMode.windowEn | 1（使能窗） |
| fftMode.windowStart | 0 （窗的起始位置相对于内存位置的偏移为0） |
| fftMode.winSymm | 1（窗是对称的，内存存储的窗只需一半） |

1. FFT算法（2D）

2DFFT的计算长度为64个复数点长度，具体计算采用硬件加速器进行计算，所以不用关注具体实现，只需配置好输入，输出数据参数及格式，使能硬件加速的FFT模式并配置好FFT计算长度即可。

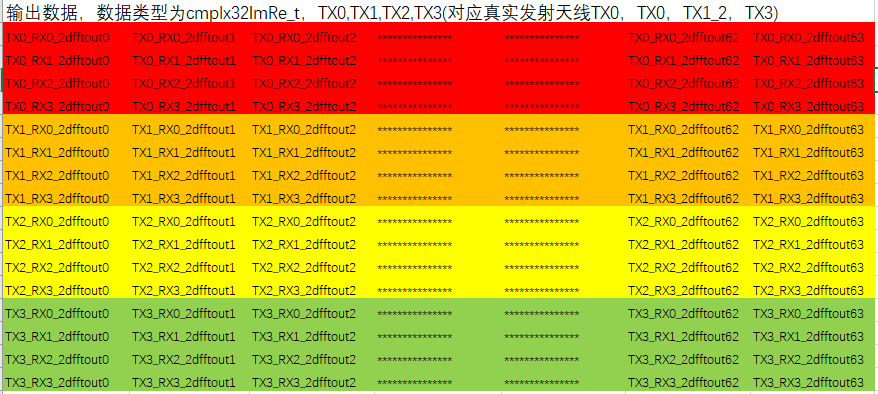
**输入数据描述**：输入数据类型为16位虚部和16位实部组成的复数（cmplx16ImRe\_t,实部类型为int16，虚部类型为int16,虚部在前实部在后），数据总量为16个虚拟通道（时分，4TX，4RX）的1DFFT结果数据，每个通道数据包含64个复数点。数据摆放格式如下图：



硬件加速器输入参数配置如下：

|  |  |
| --- | --- |
| 参数 | 配置 |
| source.srcRealComplex | HWA\_SAMPLES\_FORMAT\_COMPLEXL（复数） |
| source.srcWidth | HWA\_SAMPLES\_WIDTH\_16BIT（16位宽度） |
| source.srcSign | HWA\_SAMPLES\_SIGNED（有符号） |
| .source.srcScale | 6（输入数据点右移6位） |
| source.srcAcnt | obj->Velocity\_Parameter.numDopplerBins- 1（64 - 1） |
| source.srcAIdx | obj->Antennas\_Parameter.numRxAntennas \* obj->Antennas\_Parameter.numTxAntennas\*sizeof(cmplx16ImRe\_t) （4\*4\*4） |
| source.srcBcnt | obj->Antennas\_Parameter.numRxAntennas \* obj->Antennas\_Parameter.numTxAntennas -1（4\*4 - 1） |
| source.srcBIdx | sizeof(cmplx16ImRe\_t)（4） |

**输出数据描述**：输出数据类型为32位复数（cmplx32ImRe\_t,实部类型为int32，虚部类型为int32,虚部在前实部在后），数据总量为16个虚拟通道的2DFFT结果数据，每个通道数据包含64个复数点。数据摆放格式如下：



硬件加速器输出参数配置如下：

|  |  |
| --- | --- |
| 参数 | 配置 |
| dest.dstRealComplex | HWA\_SAMPLES\_FORMAT\_COMPLEX（复数） |
| dest.dsWidth | HWA\_SAMPLES\_WIDTH\_32BIT（32位宽度） |
| dest.dsSign | HWA\_SAMPLES\_SIGNED（有符号） |
| dest.dstScale | 1（输出数据点右移1位） |
| dest.dstAcnt | obj->Velocity\_Parameter.numDopplerBins-1（64 - 1） |
| dest.dstAIdx | sizeof(cmplx32ImRe\_t) （8） |
| dest.dstBIdx | obj->Velocity\_Parameter.numDopplerBins \* sizeof(cmplx32ImRe\_t)（64\*8） |

