DSP领域最新RISC-V指令集及DSA在无线领域中的创新应用

2025/07/16

李高山 芯昇科技芯片架构师，高级专家，中国移动拔尖人才

目 录

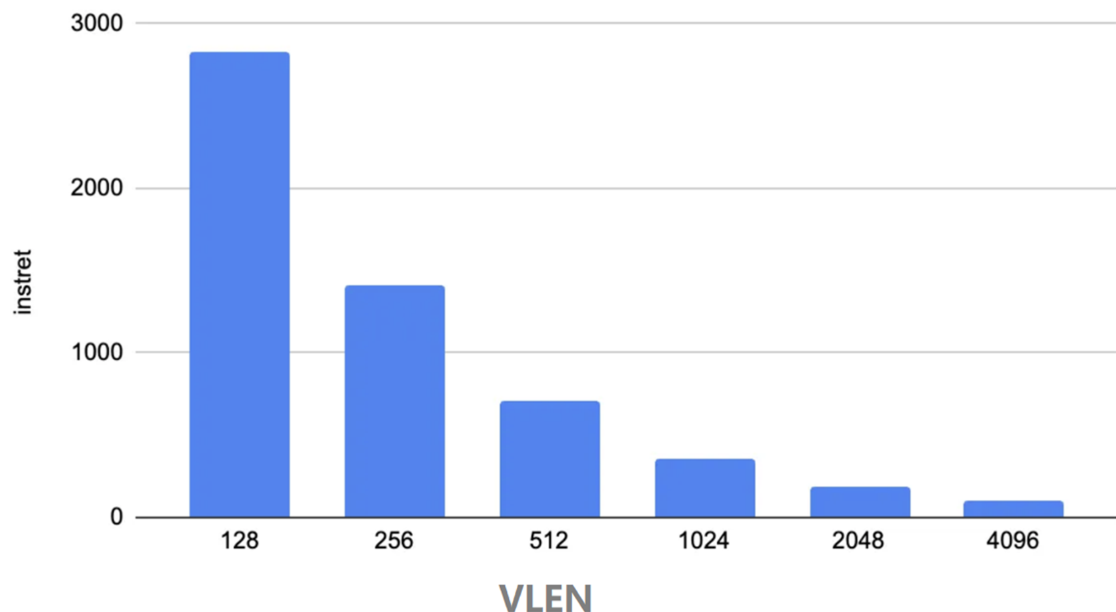
contents

01 兼容RVV的RISC-dsp指令集

02 RISC-dsp DSA的无线应用

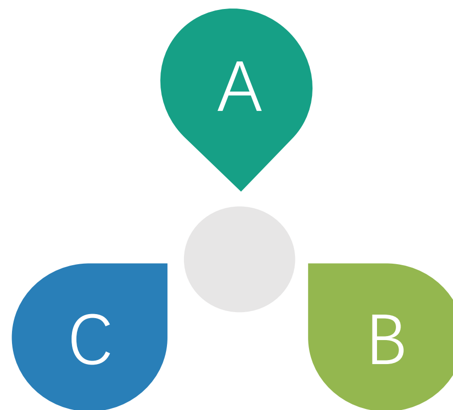
RVV vs. SIMD

与SIMD 扩展（ARM Neon，RISCV P ext，AVX）不同，RVV的VLEN是一个实现参数，并不限于某一个值，除此之外可变长度VL，以及不同的LMUL配置，可以更加高效的支持不同算力场景应用。



减少代码量

超过寄存器长度的数据可以自然的被处理



微架构容易优化

多个矢量寄存器之间的操作方便并行流水处理

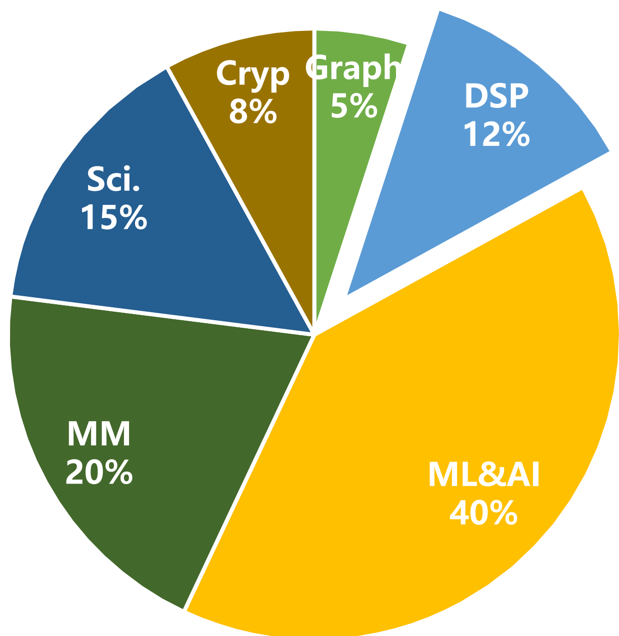
减少循环

一条指令可以处理多个矢量寄存器长度

RVV 1.0缺乏高性能DSP指令

RVV应用场景超过60%集中在AI和多媒体领域，在DSP领域指令效率和效能仍存在明显不足，DSP领域范围涵盖无线信号处理、神经网络、音视频、雷达、汽车（传感器滤波/融合）等领域。

RVV实际应用部署统计



注：以上比例基于2023-2025年产业公开信息（芯片发布、性能测试、开源项目适配进展），综合技术适配成熟度与市场落地规模估算得出。实际占比可能随RVV生态发展动态调整

RVV SIG筹备中的DSP TG工作重点

增强型定点算术指令

融合定点数的饱和、舍入操作、更宽的乘累加寄存器，改善精度和动态范围

复数算术指令

复数的加法、乘法、乘累加、点积、蝶形运算，信号处理领域最基本最重要的操作

DSP数据类型转换指令

支持新的DSP数据类型和传统数据类型之间的转换

DSP统一硬件加速接口指令

FFT、矩阵分解、非线性函数加速指令

RISC-dsp-w指令集

- 结合RISC-V开放、可扩展特性，RVV相比传统SIMD架构更高效灵活，根据RVV在DSP领域的不足定义了RISC-dsp指令集。RISC-dsp是精简指令集架构的矢量DSP指令集，完全兼容RVV1.0标准。
- RISC-dsp-w是RISC-DSP指令集子集，主要面向无线信号处理，性能优势明显。
- RISC-dsp-w指令集在RISC-V工委会以团体标准立项，同时也是RVV SIG目前正在筹备的DSP TG的参考指令集。RISC-dsp-w指令集定义可通过Gitee访问。
- RISC-dsp指令集也将支持神经网络、音视频、雷达等领域。（标准备选扩展名为"Zvsp", "Zvspw", "Zvspnn"...)

