/Falcon_18xx_dss/Source/Asw/Spm/Cfar/Cfar.c

```
uint32_t DSS_doppler_cfarCa_SO_dBwrap_withSNR(const uint16_t* inp,
                                              uint16_t* out,
                                              uint16_t* outSNR,
                                              uint32_t len,
                                              uint32_t thresholdScale, uint32_t noiseDivShift,
                                              uint32_t guardLen, uint32_t noiseLen)
{
```

CA代表单元平均(用的SO，估计照抄代码写成了_cfarCa_SO_)，dB代表要求输入数据幅值转换为dB，wrap代表求平均底噪时CUT（cell under test）两边都要算。

doppler说明这个CFAR检测是在当前距离的列数据上检测的，在多普勒仓采样数据中搜索峰值。

函数描述：对16位无符号输入向量(CFAR-SO)执行CFAR。假设输入值为对数，因此剪切样本和噪声样本之间的比较是相加的而不是相乘的。所有切割的比较都是双边的(需要时环绕)。

接口名称	说明	注
inp	输入数组指针	unsigned short格式
out	输出数组指针	有目标标记为当前单元数，没目标则不输出，检测到的峰的指数
outSNR		信噪比
len	输入数据个数	
thresholdScale	阈值范围/门限因子	用于将待测单元(CUT)与噪声单元之和进行比较:
noiseDivShift	噪声漂移	噪声和/(2^(noisedivshift))+thresholdScale。
guardLen	保护单元长度	单边长度
noiseLen	噪底单元长度	单边长度
	返回值	Number of detected peaks

SO滑窗检测：

为什么要滑窗：滑窗过程表格

sumLeft	CUT	sumRight
inp[20]~inp[27]	inp[0]	inp[5]~inp[12]
inp[21]~inp[28]	inp[1]	inp[6]~inp[13]
inp[22]~inp[29]	inp[2]	inp[7]~inp[14]
inp[23]~inp[30]	inp[3]	inp[8]~inp[15]
inp[24]~inp[31]	inp[4]	inp[9]~inp[16]
inp[25]~inp[31] + inp[0]	inp[5]	inp[10]~inp[17]
inp[26]~inp[31] + inp[0]~inp[1]	inp[6]	inp[11]~inp[18]
inp[27]~inp[31] + inp[0]~inp[2]	inp[7]	inp[12]~inp[19]
inp[28]~inp[31] + inp[0]~inp[3]	inp[8]	inp[13]~inp[20]
inp[29]~inp[31] + inp[0]~inp[4]	inp[9]	inp[14]~inp[21]
inp[30]~inp[31] + inp[0]~inp[5]	inp[10]	inp[15]~inp[22]
inp[31] + inp[0]~inp[6]	inp[11]	inp[16]~inp[23]
inp[0]~inp[7]	inp[12]	inp[17]~inp[24]
inp[1]~inp[8]	inp[13]	inp[18]~inp[25]
inp[2]~inp[9]	inp[14]	inp[19]~inp[26]
inp[3]~inp[10]	inp[15]	inp[20]~inp[27]
inp[4]~inp[11]	inp[16]	inp[21]~inp[28]
inp[5]~inp[12]	inp[17]	inp[22]~inp[29]
inp[6]~inp[13]	inp[18]	inp[23]~inp[30]
inp[7]~inp[14]	inp[19]	inp[24]~inp[31]

inp[8]~inp[15]	inp[20]	inp[25]~inp[31] + inp[0]
inp[9]~inp[16]	inp[21]	inp[26]~inp[31] + inp[0]~inp[1]
inp[10]~inp[17]	inp[22]	inp[27]~inp[31] + inp[0]~inp[2]
inp[11]~inp[18]	inp[23]	inp[28]~inp[31] + inp[0]~inp[3]
inp[12]~inp[19]	inp[24]	inp[29]~inp[31] + inp[0]~inp[4]
inp[13]~inp[20]	inp[25]	inp[30]~inp[31] + inp[0]~inp[5]
inp[14]~inp[21]	inp[26]	inp[31] + inp[0]~inp[6]
inp[15]~inp[22]	inp[27]	inp[0]~inp[7]
inp[16]~inp[23]	inp[28]	inp[1]~inp[8]
inp[17]~inp[24]	inp[29]	inp[2]~inp[9]
inp[18]~inp[25]	inp[30]	inp[3]~inp[10]
inp[19]~inp[26]	inp[31]	inp[4]~inp[11]

模板?

```
/*initializations  */
 outIdx = 0;
 sumLeft = 0;
 sumRight = 0;
 for (idx = 1; idx <= noiseLen; idx++) ) //求最左侧的noiseLen个单元的和值，
 {
     sumLeft += inp[len - guardLen - idx];
 }

 for (idx = 1; idx <= noiseLen; idx++)//求最右侧的noiseLen个单元的和值，
 {
     sumRight += inp[idx + guardLen];
 }
```