**硬件加速器模式配置如下**：

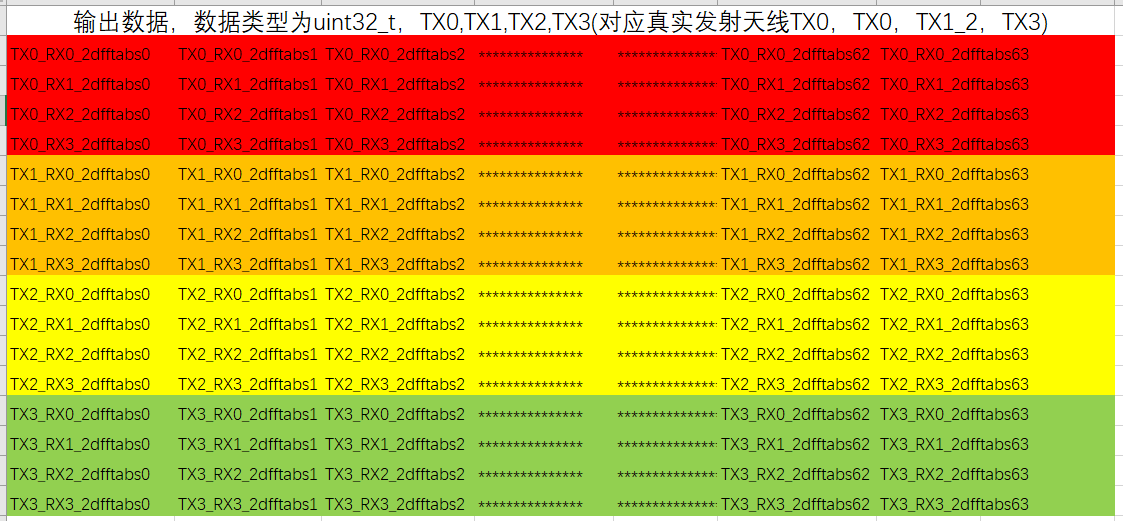
|  |  |
| --- | --- |
| 参数 | 配置 |
| fftMode.fftEn | FFT1D\_FFT\_ENABLE（使能FFT） |
| fftMode.fftSize | 6（log2(64),64为采样点长度） |
| fftMode.windowEn | 1（使能窗） |
| fftMode.windowStart | (obj->Range\_Parameter.numRangeBins/2 +1) （窗的起始位置相对于内存位置的偏移为0） |
| fftMode.winSymm | 0（窗是不对称的，内存存储的窗完整） |

1. ABS算法

Abs也是使用加速器来实现的，具体输入配置与上节2DFFT相同，模式配置也相同，只是在输出数据配置上有区别：

**输出配置描述：**输出数据为32位实数（uint32\_t）,数据点数跟2DFFT结果相同（输出数据为2DFFT输出数据点的ABS结果）。

数据摆放如下图



硬件加速器输出参数配置如下：

|  |  |
| --- | --- |
| 参数 | 配置 |
| dest.dstRealComplex | HWA\_SAMPLES\_FORMAT\_REAL（实数） |
| dest.dsWidth | HWA\_SAMPLES\_WIDTH\_32BIT（32位宽度） |
| dest.dsSign | HWA\_SAMPLES\_UNSIGNED（无符号） |
| dest.dstScale | 1（输出数据点右移1位） |
| dest.dstAcnt | obj->Velocity\_Parameter.numDopplerBins-1（64 - 1） |
| dest.dstAIdx | sizeof(uint32\_t) （4） |
| dest.dstBIdx | obj->Velocity\_Parameter.numDopplerBins \* sizeof(uint32\_)（64\*4） |

**硬件加速器模式配置如下**：

|  |  |
| --- | --- |
| 参数 | 配置 |
| fftMode.fftEn | FFT1D\_FFT\_ENABLE（使能FFT） |
| fftMode.fftSize | 6（log2(64),64为采样点长度） |
| fftMode.windowEn | 1（使能窗） |
| fftMode.windowStart | (obj->Range\_Parameter.numRangeBins/2 +1) （窗的起始位置相对于内存位置的偏移为0） |
| fftMode.winSymm | 0（窗是不对称的，内存存储的窗完整） |
| fftMode.postProcCfg.magLogEn | HWA\_FFT\_MODE\_MAGNITUDE\_ONLY\_ENABLED （使能ABS输出） |

1. 通道累加算法

通道累加算法使用硬件加速器的FFT来实现，实现原理为：对16个通道的ABS值做FFT的运算，会产生16个FFT结果值，取16个结果值当中的第一个结果值的实数部分，此值即为16个通道值得相加结果。

**输入描述**：输入数据为2DFFT的ABS结果，数据类型为uint32\_t ,包含16个虚拟通道的数据，每个通道包含64个实数值。具体摆放参考ABS的输出摆放图。

硬件加速器输入配置如下

|  |  |
| --- | --- |
| 参数 | 配置 |
| source.srcRealComplex | HWA\_SAMPLES\_FORMAT\_REAL（实数） |
| source.srcWidth | HWA\_SAMPLES\_WIDTH\_32BIT（32位宽度） |
| source.srcSign | HWA\_SAMPLES\_UNSIGNED（有符号） |
| .source.srcScale | 3（输入数据点右移3位） |
| source.srcAcnt | obj->Antennas\_Parameter.numRxAntennas \* obj->Antennas\_Parameter.numTxAntennas- 1（16- 1） |
| source.srcAIdx | obj->Velocity\_Parameter.numDopplerBins \* sizeof(uint32\_t)（64\*4） |
| source.srcBcnt | obj->Velocity\_Parameter.numDopplerBins -1（64 - 1） |
| source.srcBIdx | sizeof(uint32\_t)（4） |

**输出描述**：输出数据类型为uint32\_t,输出数据为64个数据点。

输出数据摆放如下图



硬件加速器输出数据配置如下：

|  |  |
| --- | --- |
| 参数 | 配置 |
| dest.dstRealComplex | HWA\_SAMPLES\_FORMAT\_REAL（实数） |
| dest.dsWidth | HWA\_SAMPLES\_WIDTH\_32BIT（32位宽度） |
| dest.dsSign | HWA\_SAMPLES\_UNSIGNED（无符号） |
| dest.dstScale | 0（输出数据点右移0位） |
| dest.dstAcnt | 1-1（1 - 1） |
| dest.dstAIdx | sizeof(uint32\_t) （4） |
| dest.dstBIdx | sizeof(uint32\_)（4） |