RISC-dsp-w Gitee

增强定点指令(9条)

融合定点数的饱和、舍入、移位等操作、更宽的乘累加寄存器，改善精度和动态范围

复数算术指令(30条)

复数的加法、乘法、乘累加、共轭等信号处理领域最基本最重要的操作

类型转换指令(13条)

DSP领域常用数据类型到不同数据类型的转换，复数类型，复数压缩浮点类型等

快速非线性指令(5条)

通用可配置的非线性快速运算指令支持各种类型非线性函数1/x, 1/sqrt(x), sqrt(x), artan, log2, log10, sin, cos, 各种类型的激活函数，

硬件加速指令(8条)

各种Radix-2点数的FFT和IFFT

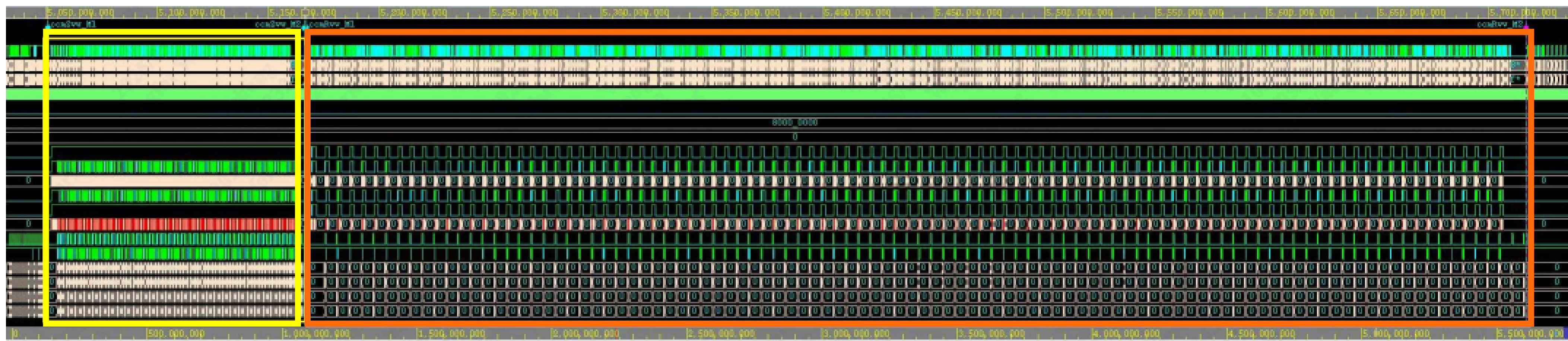
其他增强型指令(8条)

RISC-dsp-w指令集性能优势

RISC-dsp-w指令集在无线信号基本运算中时钟数和代码量相比RVV1.0有4~5倍降低。运行在相同处理器中相比RVV1.0指令集在无线信号处理方面功耗也有明显改善。

硬件时钟统计对比(芯来RISC-dsp VPU, VLEN=1024,DLAN=128)			
测试用例名称	**RISC-dsp时钟数**	**RVV时钟数**	**RVV / RISC-dsp时钟比**
32点共轭复数乘法（×100次)	1867 (硬件仿真器) 1873 (硬件)	8864(硬件仿真器) 8872(硬件)	4.7
64点复数动态定标MAC（×100次)	4247 (硬件仿真器) 4366(硬件)	18928(硬件仿真器) 18936(硬件)	4.5
32宽带平均功率(×100次)	2960(硬件仿真器) 2967(硬件)	13927(硬件仿真器) 13936(硬件)	4.7

共轭复数乘法RISC-dsp-w运行周期　　　　共轭复数乘法RVV1.0 运行周期



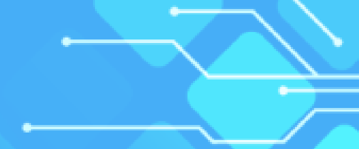
目 录

contents

兼容RVV的RISC-dsp指令集

RISC-dsp DSA的无线应用

RISC-V指令集架构与DSA



RISC-V特性

开放标准： 免授权费、可扩展的指令集（ISA）

模块化设计： 基础指令（RV32/64G）+ 可选扩展（D/C/P/Z等）

定制化潜力： 支持用户自定义指令，多个用户定制指令编码空间

DSA核心理念

领域专用： 为特定负载优化（如AI/无线/存储/视频，等）

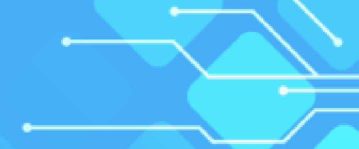
效率优势： 相比通用CPU，在特定领域中的性能可以有成倍的提升（TPU/NPU）