检测第0个单元 (优化的左侧全部超边界算法，不滑窗直接算)

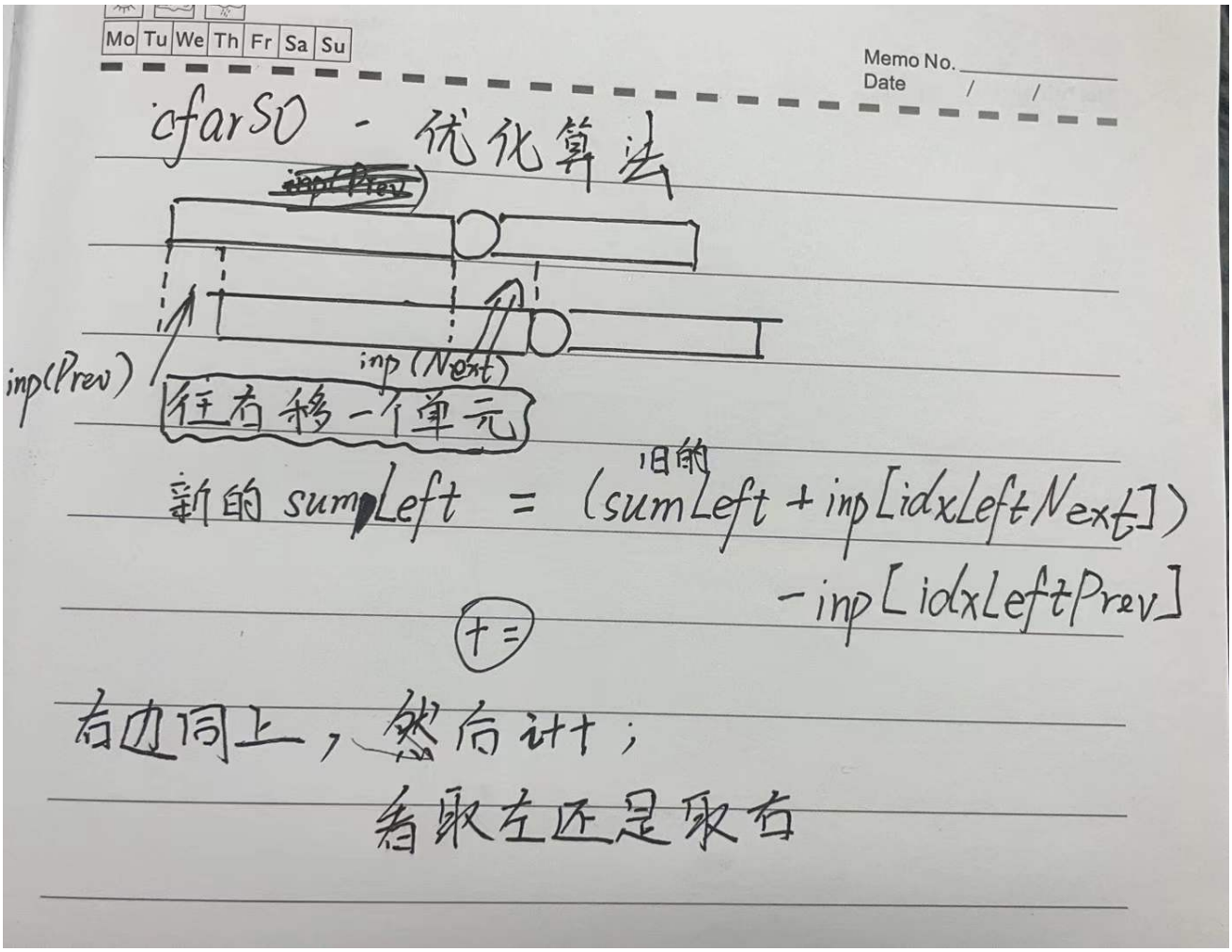
inp[0]与门限阈值做比较，底噪和为sum，sum右移noiseDivShift-1为求平均值

