数值的处理

整数

二进制整数的原码

- 最高位表示符号位，1表示负，0表示正
- 其他位存放该数的二进制的绝对值

	正数
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

	负数
-0	1000
-1	1001
-2	1010
-3	1011
-4	1100
-5	1101
-6	1110
-7	1111

0001+0010=0011 （1+2=3）正确

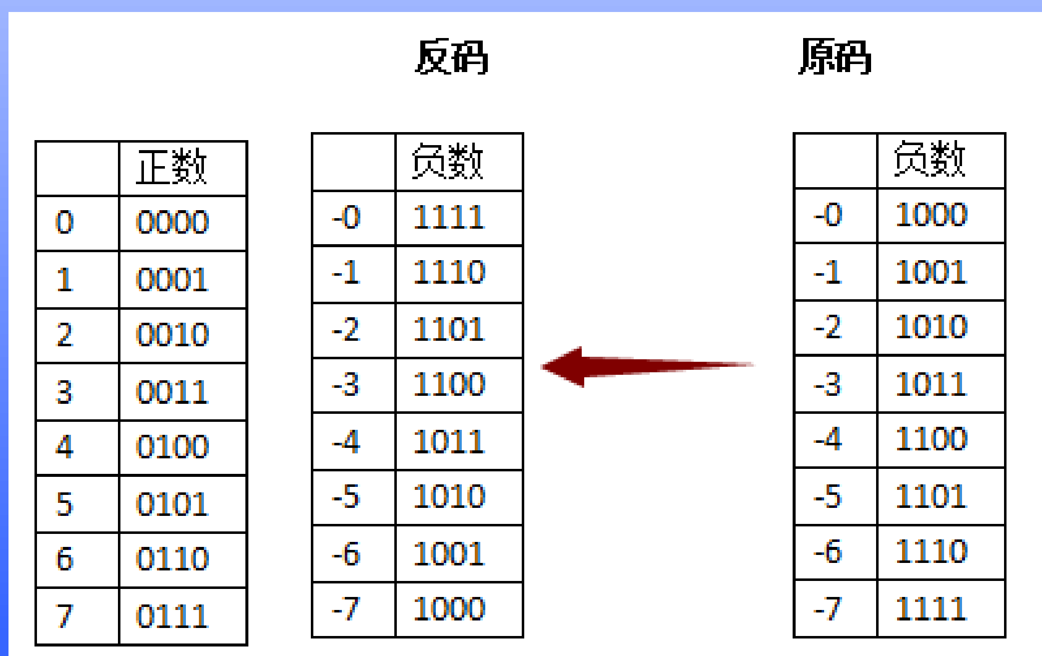
0000+1000=1000 （(+0)+(-0)=-0）

0001+1001=1010 （1+（-1）=-2）出错

整数

二进制整数的反码

- 正数的反码=原码
- 负数的反码=原码除符号位外，按位取反



	正数
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

反码

	负数
-0	1111
-1	1110
-2	1101
-3	1100
-4	1011
-5	1010
-6	1001
-7	1000

原码

	负数
-0	1000
-1	1001
-2	1010
-3	1011
-4	1100
-5	1101
-6	1110
-7	1111

0001+1110=1111 (1+(-1)=(-0)) 正确

1110+1101=1011 (-1)+(-2)=(-4) 两个负数相加，出错

整数

二进制整数的补码

- **模** 是一个计量器的容量大小，比如钟表的模M=12
- **同余** 是指两个整数A和B除以同一个正整数M，所得余数相同，比如1点和13点，2点和14点就是同余的

如果说现在时针现在停在10点钟，那么什么时候时针会停在8点钟呢？

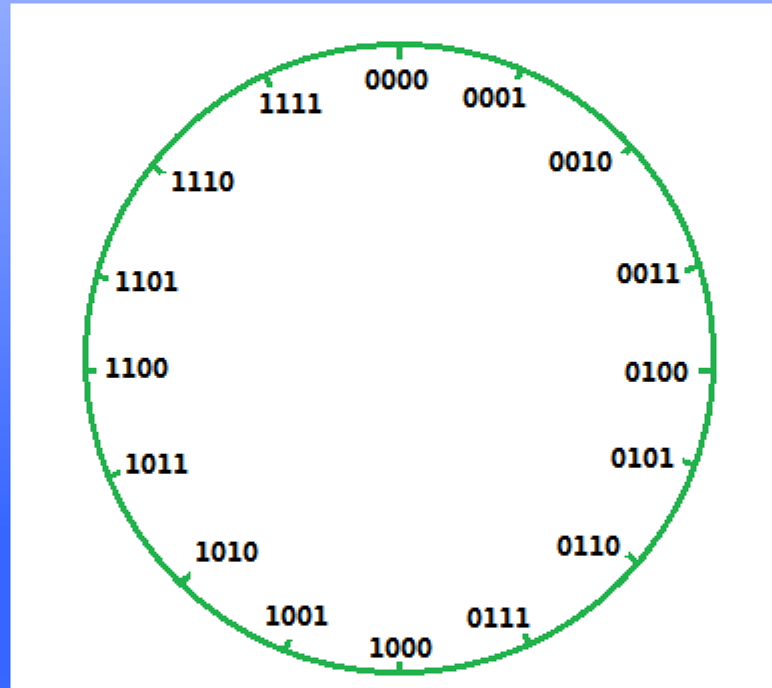
倒拨2小时 或 正拨10小时都是8点钟（20点钟）。

时钟运算中，减去一个数，其实就相当于加上另外一个数（同余数）

整数

二进制整数的补码

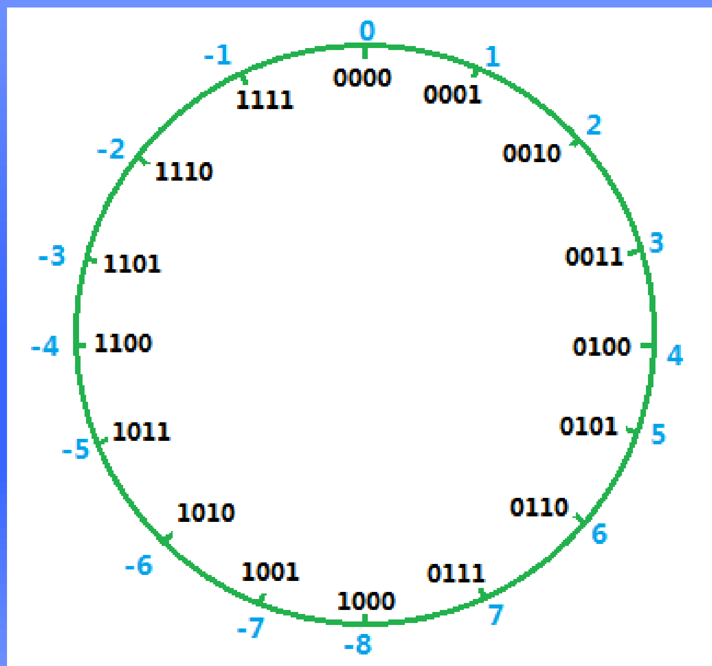
- 由于溢出归零的特点，计算机的二进制更像时钟旋转，如下图：



整数

二进制整数的补码

- 如在0000逆时针旋转1个数或者顺时针旋转16-1得到1111，那么1111代表-1，相应的顺时针移动一个数为+1，即用0001表示+1
- 0000逆时针旋转8个数得到1000，所以1000为-8，相应的顺时针旋转8个数也得到了1000，1000既能表示-8又能表示+8，为了不产生冲突，人为规定1000为-8。