可重构性： RISC-V的灵活ISA天然适配DSA需求

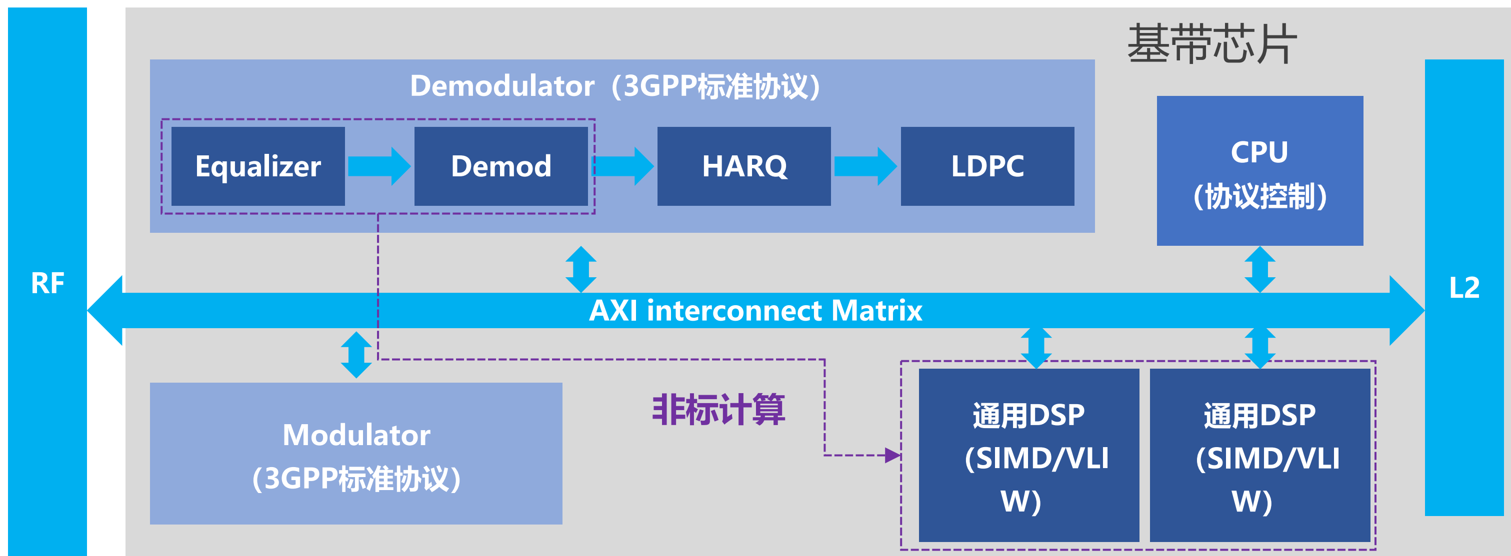
典型案例

G公司 TPU v5的Scalar Unit和Vector Unit采用了RISC-V核，这些RISC-V核提供了强大的标量和向量计算能力，并通过VCIX接口与MXU连接，MUX可以直接访问矢量寄存器。通过这种集成方式，实现了更好的能效比和编程灵活性

RISCV DSA在无线基带信号处理中的应用

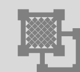


现有基带芯片架构



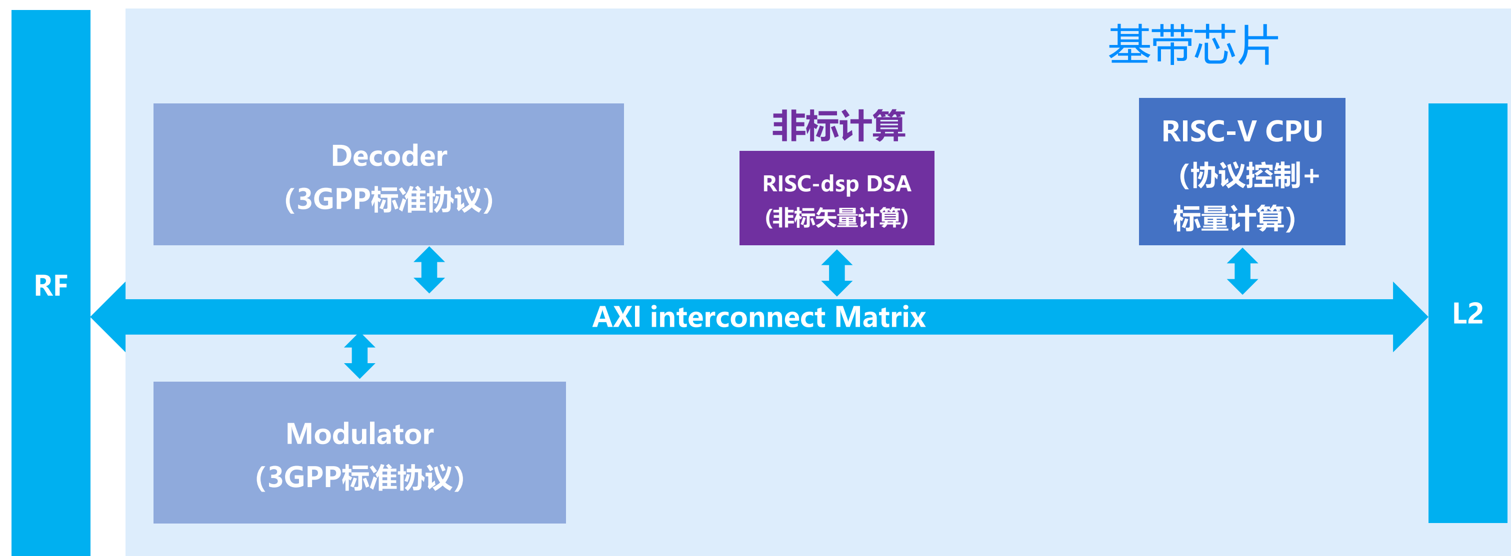
SIMD/VLIW架构的通用DSP处理无线矢量信号在PPA方面与ASIC相比没有优势

国外商用平台无法提供差异化产品（近似的性能及IP成本），无法保障供应链安全



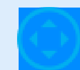
非标算法优化的国产差异化产品目前都是纯ASIC方案，灵活性和迭代速度受到制约，无法兼顾高性能和低成本

