FixedPointC

December 5, 2020

1 Fixed-point Exercise

Suppose you are working on a thermostat for ACME Corporation. In order to reduce production and power costs, they have decided to choose a microprocessor without an FPU. Your job is to take temperature data from the sensor, which outputs in Celsius, and convert it to Fahrenheit for the user interface team. They want to be able to display up to 1 decimal place accurately. You are free to decide on the appropriate `QM.N` format. Note, the formula for convertion is:

$$F = \frac{9}{5}C + 32$$

1. How many bits are needed for fractional portion in order to achieve one decimal place of accuracy?
2. Given temperature sensor has a range of -100° to 155° in Celsius, which corresponds to roughly -150° to +310° degrees Fahrenheit. How many bits do we need for the integer portion?

[2]:
```c
#include <stdint.h>

typedef int16_t q10x6_t;
typedef int32_t q20x12_t;

q10x6_t celsius_to_fahrenheit(q10x6_t celsius)
{
    const q10x6_t C0 = 115;  /* 9/5 in Q10.6 */
    const q10x6_t C1 = 2048; /* 32 in Q10.6 */

    /* Multiplying Q10.6 by Q10.6 -> Q20.12 */
    q20x12_t y = ((q20x12_t)celsius) * ((q20x12_t) C0);

    /* Rounding to nearest 6 bits */
    y += (y < 0) ? -0x10 : +0x10;
    y >>= 6; /* Q20.6 */

    /* Add 32 to result */
    y += C1;
    return (q10x6_t)y;
}
```

```
#define TESTING 1

#if (TESTING == 1)
//%cflags: -Wall -Wextra -lm

#include <math.h>
#include <stdio.h>
#include <inttypes.h>

double qminv(int m, int n);
double qmaxv(int m, int n);
long ftoq(double x, int m, int n);
double qtof(long x, int m, int n);
double clip(double v, double vmin, double vmax);

#define FRM (10)
#define FRN (6)

int main()
{
    for (double c = -100.0; c <= 155.0; c += 15.3) {
        q10x6_t qc = ftoq(c, FRM, FRN);
        q10x6_t qf = celsius_to_fahrenheit(qc);
        double ff = 9.0 / 5.0 * c + 32.0;
        double fq = qtof(qf, FRM, FRN);
        printf("% 3.1f \t% 3.1f \t% 3.1f \t%hd \t%hd\n", c, ff, fq, qc, qf);
    }
}

long ftoq(double x, int m, int n)
{
    return (long)(rintl(ldexp(clip(x, qminv(m, n), qmaxv(m,n)), n)));
}

double qtof(long x, int m, int n)
{
     return clip(ldexp((double)x, -n), qminv(m, n), qmaxv(m, n));
}

double qmaxv(int m, int n)
{
    const double one = 1.0;
    return ldexp(one, m - 1) - ldexp(one, -n);
}

double qminv(int m, int n)
{
```

```
    (void)(n);
    return ldexp(-1.0, m - 1);
}

double clip(double v, double vmin, double vmax)
{
    return fmax(vmin, fmin(v, vmax));
}

#endif
```

```
-100.0  -148.0  -147.7  -6400   -9453
-84.7   -120.5  -120.2  -5421   -7694
-69.4   -92.9   -92.7   -4442   -5934
-54.1   -65.4   -65.2   -3462   -4174
-38.8   -37.8   -37.7   -2483   -2414
-23.5   -10.3   -10.2   -1504   -655
-8.2     17.2    17.2   -525    1104
 7.1     44.8    44.8   454     2864
 22.4    72.3    72.2   1434    4624
 37.7    99.9    99.8   2413    6384
 53.0    127.4   127.2  3392    8143
 68.3    154.9   154.7  4371    9902
 83.6    182.5   182.2  5350    11661
 98.9    210.0   209.7  6330    13422
 114.2   237.6   237.2  7309    15181
 129.5   265.1   264.7  8288    16940
 144.8   292.6   292.2  9267    18699
```

2 FIR filters in fixed-point

```
#include <stdint.h>

typedef int16_t q15_t;

typedef struct fir_instance_q15
{
    q15_t    *coeff;
    q15_t    *state;
    uint32_t index;
    uint32_t taps;
} fir_q15_t;

void generic_fir_q15(fir_q15_t     *self,
                     q15_t         *dst,
                     const q15_t   *src,
                     uint32_t      frames);
```