整数

二进制整数的补码

	正数
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

补码

	负数
0	0000
-1	1111
-2	1110
-3	1101
-4	1100
-5	1011
-6	1010
-7	1001
-8	1000

反码

	负数
-0	1111
-1	1110
-2	1101
-3	1100
-4	1011
-5	1010
-6	1001
-7	1000

原码

	负数
-0	1000
-1	1001
-2	1010
-3	1011
-4	1100
-5	1101
-6	1110
-7	1111

0110(6) - 0010(2) = 0110(6) + 1110(14) = 10100(16+4=20)

什么是符号扩展？

- 符号扩展，即短数据类型的符号位填充到长数据类型的高字节位（即比短数据类型多出的那一部分），保证扩展后的数值大小不变
- 只适用于有符号数
- 符号不变，负数扩展之后还是负数
- 由ST0寄存器的SXM位控制。SXM=1时，符号扩展自动进行。

4位的例子: 把存储器中的值加载到ACC中

存储器 1101 $= -2^3 + 2^2 + 2^0 = -3$

加载并符号扩展

ACC 1111 1101 $= -2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^0$

$= -128 + 64 + 32 + 16 + 8 + 4 + 1$

$= -3$

整数的乘法(有符号的)

```
        0100
      x 1101
    00000100
    0000000
    000100
    11100
    11110100
```

```
    4
x  -3
-----
  -12
```

累加器 11110100

数据内存 ?

《sprac71a》——TI

整数的乘法(有符号的)

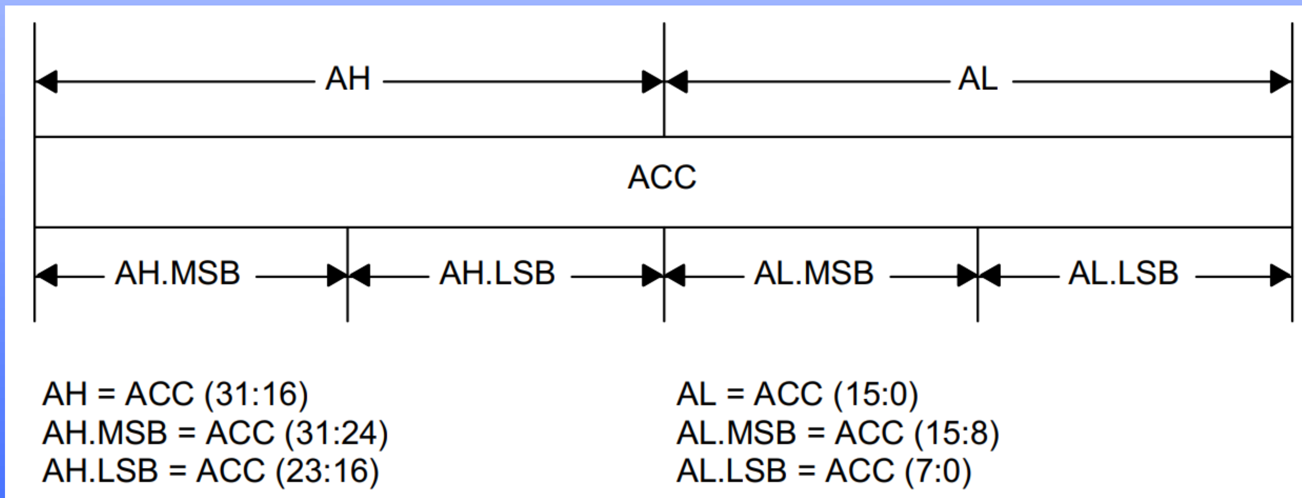
Table 1. Data Sizes for Standard Types

Type	Generic Name	Size	Alignment
signed char	schar	16	16
unsigned char	uchar	16	16
char	plain char	16	16
bool (C99)	uchar	16	16
_Bool (C99)	uchar	16	16
bool (C++)	uchar	16	16
short, signed short	int16	16	16
unsigned short	uint16	16	16
int, signed int	int16	16	16
unsigned int	uint16	16	16
long, signed long	int32	32	32
unsigned long	uint32	32	32
long long, signed long long	int64	64	32
unsigned long long	uint64	64	32
enum	--	varies (see Section 2.9)	32
float	float32	32	32
double	float64	64	32
long double	float64	64	32
pointer	--	32	16

《sprac71a》——TI

整数的乘法(有符号的)

DSP28335的累加器



小数的基本概念

$1.101b = (1*-2^0)+(1*2^{-1})+(0*2^{-2})+(1*2^{-3})$

$= -1 + 1/2 + 1/8$

$= -3/8$

纯小数的特点：纯小数×纯小数，乘积还是纯小数

小数的乘法

```
     0.100
  x  1.101
  --------
  00000100
  0000000
  000100
  11100
  --------
  11110100
```