理想基带芯片架构



引入低功耗、高性能、极致PPA的矢量信号处理器负责所有的非标矢量运算，在不牺牲PPA的情况下提供足够灵活性

标准协议运算使用低成本、高质量的ASIC IP

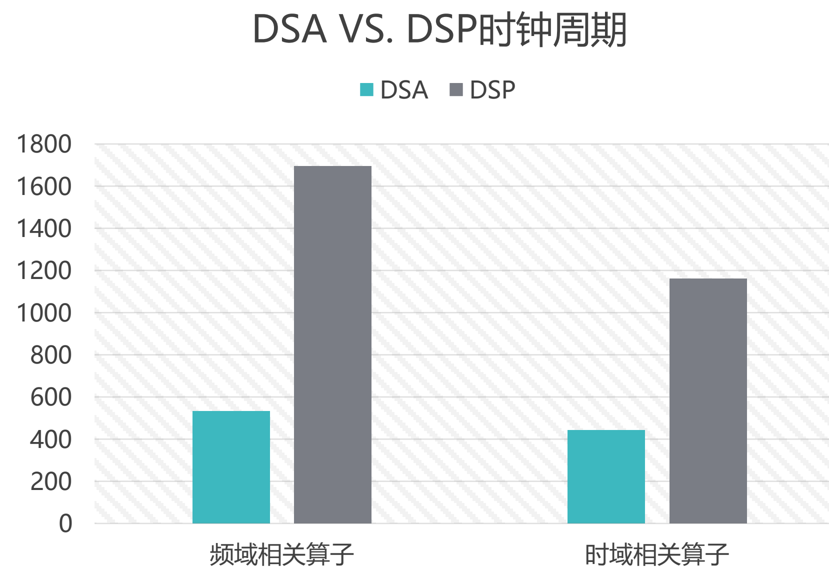


使用RISC-V标量处理器进行任务控制和标量运算

芯来ND900(RISC-dsp-w) VS. 通用DSP

RISC-dsp-w指令集定制DSA在大算力场景下MAC计算密度具有显著优势。在等效硬件MAC算力情况下，RISC-dsp-w指令集相比现有主流DSP在无线信号处理器领域也具有明显优势。同C公司 B型DSP的性能对比如下：

硬件时钟统计对比				
测试用例名称	**RISC-dsp-w DSA 时钟周期**	**C公司B型DSP 时钟周期**	**DSP/DSA 时钟比**	**DSP/DSA 等效硬件算力时钟比**
144点频域相关算子	533	1695	**3.2**	**1.75**
180点时域相关算子	442	1160	**2.6**	**1.45**



RISC-dsp-w DSA: SMIC 28HKD，0.95 mm2，16GOPs (Int16) @ 500MHz，汇编优化
C公司 B型DSP: SMIC 28HKD，0.95 mm2，8GOPs (Int16) @ 500MHz，-Os优化

说明：RISC-dsp-w DSA 为芯来定制矢量处理器，由N300+ 芯来通用vpu为基础定制

芯来ND900(RISC-dsp-w) VS. 通用DSP

- SIMD/VLIW结构的通用DSP
- 支持完全C编程
- 编译器自动优化
- PPA优于CPU

- RVV架构的通用VPU
- 支持完全C编程
- 支持自动向量化
- PPA优于SIMD/VLIW通用DSP

C公司B型DSP

C公司 DSP指令集
SMIC 28HKD
0.95mm
8Gops INT16 @500MHz

C公司B型DSP

C公司 DSP指令集
SMIC 28HKD
0.95mm
8Gops INT16 @500MHz

芯来通用VPU

RISC-dsp-w + RRV1.0
SMIC 28HKD
0.95mm
16Gops INT16
@500MHz

NICE

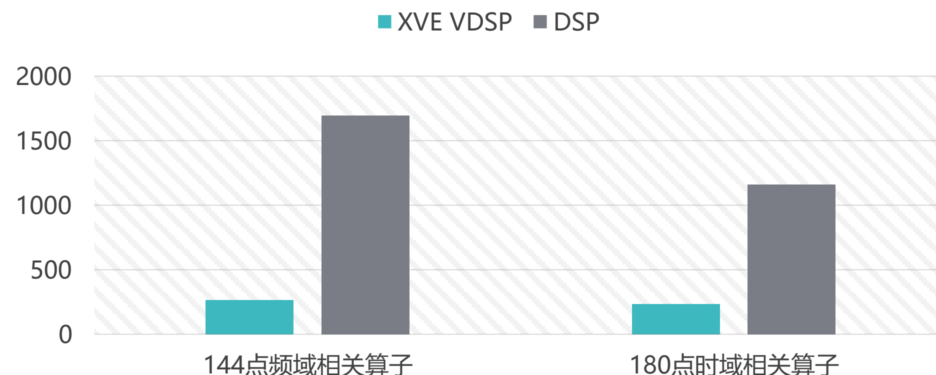
512
FFT
加速
器

C公司B型DSP

C公司 DSP指令集
SMIC 28HKD
0.95mm
8Gops INT16 @500MHz

芯昇XVE架构VDSP(RISC-dsp-w) VS. 通用DSP

硬件时钟统计对比				
测试用例名称	XVE VDSP 时钟周期	C公司B型DSP 时钟周期	DSP/XVE VDSP 时钟比	DSP/VDSP 等效MAC算力 时钟比
144点频域相关算子	266	1695	6.4	3.2
180点时域相关算子	235	1160	4.9	2.45



