1. 降采样

C代码：

void resampleData(const int16\_t \*sourceData, int16\_t \*destinationData,

uint32\_t sampleRate, uint32\_t newSampleRate,

uint32\_t srcSize)

{

uint32\_t dstSize = (uint32\_t) srcSize \* newSampleRate / sampleRate;

for (uint32\_t idx = 0; idx < dstSize; idx++)

{

float index = ((float) idx \* sampleRate) / (newSampleRate);

uint32\_t p1 = (uint32\_t) index;

float coef = index - p1;

uint32\_t p2 = (p1 == srcSize) ? srcSize : p1 + 1;

destinationData[idx] = (int16\_t) ((1.0f - coef) \* sourceData[p1] + coef \*

sourceData[p2]);

}

}

汇编：默认情况下，寄存器a0，a1，a2 ... a7依次保存函数参数。故a0保存了源数据首地址，a1保存了目的内存空间的首地址，a2和a3分别保存了旧、新采样率，a4保存了源数据的个数。

.global \_start

\_start:

la a0, s1

la a1, d1

li a2, 10

li a3, 5

li a4, 10

ebreak

0: mul a5, a3, a4

4: divu a5, a5, a2 # 计算dstSize，因为数据必定为正，所以当无符号数处理

8: fcvt.s.wu ft0, a3 # 把新采样率转化成单精度浮点数

c: addi a6, x0, 0 # idx = 0

Loop:

10: bltu a6, a5, 18f # idx < dstSize

ExitLoop:

14: jalr x0, 0(x1) # 从函数返回至中断处

ContinueLoop:

18: mul t0, a6, a2

1c: fcvt.s.wu ft1, t0 # 转化成单精度浮点数

20: fdiv.s ft2, ft1, ft0 # 计算index

24: fcvt.wu.s t1, ft2, rdn # 对index取整得到p1

28: fcvt.s.wu ft3, t1 # 将p1转化成浮点数

2c: fsub.s ft4, ft2, ft3 # 计算coef

Judge:

30: lh t3, 0(a0) # 取sourceData[p1]

34: addi t6, t3, 0 # 计算destinationData[idx]

38: bgeu t1, a4, 60f # 如果p1是最后一个数据，直接写回数据

Continue:

3c: fcvt.s.w ft5, t3 # sourceData[p1]变浮点数

40: lh t4, 2(a0) # 取sourceData[p1+1]

44: fcvt.s.w ft6, t4 # sourceData[p1+1]变浮点数

48: addi t5, x0, 1

4c: fcvt.s.w ft7, t5 # 浮点数1.0

50: fsub.s ft7, ft7, ft4 # 1.0-coef

54: fmul.s ft7, ft5, ft7

58: fmadd.s ft7, ft4, ft6, ft7 # 计算浮点数destinationData[idx]