$\frac{1}{2} \times -\frac{3}{8} = -\frac{3}{16}$

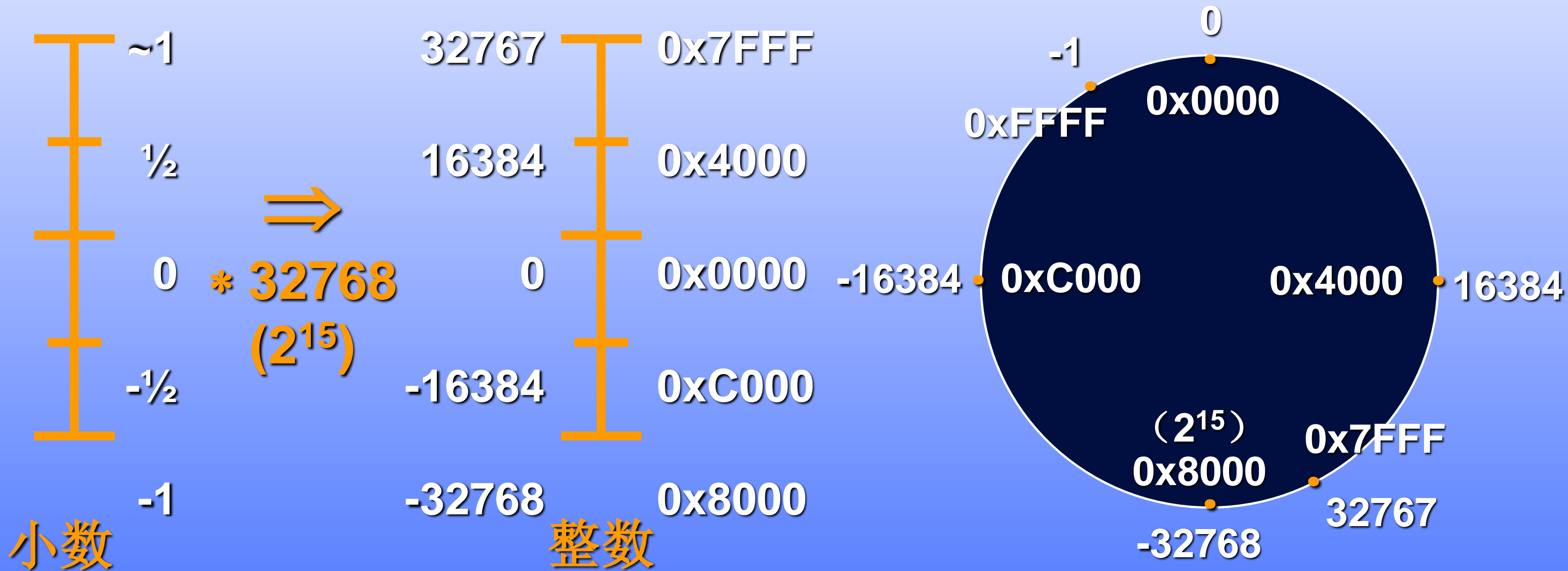
```
  1/2
x -3/8
------
 -3/16
```

累加器 1.1110100

数据内存 1.110

-1/4

传统的16位Q15定点数



- C代码示例: y = 0.707 * x

```
void main(void)
{
   int16 coef = 32768*707/1000;   // 0.707 in Q15
   int16 x, y;
   y = (int16)( (int32)coef * (int32)x ) >> 15);
}
```

整数 vs. 小数

	范围	精度
整数	由位数决定	1
小数	~+1 to -1	由位数决定

- 小数的范围有限
  - 可通过定标（移动小数点）改变范围

有什么变通方法？

IEEE-754单精度浮点

1位符号位　　8位指数位　　23位尾数位，即小数位

	Case 1:	if e = 255 and f ≠ 0,	then v = NaN
	Case 2:	if e = 255 and f = 0,	then v = $[(-1)^s]*\mathrm{Inf}$
标准化的值 →	Case 3:	if 0 < e < 255,	then v = $[(-1)^s]*[2^{(e-127)}]*(1.f)$
	Case 4:	if e = 0 and f ≠ 0,	then v = $[(-1)^s]*[2^{(-126)}]*(0.f)$
	Case 5:	if e = 0 and f = 0,	then v = $[(-1)^s]*0$

例: 0x41200000 = 0 | 100 0001 0 | 010 0000 0000 ... 0000 | b

s | e = 130 | $f = 2^{-2} = 0.25$

⇒ Case 3　　$v = (-1)^0 * 2^{(130-127)} * 1.25 = 10.0$

优点 ⇒ 通过改变指数位，可表示的动态范围较大

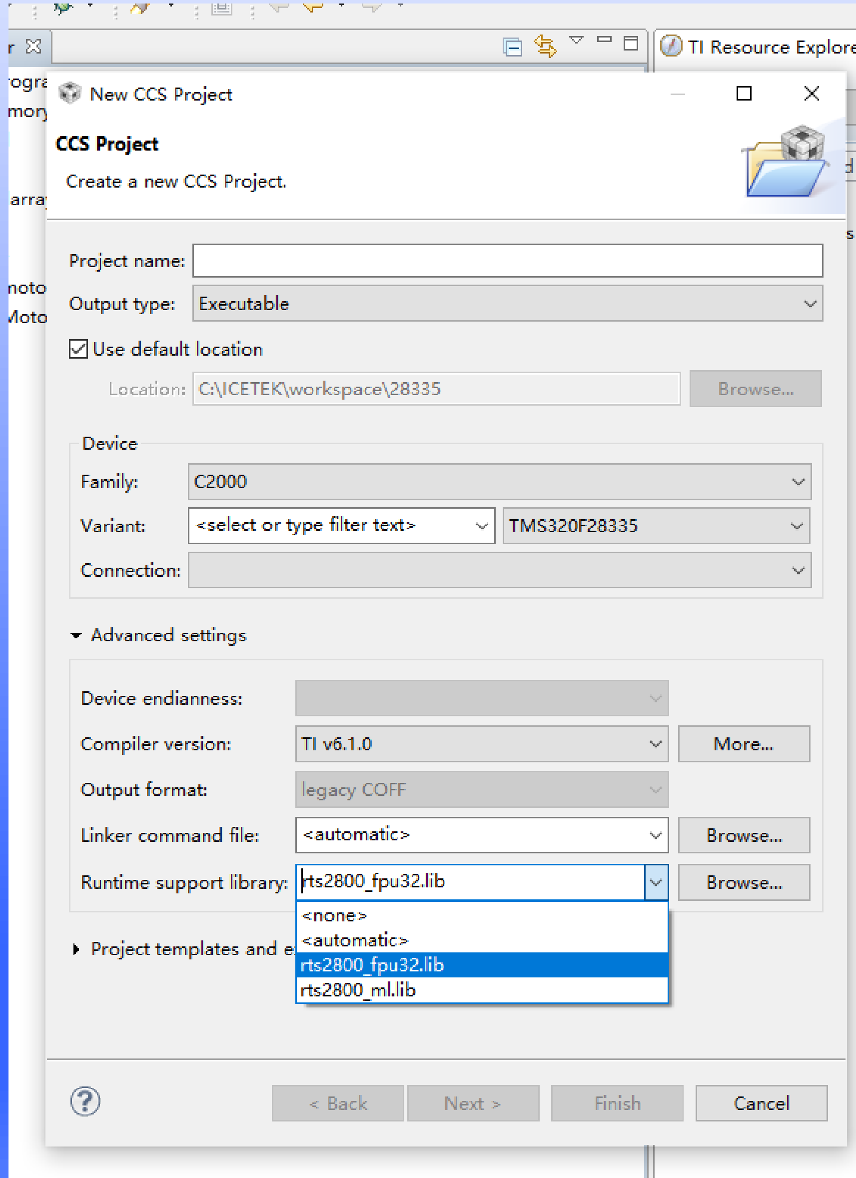
不足 ⇒ 精度依赖于指数位

精度与数值范围示意

浮点

- 非均匀分布
  - 绝对值越小，分辨率就越高
  - 绝对值越大，分辨率越低

使用FPU进行编程



1) 器件需含有硬件FPU
2) 把浮点RTS库的支持加入CCS工程中
   - 标准的RTS库
     - rts2800_fpu32.lib
     - 编译器自带
3) 选择FPU支持的选项‘fpu32’

浮点运算的正面与反面

- 优点
  - 代码书写方便
  - 不需要各种定标转换
- 不足
  - 需要更高级的器件
  - IEEE754单精度浮点的精度有限，因为尾数只有23位

浮点运算的正面与反面

- 不足
  - IEEE754单精度浮点的精度有限，因为尾数只有23位

IEEE 754 Converter (JavaScript), V0.22

	Sign	Exponent	Mantissa
Value:	+1	2^{14}	1.831060767173767
Encoded as:	0	141	6971443
Binary:	☐	☑☐☐☐☑☑☐☑	☑☑☐☑☐☑☐☐☑☑☐☐☐☐☐☐☐☑☑☐☐☑☑

You entered	30000.1	+1
Value actually stored in float:	30000.099609375	
Error due to conversion:	-0.000390625	-1
Binary Representation	01000110111010100110000000110011	
Hexadecimal Representation	0x46ea6033	