XVE vDSP
RISC-dsp-w
SMIC 28HKD
0.3mm
16Gops INT16
@500MHz

512 FFT 协处理器

- XVE架构领域专用VDSP
- C和矢量汇编混合编程
- 不支持自动向量化
- 矢量寄存器手动分配
- PPA优于ASIC

★ 512点FFT
★ 支持pipeline处理
★ Tput < 256时钟周期

C公司B型DSP
C公司 DSP指令集
SMIC 28HKD
0.95mm
8Gops INT16 @500MHz

C公司B型DSP
C公司 DSP指令集
SMIC 28HKD
0.95mm
8Gops INT16 @500MHz

C公司B型DSP
C公司 DSP指令集
SMIC 28HKD
0.95mm
8Gops INT16 @500MHz

C公司B型DSP
C公司 DSP指令集
SMIC 28HKD
0.95mm
8Gops INT16 @500MHz

C公司B型DSP
C公司 DSP指令集
SMIC 28HKD
0.95mm
8Gops INT16 @500MHz

- SIMD结构的通用DSP
- 支持完全C编程
- 编译器自动优化
- PPA优于CPU

谢谢！

ligaoshan@cmiot.chinamobile.com
gerald_hust@163.com

参考内容

Conjugate Complex Multiply

RISC-DSP version

```
avl = 32;
vtypeE32 = TA | MA | M1 | E32;
asm volatile("vsetvli %[vl], %[avl], 208;":[vl] "=&r" (vl):[avl] "r" (avl));
for (int32_t i = 0; i < 100; i++)
{
    asm volatile("vle32.v v2, (%[aAddr]);" ::[aAddr]"r"(pa));
    asm volatile("vle32.v v1, (%[bAddr]);" ::[bAddr]"r"(pb));
    asm volatile("vdscmulj.vv v4, v2, v1;"::);
    asm volatile("vse32.v  v4, (%[rZvmAddr]);" ::[rZvmAddr]"r"(pz));
    asm volatile("vsync %[syncRd],%[rs2];":[syncRd]"=&r"(syncRd):[rs2]"r"(rs2));
}
syncRd = syncRd * 2;
asm volatile("fence");
```

RVV1.0 version

```
for (int32_t i = 0; i < 100; i++)
{
    avl = 64;
    vtypeE = TA | MA | M2 | E32;
    asm volatile("vsetvli %[vl], %[avl], 209;":[vl] "=&r" (vl):[avl] "r" (avl));
    asm volatile("vlm.v v0, (%[imageCcmMaskAddr]);"::[imageCcmMaskAddr]"r"(pm));
    vtypeE = TA | MA | M1 | E16;
    asm volatile("vsetvli %[vl], %[avl], 200;":[vl] "=&r" (vl):[avl] "r" (avl));
    asm volatile("vsub.vv v1, v1,v1;"::);
    asm volatile("vle16.v v4, (%[aAddr]);"::[aAddr]"r"(pa));
    asm volatile("vle16.v v5, (%[bAddr]);"::[bAddr]"r"(pb));
    asm volatile("vmerge.vvm v2,v1,v4,v0;"::);
    asm volatile("vmerge.vvm v4,v4,v1,v0;"::);
    asm volatile("vslidedown.vi v2,v2,1;"::);
    asm volatile("vmerge.vvm    v3,v1,v5,v0;"::);
    asm volatile("vmerge.vvm    v1,v5,v1,v0;"::);
    asm volatile("vslidedown.vi v3,v3,1;"::);
    asm volatile("vmul.vv   v5,v4,v1;"::);
    asm volatile("vmul.vv   v6,v2,v3;"::);
    asm volatile("vmul.vv   v7,v4,v3;"::);
    asm volatile("vmul.vv   v8,v2,v1;"::);
    asm volatile("vadd.vv   v1,v5,v6;"::);
    asm volatile("vsub.vv   v2,v8,v7;"::);
    asm volatile("vslideup.vx   v3,v2,%[num1];"::[num1]"r"(num1));
    asm volatile("vmerge.vvm    v1,v1,v3,v0;"::);
    asm volatile("vse16.v   v1,(%[rRvvAddr]);"::[rRvvAddr]"r"(pz));
    asm volatile("vsync %[syncRd],%[rs2];":[syncRd]"=r"(syncRd):[rs2]"r"(rs2));
}
syncRd = syncRd * 2;
asm volatile("fence");
```