**硬件加速器配置如下：**

|  |  |
| --- | --- |
| 参数 | 配置 |
| fftMode.fftEn | FFT1D\_FFT\_ENABLE（使能FFT） |
| fftMode.fftSize | 4（log2(16),16为采样点长度） |
| fftMode.windowEn | 0（不使能窗） |
| fftMode.windowStart | 0（窗的起始位置相对于内存位置的偏移为0） |
| fftMode.winSymm | 0（窗是不对称的，内存存储的窗完整） |
| fftMode.butterflyScaling | 0x3ff |

1. CFAR算法（doppler维）

采用CA\_CFAR。

1. CFAR算法（Range维）

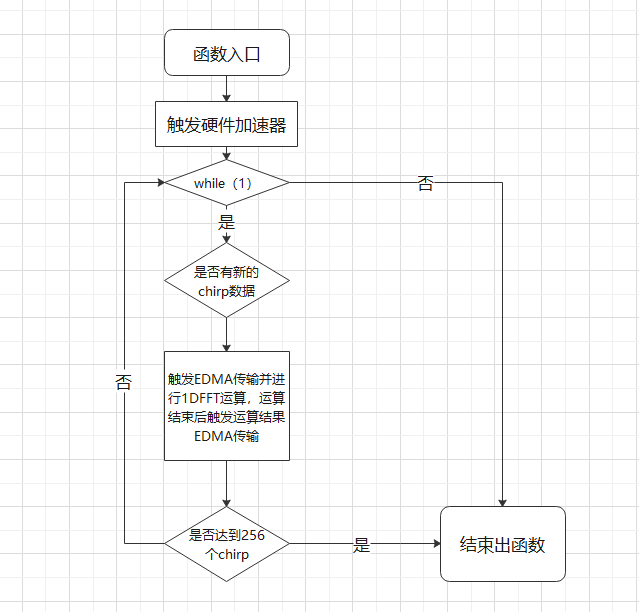
采用CA\_CAFR。

## frame数据处理介绍

FRAME数据处理主要实现分成了两块，chirp数据1DFFT处理部分和目标检测滤波部分。

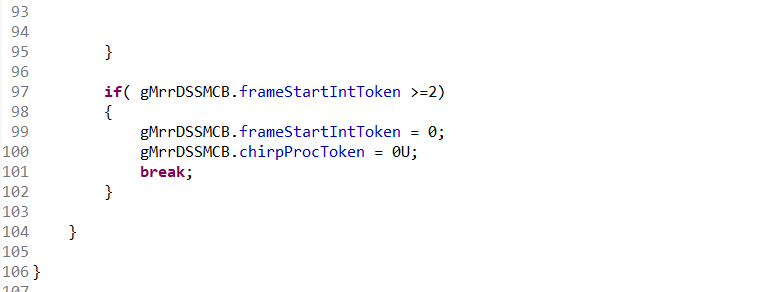
1. chirp数据1DFFT处理

**程序处理逻辑：**



**代码实现如下：**



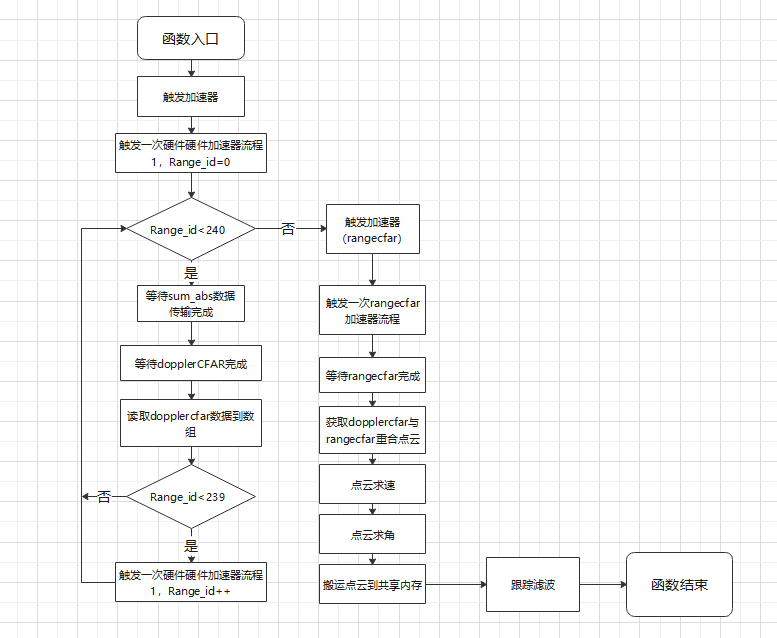


**1DFFT结果摆放位置：**



1. 目标检测，距离，速度，角度及滤波过程

**主处理逻辑：**



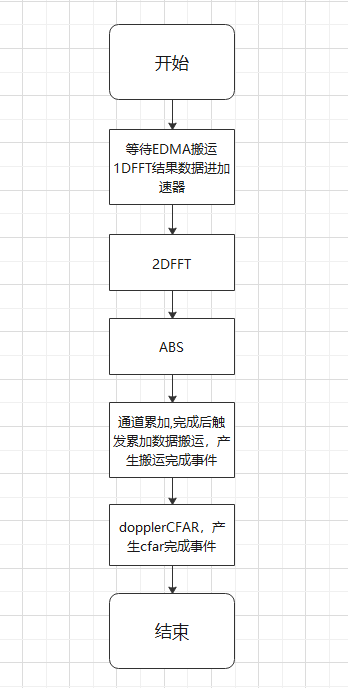
**主处理逻辑实现代码如下：**







**主处理逻辑当中的加速器流程1：**主要包含2DFFT运算，通道累加，dopplerCAFR等功能。逻辑如下图：



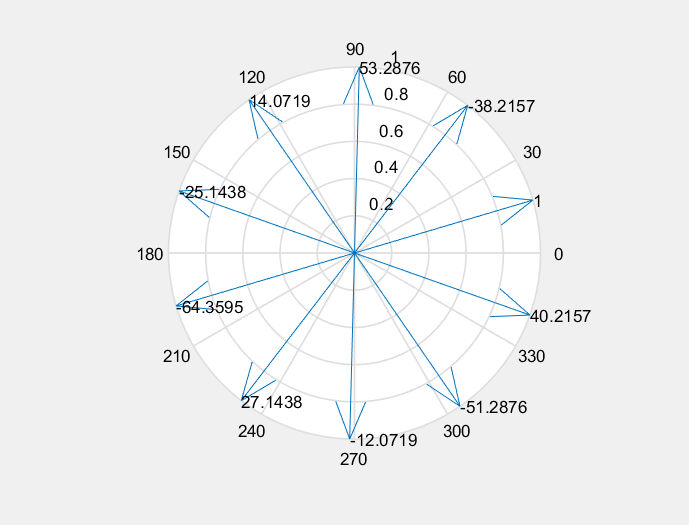
# 8速度解模糊

## 8.1速度解模糊原理

由波形特点可知，TX0连续发了两次波（我这里把两个连续TX0的第一个称TXA，第二个称TXB），假设有一个目标速度为v,当检测到这个目标时，在TXA的通道累加图上我们会的到一个目标的dopplerbinId序号。在累加图的对应处，TXA与TXB会有对应的通道值（复数）。