IEEE 754 Converter (JavaScript), V0.22

	Sign	Exponent	Mantissa
Value:	+1	2^{14}	1.8310546875
Encoded as:	0	141	6971392
Binary:	☐	☑☐☐☐☑☑☐☑	☑☑☐☑☐☑☐☐☑☑☐☐☐☐☐☐☐☐☐☐☐☐☐

You entered	30000.0001	+1
Value actually stored in float:	30000	
Error due to conversion:	-0.0001	-1
Binary Representation	01000110111010100110000000000000	
Hexadecimal Representation	0x46ea6000	

浮点运算的正面与反面

- 不足
  - IEEE754单精度浮点的精度有限，因为尾数只有23位

IEEE 754 Converter (JavaScript), V0.22

	Sign	Exponent	Mantissa
Value:	+1	2^{14}	1.8310546875
Encoded as:	0	141	6971392
Binary:	0	10001101	11010100110000000000000

You entered	30000.0001
Value actually stored in float:	30000
Error due to conversion:	-0.0001
Binary Representation	01000110111010100110000000000000
Hexadecimal Representation	0x46ea6000

+1 -1

IEEE 754 Converter (JavaScript), V0.22

	Sign	Exponent	Mantissa
Value:	+1	2^{4}	1.8750061988830566
Encoded as:	0	131	7340084
Binary:	0	10000011	11100000000000000110100

You entered	30.0001
Value actually stored in float:	30.000099182128906
Error due to conversion:	-8.1787109375E-7
Binary Representation	01000001111100000000000000110100
Hexadecimal Representation	0x41f00034

+1 -1

浮点运算的正面与反面

- 优点
  - 代码书写方便
  - 不需要各种定标转换
- 不足
  - 需要更高级的器件
  - IEEE754单精度浮点的精度有限，因为尾数只有23位

如果不使用FPU，该如何处理浮点数？

浮点数的智能定点格式-IQmath

- "I" => 整数部分
- "Q"=>小数部分

$$-2^{I-1} + \ldots + 2^{1} + 2^{0} \bullet 2^{-1} + 2^{-2} + \ldots + 2^{-Q}$$

I8Q24的例子: 0x41200000

= 0100 0001 . 0010 0000 0000 0000 0000 0000 b

= $2^{6} + 2^{0} + 2^{-3}$ = 65.125

优势 ⇒ 同一个Q值下的数值，精度一样
不足⇒ 与浮点表示相比，动态范围有限

浮点数的智能定点格式-IQmath

Data Type	Range		Resolution/Precision
	Min	Max	
_iq30	-2	1.999 999 999	0.000 000 001
_iq29	-4	3.999 999 998	0.000 000 002
_iq28	-8	7.999 999 996	0.000 000 004
_iq27	-16	15.999 999 993	0.000 000 007
_iq26	-32	31.999 999 985	0.000 000 015
_iq25	-64	63.999 999 970	0.000 000 030
_iq24	-128	127.999 999 940	0.000 000 060
_iq23	-256	255.999 999 981	0.000 000 119
_iq22	-512	511.999 999 762	0.000 000 238
_iq21	-1024	1023.999 999 523	0.000 000 477
_iq20	-2048	2047.999 999 046	0.000 000 954
_iq19	-4096	4095.999 998 093	0.000 001 907
_iq18	-8192	8191.999 996 185	0.000 003 815
_iq17	-16384	16383.999 992 371	0.000 007 629
_iq16	-32768	32767.999 984 741	0.000 015 259
_iq15	-65536	65535.999 969 482	0.000 030 518
_iq14	-131072	131071.999 938 965	0.000 061 035
_iq13	-262144	262143.999 877 930	0.000 122 070
_iq12	-524288	524287.999 755 859	0.000 244 141
_iq11	-1048576	1048575.999 511 719	0.000 488 281
_iq10	-2097152	2097151.999 023 437	0.000 976 563
_iq9	-4194304	4194303.998 046 875	0.001 953 125
_iq8	-8388608	8388607.996 093 750	0.003 906 250
_iq7	-16777216	16777215.992 187 500	0.007 812 500
_iq6	-33554432	33554431.984 375 000	0.015 625 000
_iq5	-67108864	67108863.968 750 000	0.031 250 000
_iq4	-134217728	134217727.937 500 000	0.062 500 000
_iq3	-268435456	268435455.875 000 000	0.125 000 000
_iq2	-536870912	536870911.750 000 000	0.250 000 000
_iq1	-1073741824	1 073741823.500 000 000	0.500 000 000

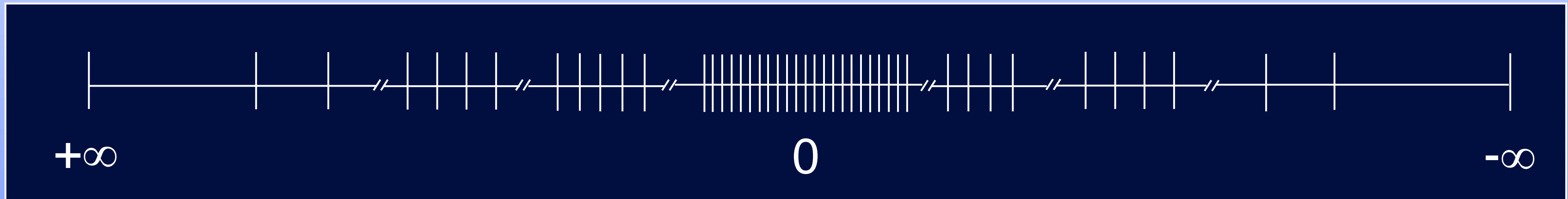
“I” => 整数部分

“Q”=>小数部分

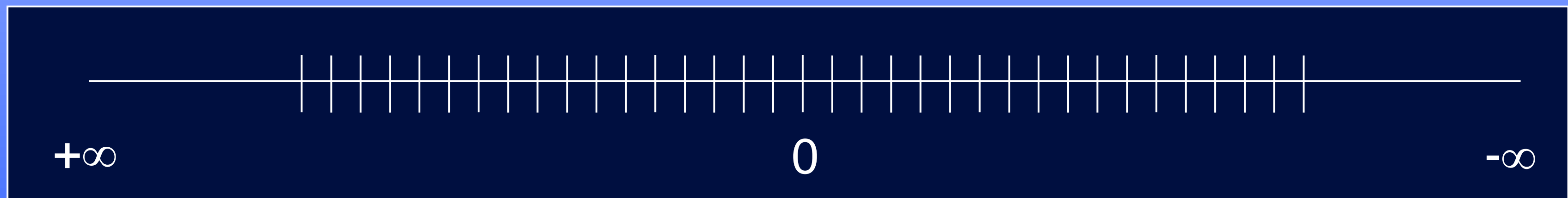
“C28x IQmath Library”——Texas Instruments

数轴上的分布对比

浮点数: 非均匀分布(精度可变)