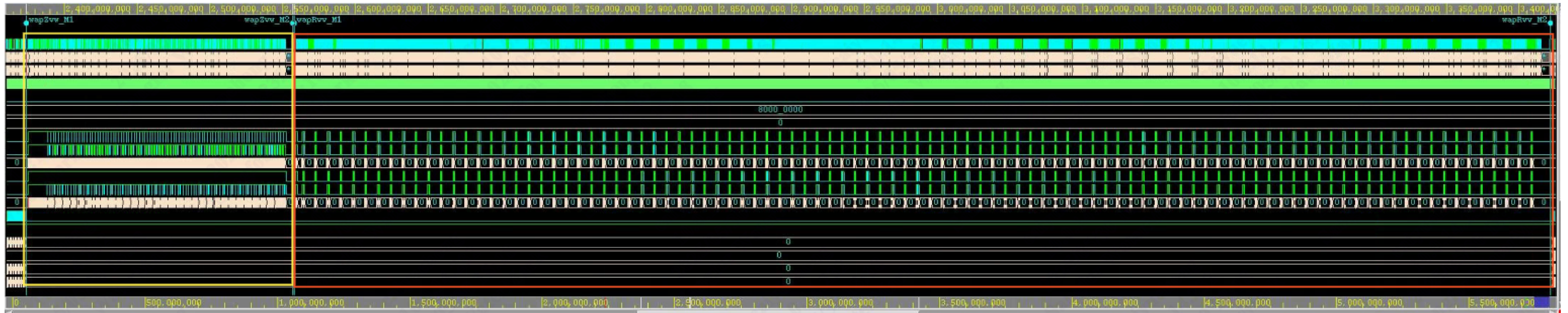
Wideband Average Power

RISC-DSP ersion

```
uint32_t vcsrA0M0R0Sa = ACCSFT0 | MULSFT0 | VXRM_RNU | VXSAT1;
asm volatile("csrw vcsr,%[vcsrA0M0R0Sa];":: [vcsrA0M0R0Sa] "r" (vcsrA0M0R0Sa));

avl = 32;
vtypeE = TA | MA | M1 | E32;
asm volatile("vsetvli %[vl], %[avl],208;":[vl] "=&r" (vl):[avl] "r" (avl));
for (int32_t i = 0; i < 100; i++)
{
    asm volatile("vle32.v v0, (%[gainShiftAdrrZvw]);" ::[gainShiftAdrrZvw]"r"(pg));
    asm volatile("vle32.v v1, (%[a0Addr]);" ::[a0Addr]"r"(pa));
    asm volatile("vdsmacini.v v0;" ::);
    asm volatile("vle32.v v2, (%[a1Addr]);" ::[a1Addr]"r"(pb));
    asm volatile("vdscmacj.vv v1,v1;" ::);
    asm volatile("vdscmacjor.vv v3, v2, v2;" ::);
    asm volatile("vdscredsum.v v4,v3;" ::);
    asm volatile("vmv.x.s  %[result], v4;":[result]"=&r"(pz):);
}
//asm volatile("vsync %[syncRd],%[rs2];":[syncRd]"=&r"(syncRd):[rs2]"r"(rs2));
asm volatile("fence");
```

RVV1.0 version

```
for (int32_t i = 0; i < 100; i++)
{
    asm volatile("vsetvli %[vl], %[avl],208;":[vl] "=&r" (vl):[avl] "r" (32));
    asm volatile("vsub.vv v0, v0,v0;"::);//VALU1
    asm volatile("vle32.v v2, (%[a0Addr]);"::[a0Addr]"r"(VM_SRC1_ADDR));//LSD
    asm volatile("vsrl.vx v3, v2, %[num16];"::[num16]"r"(16));//VALU2 IM(L)
    asm volatile("vle32.v v1, (%[gainShiftAddr]);" ::[gainShiftAddr]"r"(VM_GAIN_RVV_ADDR));//
    asm volatile("vsll.vx v4, v2, %[num16];"::[num16]"r"(16));//VALU2 RE(H)
    asm volatile("vle32.v v9, (%[a1Addr]);"::[a1Addr]"r"(VM_SRC1_ADDR + 32*4));//LSD
    asm volatile("vsrl.vx v5, v4, %[num16];"::[num16]"r"(16));//VALU2 RE(L)
    asm volatile("vsrl.vx v13, v9, %[num16];"::[num16]"r"(16));//VALU2 IM1(L)
    asm volatile("vmul.vv v6, v5, v5;"::);//VALU1 RE(L)*RE(L)
    asm volatile("vsll.vx v14, v9, %[num16];"::[num16]"r"(16));//VALU2 RE1(H)
    asm volatile("vsra.vx v6, v6, %[mulShfit];"::[mulShfit]"r"(0));//VALU2
    asm volatile("vmul.vv v7, v3, v3;"::);//VALU1 IM(L)*IM(L)
    asm volatile("vsrl.vx v15, v14, %[num16];"::[num16]"r"(16));//VALU2 RE1(L)
    asm volatile("vsra.vx v7, v7, %[mulShfit];"::[mulShfit]"r"(0));//VALU2
    asm volatile("vmul.vv v17, v13, v13;"::);//VALU1 IM1(L)*IM1(L)
    asm volatile("vadd.vv v8, v7, v6;"::[num16]"r"(16));//VALU1 RE(L)*RE(L)+IM(L)*IM(L)
    asm volatile("vmul.vv v16, v15, v15;"::);//VALU1 RE1(L)*RE1(L)
    asm volatile("vsra.vv v11, v8, v1;"::);//VALU2 vsra(RE(L)*RE(L)+IM(L)*IM(L))
    asm volatile("vadd.vv v10, v17, v16;"::);//VALU1 RE1(L)*RE1(L)+IM1(L)*IM1(L)
    asm volatile("vsra.vv v12, v10, v1;"::);//VALU2 vsra(RE1(L)*RE1(L)+IM1(L)*IM1(L))
    asm volatile("vadd.vv v13, v11, v12;"::);//VALU1
    asm volatile("vsra.vx v14, v13, %[accShfit];"::[accShfit]"r"(0));//VALU2
    asm volatile("vredsum.vs v15, v14, v0;"::);//VALU2
    asm volatile("vmv.x.s  %[result], v15;":[result]"=&r"(*pz):);//VALU2
    //asm volatile("vsync %[syncRd],%[rs2];":[syncRd]"=&r"(syncRd):[rs2]"r"(0xFF));
}
asm volatile("fence");
```