假设TXA的通道值为CH\_valueTXA ,TXB的通道值为CH\_valueTXB。让CH\_valueTXB除以CH\_valueTXA得到CH\_valueTXBA,我们使用atan2(CH\_valueTXBA.imag, CH\_valueTXBA.real)可得出一个相位值PhaseBA（TXA与TXB的时间间隔为TC，原理公式得出的是真实相位差，但是根据通道值计算出的相位差范围-pi到pi之间,与真实值相差n倍2pi）。

根据dopplerbinId序号，可以扩展出一组目标存在的可能速度（dopplerbinId乘上速度分辨率，然后再加上m倍2倍最大不模糊速度，m为整数，m取值为-5，-4，-3，-2，-1，0，1，2，3，4），可能速度为v0,v1,v2,v3,v4,v5,v6,v7,v8,v9,v10。真实速度就存在于这13个可能速度当中。将所有可能速度代入公式可得出10个相位值Phase\_x(代表对应的速度序号0，1…….9,10)，相位位置分布如下图（目标速度为1m/s）。



使用公式Weizhi\_cha\_x =( cos(PhaseBA) – cos(Phase\_x))^2 + ( sin(PhaseBA) – sin(Phase\_x))^2得出一组相位位置差值。我们求取Weizhi\_cha\_x当中的最小值，则此时x对应的速度为真实速度（注意：假设Weizhi\_cha\_3最小，则v3为求取的真实速度）。

## 8.2程序实现

**程序实现基本步骤如下**：

1. 初始化获得波形两个连续TX0发送chirp的时间间隔T\_dif\_sum，波长lambda，doppler最大速度MaxVelocity，速度分辨率Vres，MaxVelocity的倒数MaxVelocity\_Inv，lambda的倒数lambda\_Inv，列出不同模糊倍数的速度列表MaxVelocitylist[20](**注**：列出了-9，-8………9,10等倍MaxVelocity速度)。
2. 获得连续两个TX0发送chirp的目标通道值，一共8个复数，TX0\_RX0\_value，TX0\_RX1\_value, TX0\_RX2\_value, TX0\_RX3\_value, TX1\_RX0\_value ,TX1\_RX1\_value, TX1\_RX2\_value, TX1\_RX3\_value。
3. 获得各个RX通道由于时间差导致的相位差(可能模糊)。