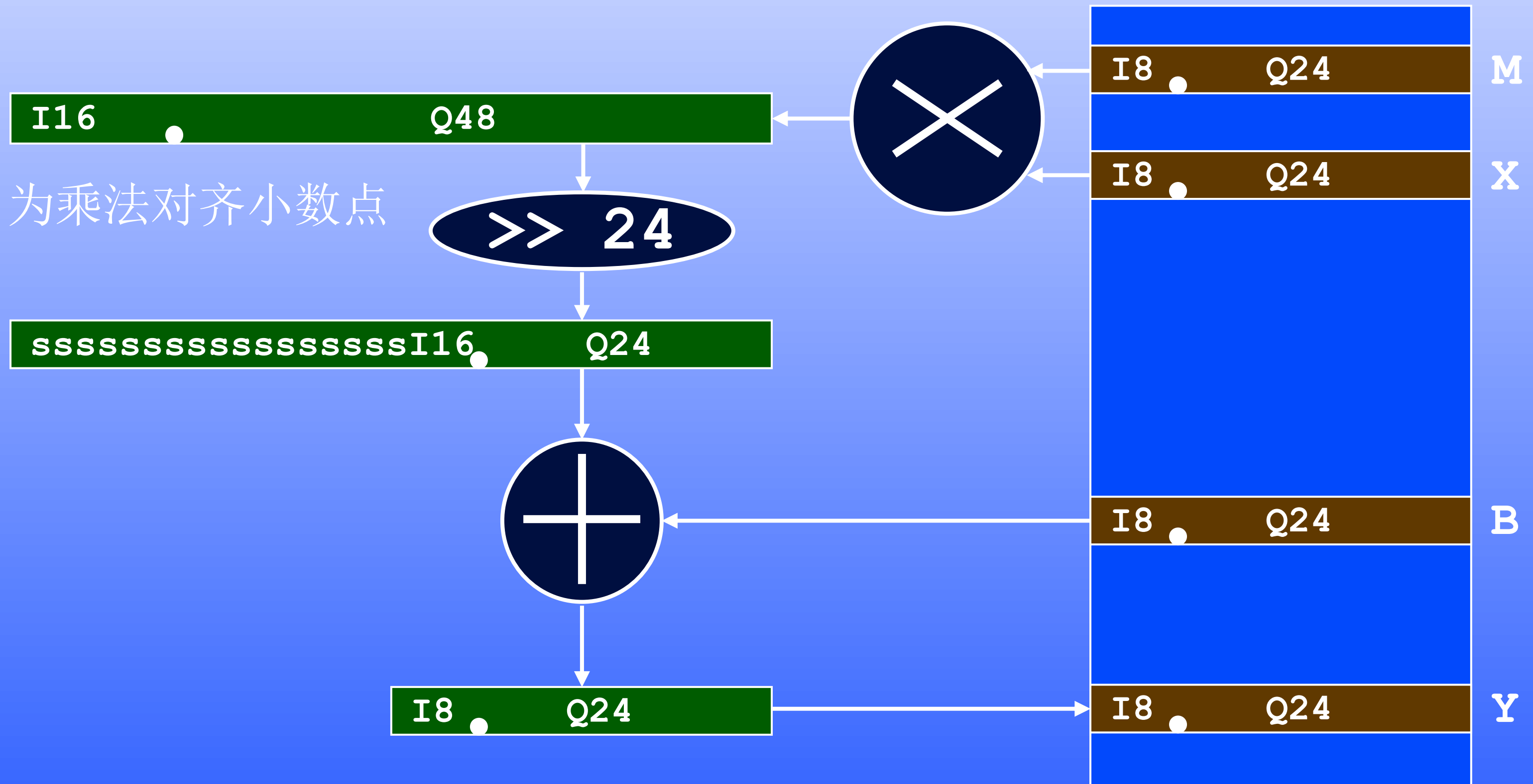
IQ格式: 均匀分布(精度一致)



- 32位的浮点数和IQ格式表示的定点数，在数轴上都有 2^{32} 种可表示的值
- 浮点数可表达数值范围大得多，靠近0点精度大得多
- IQ格式可表达数值范围小，精度一致

32位 IQmath法

y = mx + b



用C代码表示

```
Y = ((int64) M * (int64) X) >> Q + B;
```

乘法操作

```
Y = ((i64) M * (i64) X) >> Q + B;
```

重定义乘法操作:

```
_IQmpy(M,X) == ((i64) M * (i64) X) >> Q
```

最终简化为:

```
Y = _IQmpy(M,X) + B;
```

C28x的编译器支持内建的“_IQmpy”的函数，生成的汇编代码示例:

```
MOVL    XT,@M
IMPYL   P,XT,@X        ; P   = low  32-bits of M*X
QMPYL   ACC,XT,@X      ; ACC = high 32-bits of M*X
LSL64   ACC:P,#(32-Q)  ; ACC = ACC:P << 32-Q
                       ; (same as P = ACC:P >> Q)
ADDL    ACC,@B         ; Add B
MOVL    @Y,ACC         ; Result = Y = _IQmpy(M*X) + B
; 7 Cycles
```

IQmath 方法
形式上与浮点数写法类似

浮点数

```
float  Y, M, X, B;

Y = M * X + B;
```

C代码下的
“IQmath”

```
_iq  Y, M, X, B;

Y = _IQmpy(M, X) + B;
```

C++代码下的
“IQmath”

```
iq  Y, M, X, B;

Y = M * X + B;
```

IQmath 方法
使用全局的GLOBAL_Q来简化书写

为整个工程定义一个全局的GLOBAL_Q来简化书写

- GLOBAL_Q

Q的选择根据数值的精度和范围来决定：

GLOBAL_Q	Max Val	Min Val	Resolution
28	7.999 999 996	-8.000 000 000	0.000 000 004
24	127.999 999 94	-128.000 000 00	0.000 000 06
20	2047.999 999	-2048.000 000	0.000 001

```
#define  GLOBAL_Q  18      // set in “IQmathLib.h” file
_iq  Y, M, X, B;
Y = _IQmpy(M,X) + B;       // all values are in Q = 18
```

特殊情况下可以指定的Q值用于不同范围、精度下的运算

```
_iq20  Y, M, X, B;
Y = _IQ20mpy(M,X) + B;     // all values are in Q = 20
```

Iqmath提供了浮点数与定点数之间的兼容性(提高代码在不同硬件之间的可移植性)

1) 可统一使用IQ函数

```
Y = _IQmpy(M, X) + B;
```

2) 在IQmathLib.h中定义浮点数的表示方法

```
#if MATH_TYPE == IQ_MATH
```

```
#if MATH_TYPE == FLOAT_MATH
```

3) 编译器会自动执行相应的转换

```
Y = (float)M * (float)X + (float)B;
```

定点代码

可在定点、不含FPU的
F282xx
系列上编译、运行

浮点代码

可在浮点、含FPU的
F283xx *
系列上编译、运行

所有的"IQmath"都有相对应的浮点运算

* 还可以在其它浮点编译环境下运行 (例如PC, Matlab, 支持RTS库的定点处理器等.)