5c: fcvt.w.s t6, ft7 # 转化成定点数

60: sh t6, 0(a1) # 写回内存

64: addi a1, a1, 2 # 目的内存地址偏移

68: addi a7, x0, 2

6c: divu t0, a2, a3

70: mul a7, a7, t0

74: add a0, a0, a7 # 源数据地址偏移

78: addi a6, a6, 1 # idx++

ebreak

7c: jal x0, 10b # 跳转到循环变量判断

.section .data

s1: .half 0x0001

s2: .half 0x0002

s3: .half 0x0003

s4: .half 0x0004

s5: .half 0x0005

s6: .half 0x0006

s7: .half 0x0007

s8: .half 0x0008

s9: .half 0x0009

s10: .half 0x000a

d1: .half 0x0000

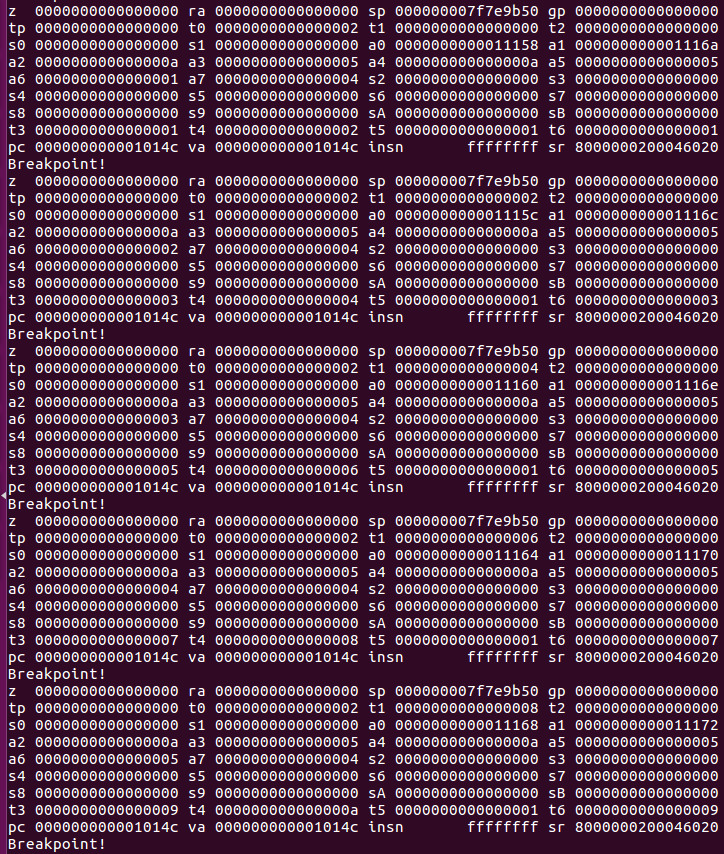
d2: .half 0x0000

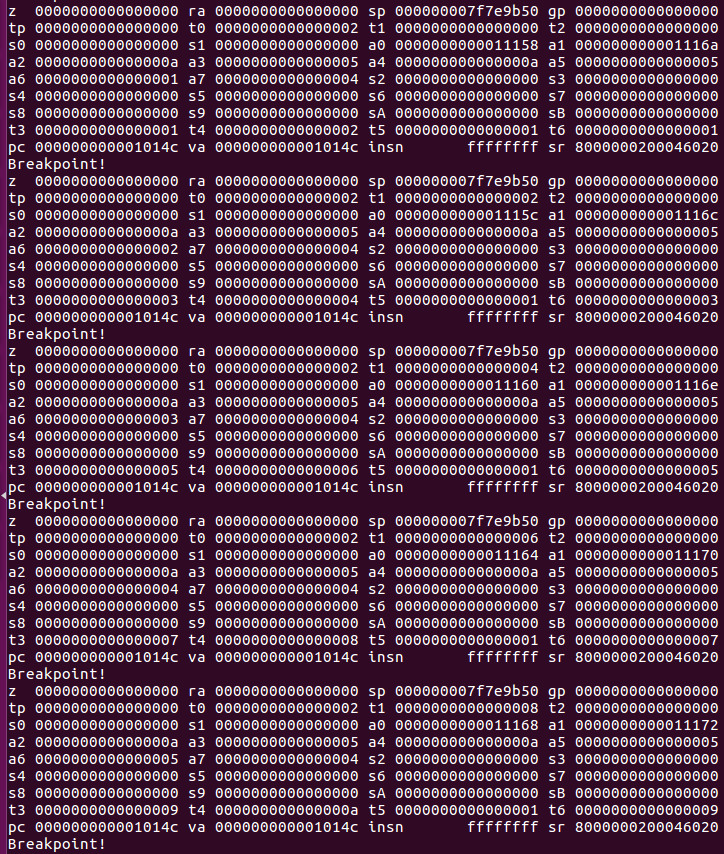
d3: .half 0x0000

d4: .half 0x0000

d5: .half 0x0000

为了模拟实际情况，假设原采样率为10，源数据正好10个，新采样率为5。预期的采样结果应该是t6出现1，3，5，7，9，仿真结果与预期一致。





1. FIR

C代码：

void FIR(const int16\_t \*sourceData, const int16\_t \*weightData,

int16\_t \*destinationData, uint32\_t dataSize, uint32\_t N)

{

uint32\_t i, j;

for (i = 0; i < dataSize; i++)