TX1\_0\_0value = TX1\_RX0\_value/ TX0\_RX0\_value,

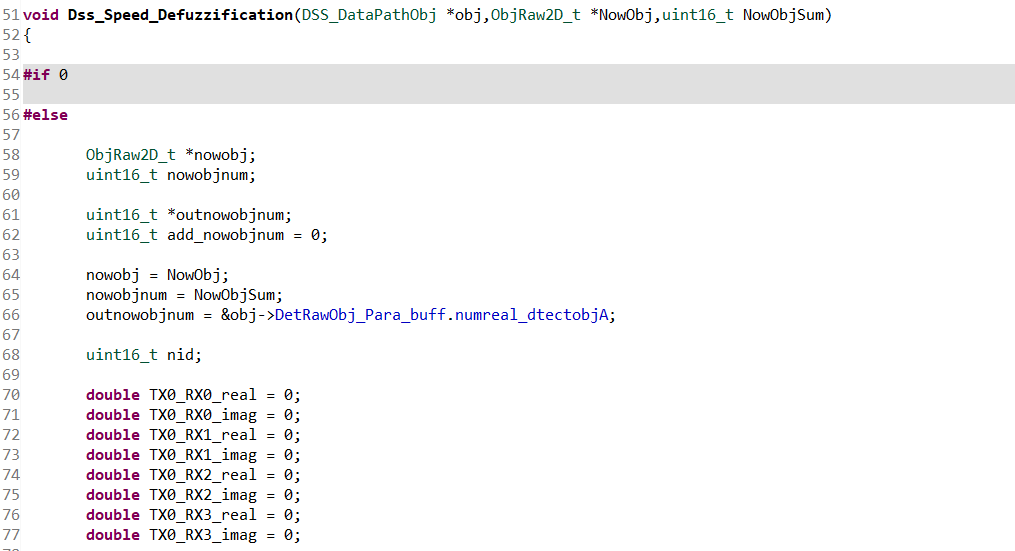
TX1\_0\_1value = TX1\_RX1\_value/ TX0\_RX1\_value,

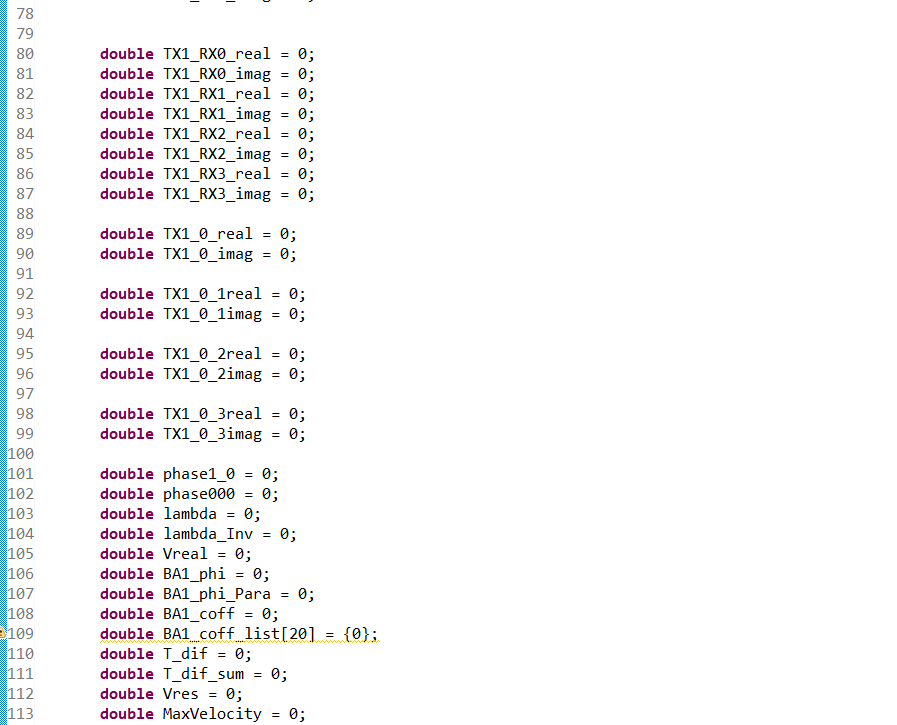
TX1\_0\_2value = TX1\_RX2\_value/ TX0\_RX2\_value,