IQmath的库函数: 数学与三角函数

<table>
<tr><th>运算</th><th>浮点</th><th>“IQmath”的C代码</th><th>“IQmath” 的C++代码</th></tr>
<tr><td>类型</td><td>float A, B;</td><td>_iq A, B;</td><td>iq A, B;</td></tr>
<tr><td>常数</td><td>A = 1.2345</td><td>A = _IQ(1.2345)</td><td>A = IQ(1.2345)</td></tr>
<tr><td>乘法</td><td>A * B</td><td>_IQmpy(A , B)</td><td>A * B</td></tr>
<tr><td>除法</td><td>A / B</td><td>_IQdiv (A , B)</td><td>A / B</td></tr>
<tr><td>加法</td><td>A + B</td><td>A + B</td><td>A + B</td></tr>
<tr><td>减法</td><td>A - B</td><td>A - B</td><td>A – B</td></tr>
<tr><td>布尔运算</td><td>>, >=, <, <=, ==, |=, &&, ||</td><td>>, >=, <, <=, ==, |=, &&, ||</td><td>>, >=, <, <=, ==, |=, &&, ||</td></tr>
<tr><td>三角函数
幂操作</td><td>sin(A),cos(A)
sin(A*2pi),cos(A*2pi)
asin(A),acos(A)
atan(A),atan2(A,B)
atan2(A,B)/2pi
sqrt(A),1/sqrt(A)
sqrt(A*A + B*B)</td><td>_IQsin(A), _IQcos(A)
_IQsinPU(A), _IQcosPU(A)
_IQasin(A),_IQacos(A)
_IQatan(A), _IQatan2(A,B)
_IQatan2PU(A,B)
_IQsqrt(A), _IQisqrt(A)
_IQmag(A,B)</td><td>IQsin(A),IQcos(A)
IQsinPU(A),IQcosPU(A)
IQasin(A),IQacos(A)
IQatan(A),IQatan2(A,B)
IQatan2PU(A,B)
IQsqrt(A),IQisqrt(A)
IQmag(A,B)</td></tr>
<tr><td>饱和处理</td><td>exp(A)
if(A > Pos) A = Pos
if(A < Neg) A = Neg</td><td>_IQexp(A)
_IQsat(A,Pos,Neg)</td><td>IQexp(A)
IQsat(A,Pos,Neg)</td></tr>
</table>

最高精度可达28到31位

IQmath的库函数：转换函数

运算	浮点	“IQmath”的C代码	“IQmath” 的C++代码
iq to iqN	A	_IQtoIQN(A)	IQtoIQN(A)
iqN to iq	A	_IQNtoIQ(A)	IQNtoIQ(A)
integer(iq)	(long) A	_IQint(A)	IQint(A)
fraction(iq)	A – (long) A	_IQfrac(A)	IQfrac(A)
iq = iq*long	A * (float) B	_IQmpyI32(A,B)	IQmpyI32(A,B)
integer(iq*long)	(long) (A * (float) B)	_IQmpyI32int(A,B)	IQmpyI32int(A,B)
fraction(iq*long)	A - (long) (A * (float) B)	_IQmpyI32frac(A,B)	IQmpyI32frac(A,B)
qN to iq	A	_QNtoIQ(A)	QNtoIQ(A)
iq to qN	A	_IQtoQN(A)	IQtoQN(A)
string to iq	atof(char)	_atoIQ(char)	atoIQ(char)
IQ to float	A	_IQtoF(A)	IQtoF(A)
IQ to ASCII	sprintf(A,B,C)	_IQtoA(A,B,C)	IQtoA(A,B,C)

IQmath.lib > 包含了所有的库函数
IQmathLib.h > C头文件
IQmathCPP.h > C++头文件

例：交流异步电机的控制算法

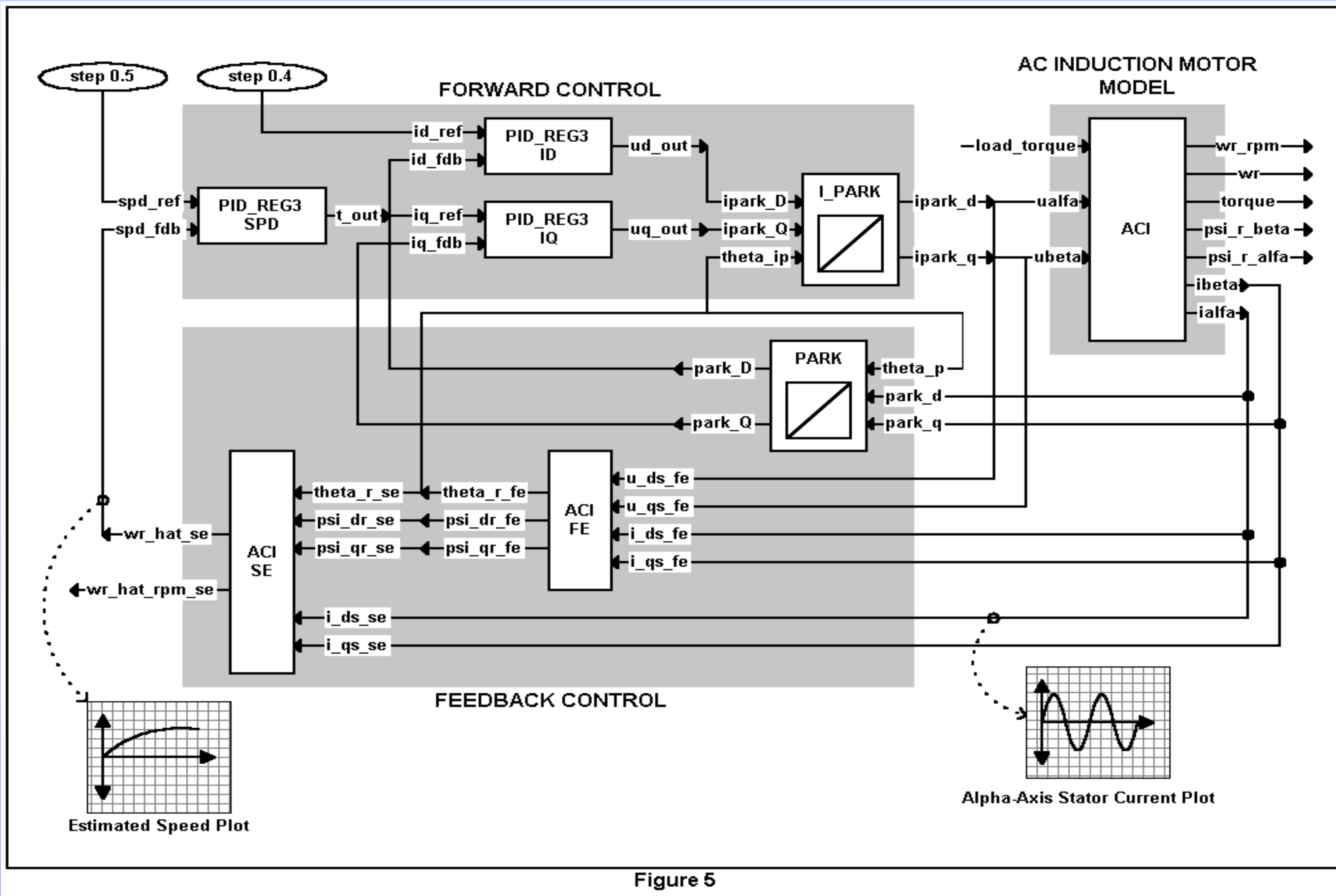


Figure 5

- 无传感器的直接转子磁链定向控制
- 目标: 转速与电机定子电流的估算

以Park变换为例

浮点数的写法

```
#include "math.h"

#define  TWO_PI   6.28318530717959
void park_calc(PARK *v)
{
     float cos_ang , sin_ang;
     sin_ang = sin(TWO_PI * v->ang);
     cos_ang = cos(TWO_PI * v->ang);

     v->de = (v->ds * cos_ang) + (v->qs * sin_ang);
     v->qe = (v->qs * cos_ang) - (v->ds * sin_ang);
}
```