{

for (j = 0; j < N; j++)

{

if(i >= j) destinationData[i] += sourceData[i-j] \* weightData[j];

}

}

}

汇编：a0=输入数据首地址，a1=系数数组首地址， a2=输出数据首地址， a3=输入数组大小， a4=阶数N

.global \_start

\_start:

la a0, s1

la a1, w1

la a2, d1

li a3, 10

li a4, 3

0: addi a7, a2, 0 # 复制输出数据的首地址

4: andi s2, a4, 1 # 由于SIMD指令一次取两个数，所以要判断N的奇偶性

8: beq s2, x0, 10f

c: addi a4, a4, -1 # 若N为奇数，-1变偶数，最后一次单独加

10: addi a5, x0, 0 # i = 0

OutLoop:

14: bltu a5, a3, 1c # i < dataSize

ExitOutLoop:

18: jalr x0, 0(x1) # 从函数返回至中断处

InnerLoop:

1c: addi s0, x0, 2

20: mul s0, s0, a5

24: add s0, s0, a0 # 根据i计算本次起始输入数据的位置

28: addi s1, a1, 0 # 权值地址复位

2c: addi a6, x0, 0 # j = 0

30: addi t5, x0, 0 # 上次的计算结果清零

34: bltu a6, a4, 60f # j < N，进入内循环，否则内循环结束

38: addi a5, a5, 1 # i++

3c: ble a5, a4, 54b # 若输入数据小于N，无需额外加一次，直接跳转

40: beq s2, x0, 54b # 若N为偶数，无需额外加一次，直接跳转

44: lh t0, 0(s0) # 取一个16位的输入数据

48: lh t1, 0(s1) # 取一个16位的系数

4c: mul t2, t0, t1 # 改用正常乘法

50: qadd16 t5, t2, t5

54: sh t5, 0(a7) # 写内存

58: addi a7, a7, 2 # 完成一次循环，输入数据前进一个

5c: jal x0, 14b # 跳转到外循环

60: bltu a5, a6, 38b # i < j，则内循环结束，跳转到外循环

64: bne a5, a6, 7c # i = j则只剩一个输入数据，取数模式改变，不等则数据≥2正常取数

68: lh t0, 0(s0) # 取一个16位的输入数据

6c: lh t1, 0(s1) # 取一个16位的系数

70: mul t2, t0, t1 # 改用正常乘法

74: qadd t5, t2, t5 # 计算最终结果

78: jal x0, 38b # 内循环结束，跳转到i++

7c: lw t0, -2(s0) # 取两个16位的输入数据

80: lw t1, 0(s1) # 取两个16位的系数

84: smulbt t2, t0, t1

88: smultb t3, t0, t1

8c: qadd16 t4, t2, t3 # 中间结果完成两次计算

90: qadd16 t5, t4, t5 # 累加

94: addi a6, a6, 2 # j += 2

98: addi s0, s0, -4 # 输入数据地址偏移

9c: addi s1, s1, 4 # 系数地址偏移

a0: jal x0, 34b # 跳转到内循环

.section .data

s1: .half 0x0001

s2: .half 0x0002

s3: .half 0x0003

s4: .half 0x0004

s5: .half 0x0005

s6: .half 0x0006

s7: .half 0x0007

s8: .half 0x0008

s9: .half 0x0009

s10: .half 0x000a

w1: .half 0x0001

w2: .half 0x0002

w3: .half 0x0003

w4: .half 0x0004

w5: .half 0x0005

w6: .half 0x0006

w7: .half 0x0007

w8: .half 0x0008

w9: .half 0x0009

w10: .half 0x000a

d1: .half 0x0000

d2: .half 0x0000

d3: .half 0x0000

d4: .half 0x0000

d5: .half 0x0000

d6: .half 0x0000

d7: .half 0x0000

d8: .half 0x0000

d9: .half 0x0000

d10: .half 0x0000

最终结果保存在寄存器t5里面。仿真结果和预期一致。

以最后一次为例，t5 = 1 \* 10 + 2 \* 9 + 3 \* 8 = 52 =0x34