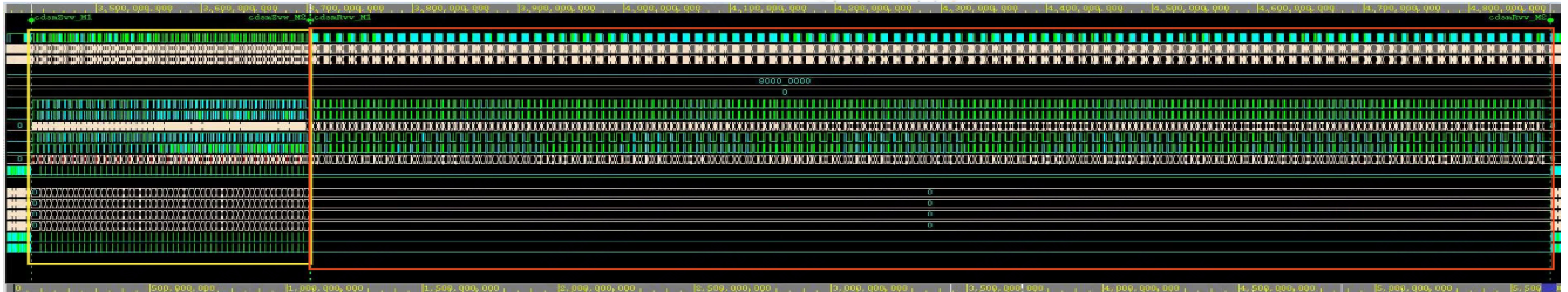
Complex Dynamic Scaling MAC

RISC-DSP version

```
uint32_t vcsrA0M0R0Sa = ACCSFT0 | MULSFT0 | VXRM_RNU | VXSAT1;
asm volatile("csrw vcsr,%[vcsrA0M0R0Sa];":: [vcsrA0M0R0Sa] "r" (vcsrA0M0R0Sa));
asm volatile("vdsmacini.s %[gainShift];" ::[gainShift]"r"(gainShift));
for (int32_t i = 0; i < 100; i+=2)
{
    //vtypeE = TA | MA | M2 | E32;
    asm volatile("vsetvli %[vl], %[avl],209;":[vl] "=&r" (vl):[avl] "r" (64));
    asm volatile("vle32.v v0, (%[aCdsmAddr]);" ::[aCdsmAddr]"r"(pa));
    asm volatile("vle32.v v2, (%[bCdsmAddr]);" ::[bCdsmAddr]"r"(pb));
    asm volatile("vdscmacjo.vv v10,v0, v2;"::);
    //vtypeE = TA | MA | M1 | E32;
    asm volatile("vsetvli %[vl], %[avl],208;":[vl] "=&r" (vl):[avl] "r" (32));
    asm volatile("vdscredsum.v v12,v10;"::);

    asm volatile("vsetvli %[vl], %[avl],209;":[vl] "=&r" (vl):[avl] "r" (64));
    asm volatile("vle32.v v4, (%[aCdsmAddr]);" ::[aCdsmAddr]"r"(pa));
    asm volatile("vle32.v v6, (%[bCdsmAddr]);" ::[bCdsmAddr]"r"(pb));

    asm volatile("vsetvli %[vl], %[avl],208;":[vl] "=&r" (vl):[avl] "r" (1));
    asm volatile("vmv.x.s  %[result], v12;" :[result]"=&r"(*pz):);

    asm volatile("vsetvli %[vl], %[avl],209;":[vl] "=&r" (vl):[avl] "r" (64));
    asm volatile("vdscmacjo.vv v12, v4, v6;"::);

    asm volatile("vsetvli %[vl], %[avl],208;":[vl] "=&r" (vl):[avl] "r" (32));
    asm volatile("vdscredsum.v v14,v12;"::);

    asm volatile("vsetvli %[vl], %[avl],208;":[vl] "=&r" (vl):[avl] "r" (1));
    asm volatile("vmv.x.s  %[result], v14;" :[result]"=&r"(*pz):);
}
asm volatile("fence");
```

RVV1.0 version

```
for (int32_t i = 0; i < 100; i++)
{
    asm volatile("vsetvli %[vl], %[avl],208;":[vl] "=&r" (vl):[avl] "r" (1));
    asm volatile("vmv.v.i v26, 0;");//VALU2
    asm volatile("vsetvli %[vl], %[avl],209;":[vl] "=&r" (vl):[avl] "r" (64));
    asm volatile("vle32.v v2, (%[aCdsmAddr]);"::[aCdsmAddr]"r"(VM_SRC1_ADDR));//LSD
    asm volatile("vsrl.vx v6, v2, %[num16];"::[num16]"r"(16));//VALU2 IM0(L)
    asm volatile("vle32.v v4, (%[bCdsmAddr]);"::[bCdsmAddr]"r"(VM_SRC2_ADDR));//LSD
    asm volatile("vsrl.vx v12, v4, %[num16];"::[num16]"r"(16));//VALU2 IM1(L)
    asm volatile("vsll.vx v8, v2, %[num16];"::[num16]"r"(16));//VALU2 RE0(H)
    asm volatile("vsll.vx v30, v4, %[num16];"::[num16]"r"(16));//VALU2 RE1(H)
    asm volatile("vmul.vv v26, v6, v12;"::);//VALU1 IM0(L)*IM1(L)
    asm volatile("vsrl.vx v10, v8, %[num16];"::[num16]"r"(16));//VALU2 RE0(L)
    asm volatile("vsrl.vx v14, v30, %[num16];"::[num16]"r"(16));//VALU2 RE1(L)
    asm volatile("vmul.vv v16, v10, v14;"::);//VALU1  RE0(L)*RE1(L)
    asm volatile("vadd.vv v16, v16, v26;"::);//VALU1   RE0(L)*RE1(L)+IM0(L)*IM1(L)
    asm volatile("vmul.vv v28, v6, v14;"::);//VALU1  IM0(L)*RE1(L)
    asm volatile("vsetvli %[vl], %[avl],209;":[vl] "=&r" (vl):[avl] "r" (32));
    asm volatile("vredsum.vs v22, v16, v26;"::);//VALU2
    asm volatile("vsetvli %[vl], %[avl],209;":[vl] "=&r" (vl):[avl] "r" (64));
    asm volatile("vmul.vv v18, v10, v12;"::);//VALU1  IM1(L)*RE0(L)
    asm volatile("vsub.vv v18, v28, v18;"::);//VALU1  IM0(L)*RE1(L)-IM1(L)*RE0(L)
    asm volatile("vsetvli %[vl], %[avl],209;":[vl] "=&r" (vl):[avl] "r" (32));
    asm volatile("vredsum.vs v20, v18, v26;"::);//VALU2
    asm volatile("vsetvli %[vl], %[avl],208;":[vl] "=&r" (vl):[avl] "r" (1));
    asm volatile("vsll.vx v0, v20, %[num16];"::[num16]"r"(16));//VALU2
    asm volatile("vxor.vv v2, v0, v22;"::);//VALU1
    asm volatile("vmv.x.s %[result], v2;"::[result]"r"(*pz));//VALU2
    //asm volatile("vsync %[syncRd],%[rs2];":[syncRd]"=&r"(syncRd):[rs2]"r"(0xFF));
}
asm volatile("fence");
```