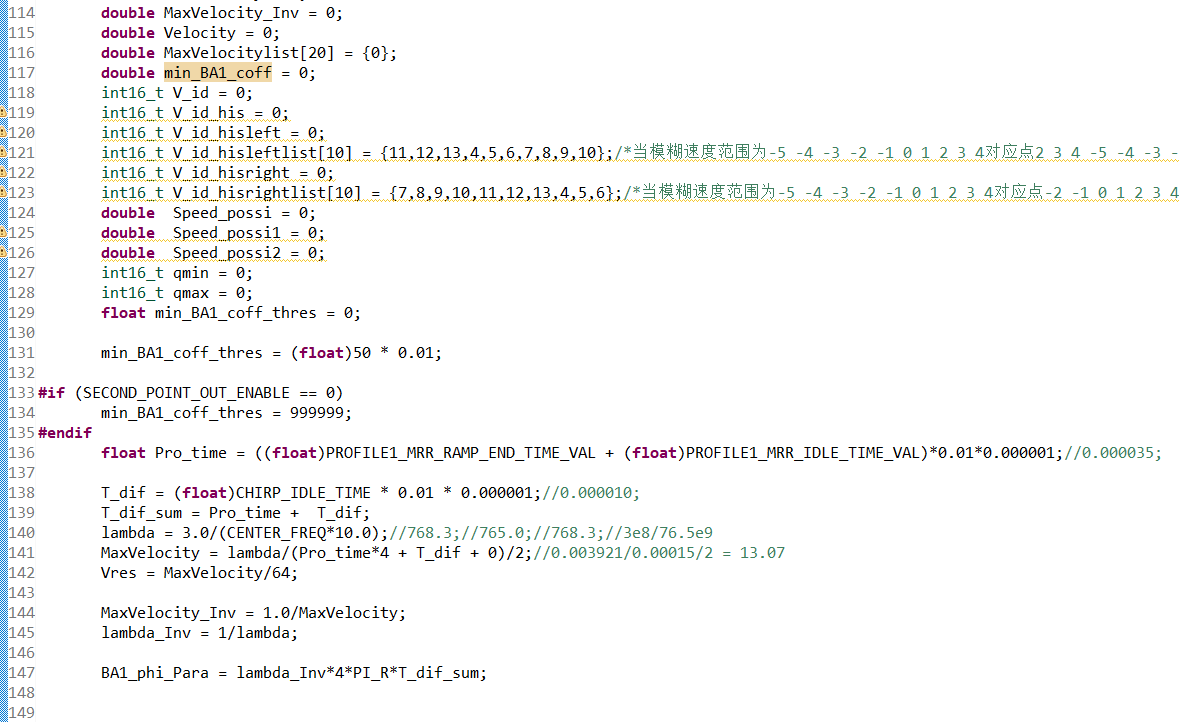
TX1\_0\_3value = TX1\_RX3\_value/ TX0\_RX3\_value,

1. 将四个通道相位差复数值直接相加，再使用atan2得出目标前后chirp探测中的相位差（可能模糊），如：phase1\_0 = atan2((TX1\_0\_0value.imag + TX1\_0\_1value.imag + TX1\_0\_2value.imag + TX1\_0\_3value.imag), (TX1\_0\_0value.imag + TX1\_0\_1value.imag + TX1\_0\_2value.imag + TX1\_0\_3value.imag))。
2. 使用公式Velocity = (double)nowobj[nid].dopplerIdx1 \* Vres求出目标的基本速度，再使用Velocity加上不同模糊倍数的速度MaxVelocitylist[20]得出目标的可能速度，使用这些可能速度带入公式（TC等于T\_dif\_sum）得出相位BA1\_phi。然后再将所有的可能相位BA1\_phi代入BA1\_coff =( cos(BA1\_phi) – cos(phase1\_0))^2 + ( sin(BA1\_phi) – sin(phase1\_0))^2得出点位置差的距离平方BA1\_coff，找到使得BA1\_coff取得最小值的目标速度。
3. 最后对BA1\_coff的最小值min\_BA1\_coff做一个阈值判断，满足则目标成功解出速度，否则未解出速度。