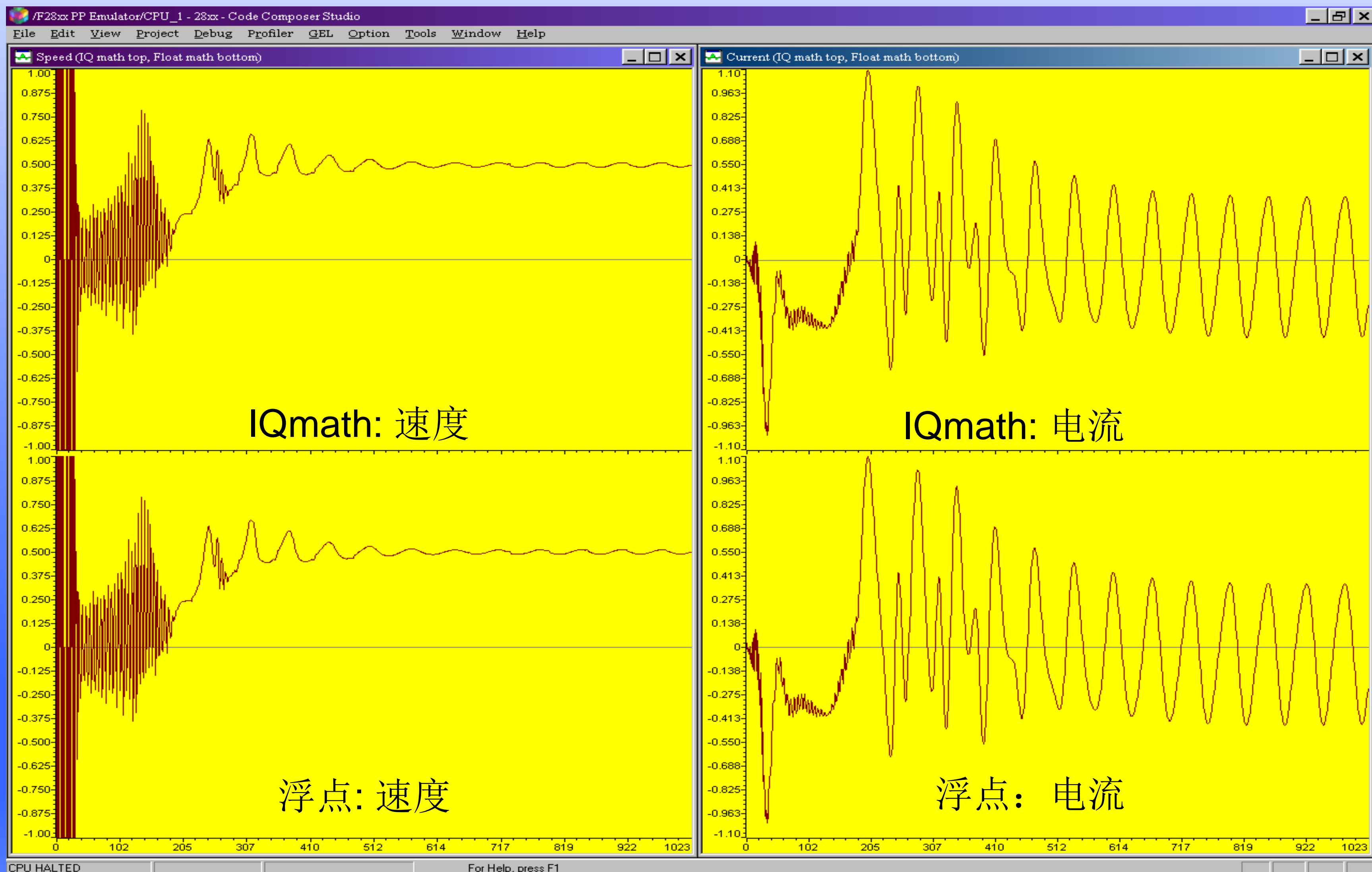
以Park变换为例IQmath的写法

```
#include “math.h”
#include  “IQmathLib.h”
#define  TWO_PI   _IQ(6.28318530717959)
void park_calc(PARK *v)
{
      _iq   cos_ang , sin_ang;
      sin_ang = _IQsin(_IQmpy(TWO_PI , v->ang));
      cos_ang = _IQcos(_IQmpy(TWO_PI , v->ang));

      v->de = _IQmpy(v->ds , cos_ang) + _IQmpy(v->qs , sin_ang);
      v->qe = _IQmpy(v->qs , cos_ang) - _IQmpy(v->ds , sin_ang);
}
```

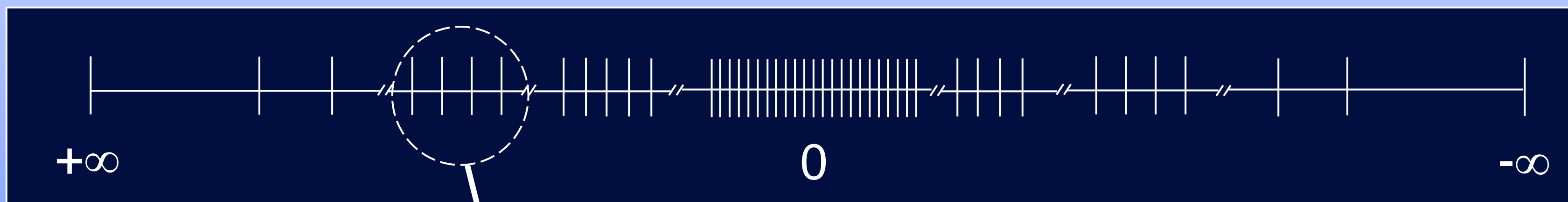
交流异步电机的控制算法

配置GLOBAL_Q = 24, 系统可稳定运行



IQ24与IEEE754的精度在特定区域一致

浮点:

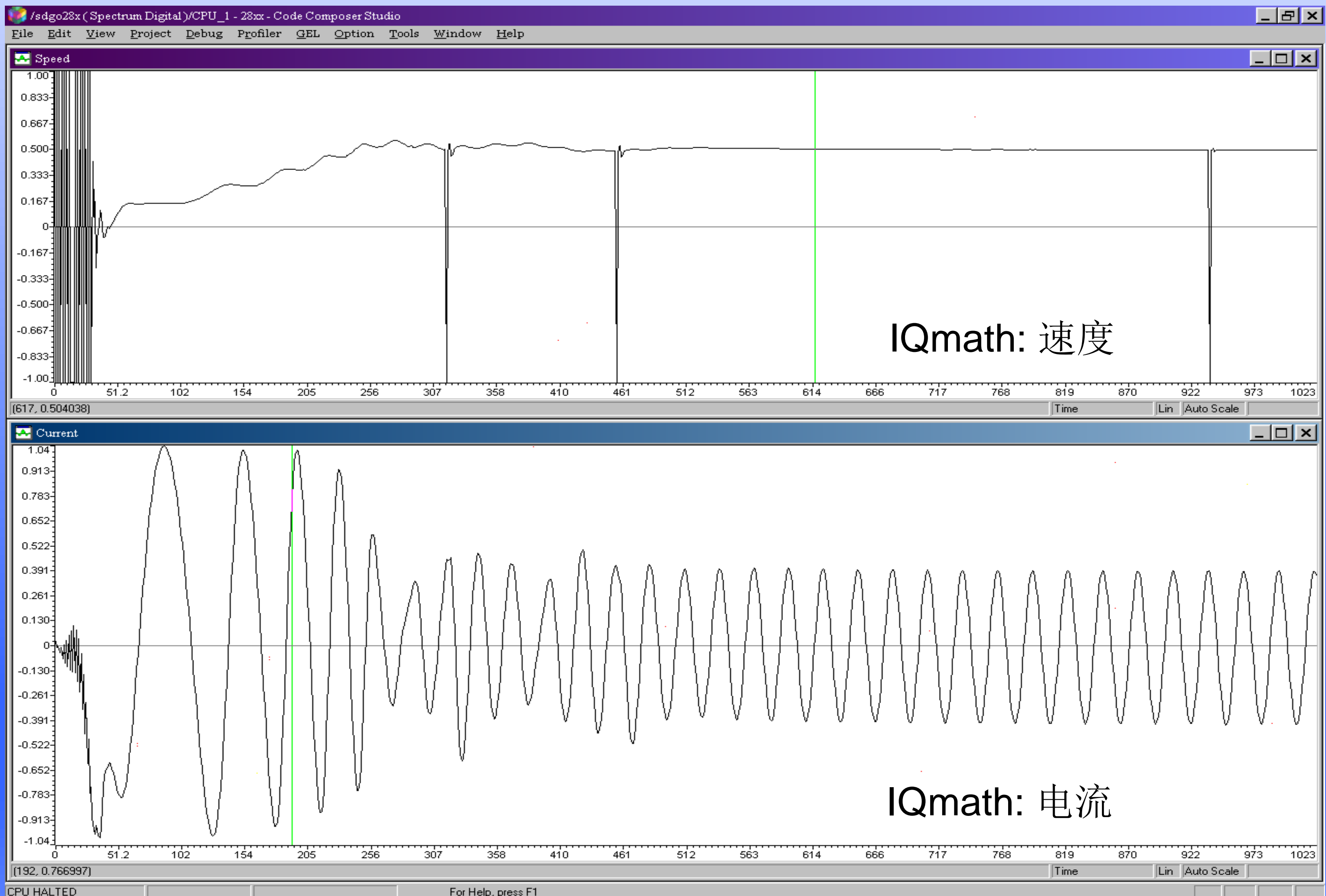


精度相同

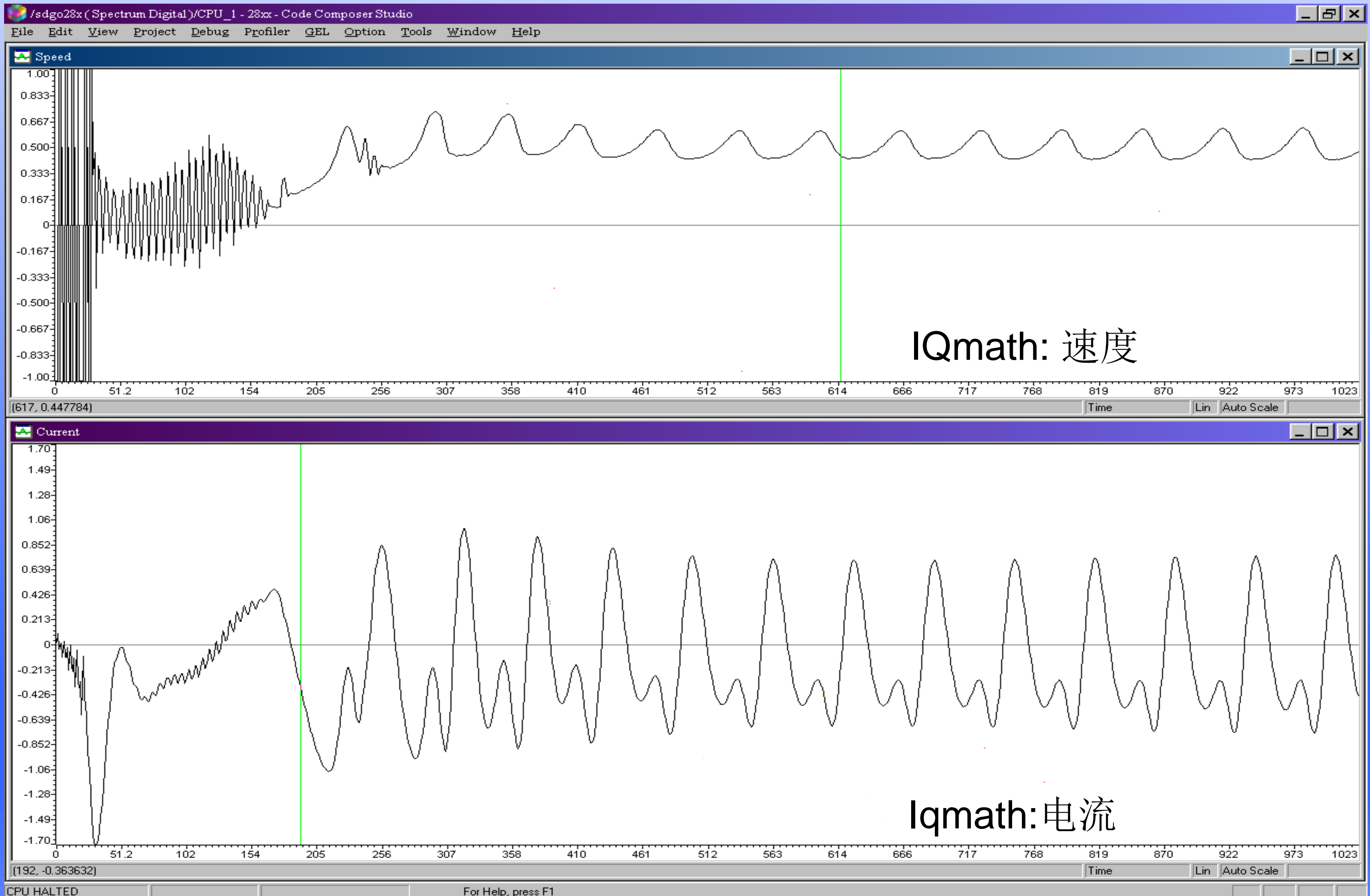
I8Q24 定点:

特定计算出现的区域，精度相同！

配置GLOBAL_Q = 27, 此时系统不稳定
因为可表示的数的范围不够了



配置GLOBAL_Q = 16, 此时系统不稳定因为可表示的数的精度不够了



交流异步电机的控制算法

系统可稳定运行的Q范围（基于此例程，不一定适用于各种算法）

Q的范围	稳定程度
Q31 to Q27	不稳定，表示的数的范围不够
Q26 to Q19	稳定的
Q18 to Q0	不稳定，分辨率/精度不够，产生了较大的量化误差

GLOBAL_Q的选取需要对所有变量的范围进行预估，然后进行测试