OSP（offset scalar product）

OSP (Offset Scalar Product) is defined as the differential autocorrelation in the frequency - domain direction of a total of 180 DMRS, SSS, and the LS estimates obtained from the differences, that is:

$$OSP_{com} = \sum_{l=1}^{3} \sum_{\substack{k=[0:4:239] \\ +nid\%4}} \hat{H}_{LS}(k+1,l)\hat{H}_{LS}^{*}(k,l)$$

$$OSP_{real} = \max(\mathrm{real}(OSP_{com}), 0)$$

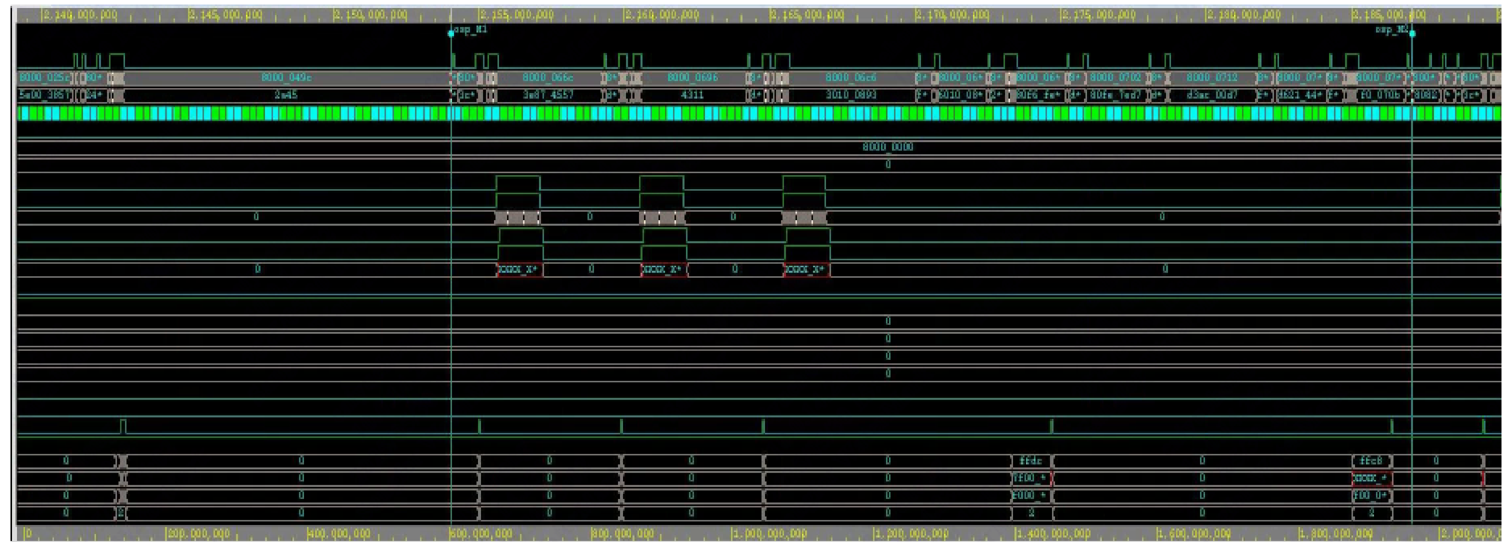
For a total of 180 points of DMRS, SSS, and the interpolated LS estimates, perform the misaligned conjugate multiplication in the frequency - domain. Calibration: SC16.15 * SC16.15 = SC33.30. Accumulate the results of 59*3 = 177 multiplications to obtain ospComm. Calibration: SC40.29. Shift 8 bits to get SC32.22. Take the real part to obtain ospReal:

```
void Test_Osp_Calc()
{
    T_L1R_CSM_ComplexS32    ospRst;

    MCPS_OPEN_FILE
    MCPS_START_CLOCK

    L1R_algo_csm_SsbMeas_CalcOsp((T_L1R_CSM_ComplexS16*)&gOspTstInData

    MCPS_STOP_AND_LOG("Test_Osp_Calc_Cycle ")
    MCPS_CLOSE_FILE
```

frcc	00000010	16

frcc	000006af	1711

某公司 clock = 1711 - 16 = 1695, 16 is start counter, 1711 is end counter, use 某公司 lib function clock( ) to get processor clock cycles.



RISC-DSP VPU cycles for OSP = 533

NSP（normal scalar product）

NSP (Normal Scalar Product) is defined as the differential autocorrelation in the time - domain of a total of 180 DMRS, SSS, and the LS estimates obtained from the differences, that is:

$$NSP = \sum_{l=1,2} \sum_{\substack{k=[0:4:239] \\ +nid\%4}} \hat{H}_{LS}(k,l)\hat{H}^{*}_{LS}(k,l+1) = \sum_{l=1,2} \sum_{k} H(l,k)H^{*}(l+1,k)$$

For a total of 180 points of DMRS, SSS, and the interpolated LS estimates, perform the misaligned conjugate multiplication in the time - domain. Calibration: SC16.15 * SC16.15 = SC33.30. Accumulate 60×2 = 120 products to obtain nsp. Calibrate to SC40.30 and shift 8 bits to get SC32.22:

```
void Test_Nsp_Calc()
{
    T_L1R_CSM_ComplexS32   nspRst;

    MCPS_OPEN_FILE
    MCPS_START_CLOCK

    L1R_algo_csm_SsbMeas_CalcNsp((T_L1R_CSM_ComplexS16*) gNspTestInDataRx0[0], &nspRst);

    MCPS_STOP_AND_LOG("Test_Nsp_Calc_Cycle ")
    MCPS_CLOSE_FILE
    return;
}
```

frcc	00000013	19

frcc	0000049b	1179

某公司 clock = 1179 - 19 = 1160, 19 is start counter, 1179 is end counter, use 某公司 lib function clock( ) to get processor clock cycles.

RISC-DSP VPU cycles for NSP = 442