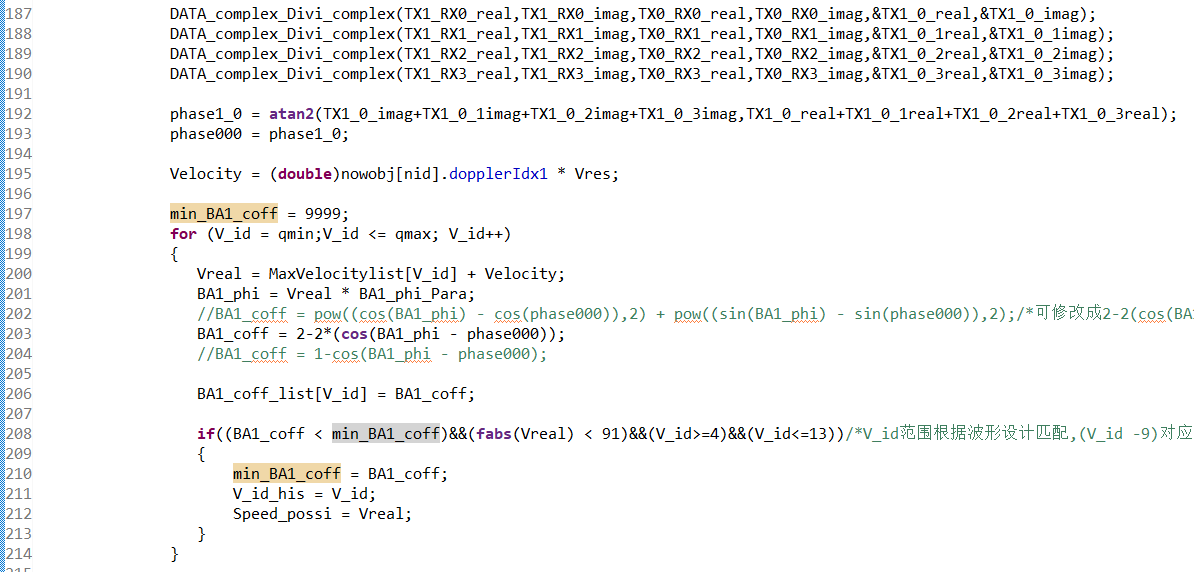
**代码如下**：

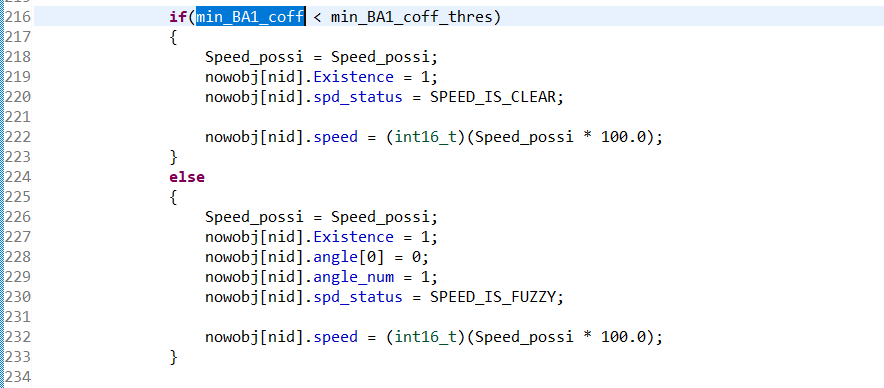


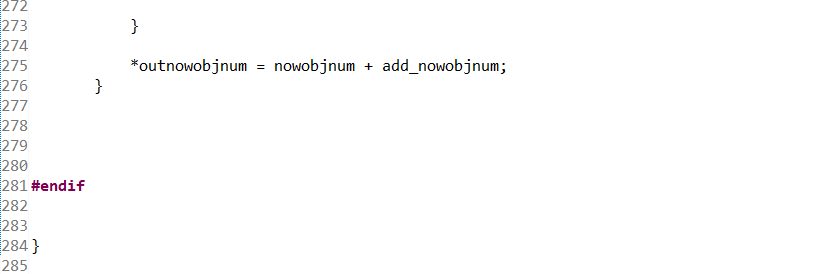






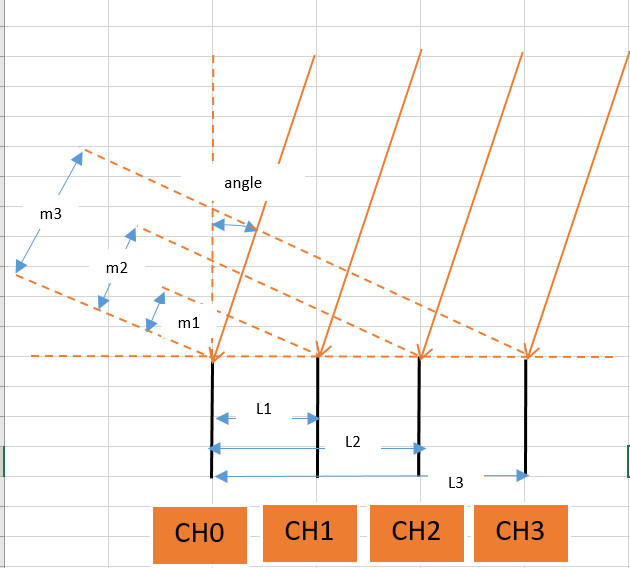






# 9 DBF求角

## 9.1求角原理



（**注：**本项目使用12天线求角，这里只是简单说明以下）

当有一个目标在雷达天线水平角angle处时（我们设此时角度等于负值），4个通道会接收到返回波信号，4个通道的相位差（相对于通道CH0）为0，m1,m2,m3。由上图可以看出m1 = L1\*sind(angle), m2 = L2\*sind(angle)，m3 = L3\*sind(angle)。

我们使用不同角度角度带你anglex(x取值范围为-75°到75°)去生成不同角度下的通道相位差,M1\_x = L1\*sind(anglex), M2\_x = L2\*sind(anglex), M3\_x = L3\*sind(anglex)。使用关系式 A\_x = abs(exp(j\*m1)\*exp(-j\*M1\_x)+ exp(j\*m2)\*exp(-j\*M2\_x)+ exp(j\*m3)\*exp(-j\*M3\_x))。则当anglex 等于angle时会取得A\_x最大值（**注**：角度不模糊的情况，假设通道之间间距为半波长），则可得出实际角度。

## 9.2程序实现

本项目是采用12通道求角，通道间隔数据为M[12] ={0,3.862,4.862,8.724,2.908,6.770,7.770,11.633,5.816,9.679,10.679,14.541}，数值都是半波长得倍数。

程序实现基本步骤如下：

1. 初始化DFT的旋转因子：求角范围为-75°到75°，在这个角度范围内取等间隔的576个角度点，角度间隔ANGE\_RES = （75-（-75））\*Pi/180/576 = 0.0045451。则576个角度点当中的任一个角度可表示为angle\_x = (x-287)\* ANGE\_RES。按照如下公式生成旋转因子(注：x代表角度点，n = 0,1,2,3,4,5…10,11;对应12个通道)

ch\_x\_constreal \_n= cos(sin(angle\_x)\*Pi\*M[n])\*30000;

ch\_x\_constimag \_n= sin(sin(angle\_x)\*Pi\*M[n])\*(-30000);

1. 将角度点对应的校准值加到旋转因子中，假设(calidata\_real\_x\_n, calidata\_imag\_x\_n)为指定角度点对应通道的校准值，则按照下列公式加入：