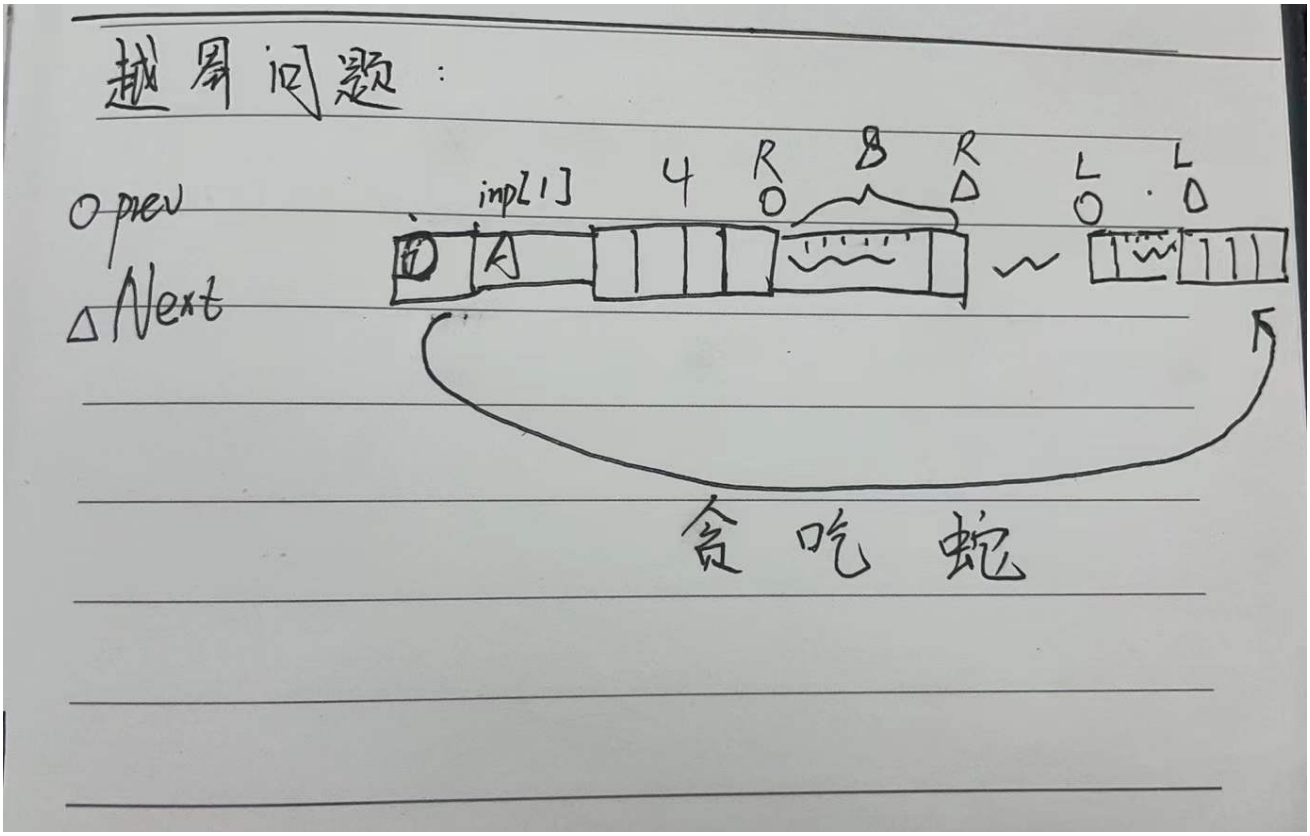
```
    /*CUT 0: */
    sum = (sumLeft < sumRight) ? sumLeft:sumRight;//sum =  sumRight += inp[idx +
guardLen + 1];此时idx从0开始，模板从1开始。
    if ((uint32_t) inp[0] > ((sum >> (noiseDivShift-1)) + thresholdScale))
    {
        out[outIdx] = 0;//发现目标信号标记为0
        outSNR[outIdx] = inp[0]  - (sum >> (noiseDivShift - 1));
        outIdx++;
    }
```

检测第1到第（guardLen）个单元，左侧全部超边界

e.g. len = 32 , guardLen = 4 , noiseLen =8

优化的算法：






```
    /* CUT 1 to guardLen: */
    idxLeftPrev = len - guardLen - noiseLen;    /*e.g. 32-4-8 = 20 */
    idxLeftNext = idxLeftPrev + noiseLen; /*e.g. 28 */
    idxRightPrev = 1 + guardLen;    /*e.g. 1+4=5 */
    idxRightNext = idxRightPrev + noiseLen;   /*e.g. 13 */

    for (idxCUT = 1; idxCUT <= guardLen; idxCUT++)
    {                                                   //sumLeft一直是8个单元的和值，每次for循环比之前的整体往右移动一个单元
        sumLeft += inp[idxLeftNext] - inp[idxLeftPrev];//sumLeft = (sumLeft + inp[idxLeftNext]) - inp[idxLeftPrev];
        idxLeftNext++;
        idxLeftPrev++;
        sumRight += inp[idxRightNext] - inp[idxRightPrev];
        idxRightNext++;
        idxRightPrev++;

        sum = (sumLeft < sumRight) ? sumLeft:sumRight;

        if ((uint32_t) (inp[idxCUT]) > ((sum >> (noiseDivShift - 1)) + thresholdScale))
        {
            out[outIdx] = idxCUT;
            outSNR[outIdx] = inp[idxCUT]  - (sum >> (noiseDivShift - 1));
            outIdx++;
        }
    }
```

剩下的根据滑窗过程表格写代码（类似）

//* CUT guardLen+1 to guardLen+noiseLen: e.g. CUT 5 to 12 *///左侧一部分超边界

//不超边界 e.g. CUT 13 to 19

//* noiseLen number of CUTs before the last guardLen CUTs. e.g. CUT 20 to 27 *///右侧一部分超边界

//* The last guardLen number of CUTs *///右侧全部超边界