Cosvv = ch\_x\_constreal \_0 \* calidata\_real\_x\_0 - ch\_x\_constimag \_0 \* calidata\_imag\_x\_0;

Sinvv = ch\_x\_constreal \_0 \* calidata\_imag\_x\_0 + ch\_x\_constimag \_0 \* calidata\_real\_x\_0;

ch\_x\_constreal \_0 = Cosvv;

ch\_x\_constimag\_0 = sinvv;

Cosvv = ch\_x\_constreal \_1 \* calidata\_real\_x\_1 - ch\_x\_constimag \_1\* calidata\_imag\_x\_1;

Sinvv = ch\_x\_constreal \_1 \* calidata\_imag\_x\_1 + ch\_x\_constimag \_1 \* calidata\_real\_x\_1;

ch\_x\_constreal \_1 = Cosvv;

ch\_x\_constimag\_1= Sinvv;

Cosvv = ch\_x\_constreal \_2 \* calidata\_real\_x\_2 - ch\_x\_constimag \_2 \* calidata\_imag\_x\_2;

Sinvv = ch\_x\_constreal \_2 \* calidata\_imag\_x\_2 + ch\_x\_constimag \_2 \* calidata\_real\_x\_2;

ch\_x\_constreal \_2 = Cosvv;

ch\_x\_constimag\_2 = sinvv;

Cosvv = ch\_x\_constreal \_3 \* calidata\_real\_x\_3 - ch\_x\_constimag \_3\* calidata\_imag\_x\_3;

Sinvv = ch\_x\_constreal \_3 \* calidata\_imag\_x\_3 + ch\_x\_constimag \_3 \* calidata\_real\_x\_3;

ch\_x\_constreal \_3 = Cosvv;

ch\_x\_constimag\_3= Sinvv;

此处省略余下的8个通道计算，余下8个通道计算公式与上相同。

1. 设目标通道值为inputdata\_n(n=0,1,2…10,11,对应12个通道)，由于是时分波形，需要将通道4，5，6，7，8，9，10，11一共8个通道进行速度补偿。按照公式(定义phase\_ca = 4\*Pi\*TC/bochang=320235.733)可得4，5，6，7通道的补偿值为Bu0\_real = cos(speed\*phase\_ca),Bu0\_imag = sin(speed\*phase\_ca)\*(-1);通道8，9，10，11的补偿值为Bu1\_real = cos(speed\*phase\_ca\*2),Bu1\_imag = sin(speed\*phase\_ca\*2)\*(-1);按照下列公式补偿：

通道4，5，6，7为

Cosv =inputdata\_n.real \* Bu0\_real - inputdata\_n.imag \* Bu0\_imag;

Sinv = inputdata\_n.real \* Bu0\_imag + inputdata\_n.imag \* Bu0\_real;

inputdata\_n.real = Cosv;

inputdata\_n.imag = Sinv;

通道8，9，10，11为

Cosv =inputdata\_n.real \* Bu1\_real - inputdata\_n.imag \* Bu1\_imag;

Sinv = inputdata\_n.real \* Bu1\_imag + inputdata\_n.imag \* Bu1\_real;

inputdata\_n.real = Cosv;

inputdata\_n.imag = Sinv;

(4) DFT计算，一共需要计算576个角度点，但是为了减少运算，这里从中抽取192个点进行运算。例如原来576个点是要计算第0，1，2…574，575点，现在抽取计算第0，3，6，9…573点，我们将计算公式如下(x = 角度点)：

CH0\_real = inputdata\_0.real \* ch\_x\_constreal \_0 - inputdata\_0.imag \* ch\_x\_constimag \_0;

CH0\_imag = inputdata\_0.real \* ch\_x\_constimag \_0 + inputdata\_0.imag \* ch\_x\_constreal \_0;

CH1\_real = inputdata\_1.real \* ch\_x\_constreal \_1 - inputdata\_1.imag \* ch\_x\_constimag \_1;

CH1\_imag = inputdata\_1.real \* ch\_x\_constimag \_1 + inputdata\_1.imag \* ch\_x\_constreal \_1;

此处省略10个通道的公式，通道计算跟上面两个点相同。

CH\_real = CH0\_real + CH1\_real……+ CH10\_real + CH11\_real;

CH\_imag = CH0\_imag + CH1\_imag……+ CH10\_imag + CH11\_imag;

CH\_x\_mag = CH\_real \* CH\_real + CH\_imag \* CH\_imag;

(5)找取192个角度点赋值当中对应的最大值和第二大值，幅值为x1\_mag,x2\_mag设对应的角度点为x1,x2。

(6)如果x2\_mag大于x1\_mag的百分之80，则两个角度点都有效。

1. 在角度点x1附近选择（(x1-8）, (x1-7）…x1…(x1+7),(x1+8)）等角度点按照第四步计算公式进行计算，再得出最大幅值对应的角度点x11.如果有第二个角度点按照同样得操作计算。（注：括号中所选的角度点一定要在0，1，2…574,575之间，超过则去除不要）
2. 按照公式：angle = （x11-287）\* 75/288得出最终角度。