1. CMSIS DSP documentation for fixed-point FIR filter
2. CMSIS DSP implementation for FIR filter arm_fir_q15.c

[7]:
```c
#include <stdint.h>

typedef int16_t q15_t;
typedef int64_t q63_t;
typedef int32_t q31_t;

static q15_t q15_rsat(q63_t x);

typedef struct fir_instance_q15
{
    q15_t    *coeff;
    q15_t    *state;
    uint32_t index;
    uint32_t taps;
} fir_q15_t;

void generic_fir_q15(fir_q15_t     *self,
                     q15_t         *dst,
                     const q15_t   *src,
                     uint32_t      frames)
{
    uint32_t taps = self->taps;
    uint32_t index = self->index;

    q15_t *state = self->state;
    const q15_t *coeff = self->coeff;

    for (uint32_t i = 0; i < frames; ++i) {

        /* Place input in circular buffer */
        state[index] = src[i];

        /* Circularly increment index */
        ++index;
        if (index >= taps)
            index = 0;

        q63_t acc = 0; /* Why did I use 64 bits here, why not 32 bit␣
↪accumulator? */

        /* Convolution Loop */
        do {
            for (uint32_t j = 0, k = index -1; j < index; ++j, --k)
                acc += ((q31_t) coeff[j]) * ((q31_t) state[k]);
```

```
            for (uint32_t j = index, k = taps - 1; j < taps; ++j, --k)
                acc += ((q31_t)coeff[j]) * ((q31_t) state[k]);
        } while(0);

        dst[i] = q15_rsat(acc); /* Typically this is a machine instruction */
    }

    self->index = index;
}

static q15_t q15_rsat(q63_t x)
{
    const q63_t vmax =  32767;
    const q63_t vmin = -32768;

    /* Q1.15 * Q1.15 -> Q2.30 */
    /* Round to nearest Q1.15 */
    x += (x < 0) ? vmin : vmax;

    /* Rescale to Q1.15 */
    x = x >> 15;

    /* Saturate result */
    x = (x >= vmax) ? vmax : x;
    x = (x <= vmin) ? vmin : x;

    return (q15_t)x;
}

#define TESTING (1)
#if (TESTING == 1)
//%cflags: -Wall -Wextra -lm

#include <math.h>
#include <stdio.h>
#include <inttypes.h>

double qminv(int m, int n);
double qmaxv(int m, int n);
long ftoq(double x, int m, int n);
double qtof(long x, int m, int n);
double clip(double v, double vmin, double vmax);

#define FRM  (1)
#define FRN  (15)
#define TAPS (33)
#define BUF  (16)
```

```
int main()
{
/*<autogen-fir>*/
q15_t fir_coeff[45] =␣
 ↪{171,-201,-209,1,148,-33,-230,-29,271,90,-326,-188,375,328,-420,-537,458,888,-487,-1643,506
 ↪
q15_t fir_state[45] = {0};
uint32_t fir_taps = 45;
uint32_t fir_index = 0;
fir_q15_t fir_filter = { fir_coeff, fir_state, fir_index, fir_taps };
/*</autogen-fir>*/

    q15_t ibuf[BUF] = {0};
    q15_t obuf[BUF] = {0};

    for (int i = 0; i < 3; ++i) {
        if (i == 0)
            ibuf[0] = 32767; // 0.99999...
        else
            ibuf[0] = 0;

        /* Process FIR filter */
        generic_fir_q15(&fir_filter, obuf, ibuf, BUF);

        /* Print input and output buffers */
        for (int j = 0; j < BUF; ++j) {
            printf("%d \t%d \t%f \t%f\n",
                   ibuf[j], obuf[j],
                   qtof(ibuf[j], FRM, FRN), qtof(obuf[j], FRM, FRN));
        }
    }
}

long ftoq(double x, int m, int n)
{
    return (long)(rintl(ldexp(clip(x, qminv(m, n), qmaxv(m,n)), n)));
}

double qtof(long x, int m, int n)
{
     return clip(ldexp((double)x, -n), qminv(m, n), qmaxv(m, n));
}

double qmaxv(int m, int n)
{
    const double one = 1.0;
```

```
    return ldexp(one, m - 1) - ldexp(one, -n);
}

double qminv(int m, int n)
{
    (void)(n);
    return ldexp(-1.0, m - 1);
}

double clip(double v, double vmin, double vmax)
{
    return fmax(vmin, fmin(v, vmax));
}

#endif
```

```
32767   171     0.999969        0.005219
0       -202    0.000000        -0.006165
0       -210    0.000000        -0.006409
0       1       0.000000        0.000031
0       148     0.000000        0.004517
0       -34     0.000000        -0.001038
0       -231    0.000000        -0.007050
0       -30     0.000000        -0.000916
0       271     0.000000        0.008270
0       90      0.000000        0.002747
0       -327    0.000000        -0.009979
0       -189    0.000000        -0.005768
0       375     0.000000        0.011444
0       328     0.000000        0.010010
0       -421    0.000000        -0.012848
0       -538    0.000000        -0.016418
0       458     0.000000        0.013977
0       888     0.000000        0.027100
0       -488    0.000000        -0.014893
0       -1644   0.000000        -0.050171
0       506     0.000000        0.015442
0       5183    0.000000        0.158173
0       7681    0.000000        0.234406
0       5183    0.000000        0.158173
0       506     0.000000        0.015442
0       -1644   0.000000        -0.050171
0       -488    0.000000        -0.014893
0       888     0.000000        0.027100
0       458     0.000000        0.013977
0       -538    0.000000        -0.016418
0       -421    0.000000        -0.012848
```

```
0       328     0.000000        0.010010
0       375     0.000000        0.011444
0       -189    0.000000        -0.005768
0       -327    0.000000        -0.009979
0       90      0.000000        0.002747
0       271     0.000000        0.008270
0       -30     0.000000        -0.000916
0       -231    0.000000        -0.007050
0       -34     0.000000        -0.001038
0       148     0.000000        0.004517
0       1       0.000000        0.000031
0       -210    0.000000        -0.006409
0       -202    0.000000        -0.006165
0       171     0.000000        0.005219
0       0       0.000000        0.000000
0       0       0.000000        0.000000
0       0       0.000000        0.000000
```

3 IIR filters in fixed-point

```c
#include <stdint.h>

typedef int16_t q15_t;

typedef struct iir_instance_q15
{
    q15_t    *coeff; /* b0, b1, b2, a1, a2... */
    q15_t    *state; /* x1, x2, y1, y2, ... */
    uint32_t nstage;
    int32_t  shift;
} iir_q15_t;

void generic_iir_q15(iir_q15_t     *self,
                     q15_t         *dst,
                     const q15_t   *src,
                     uint32_t      frames);
```

1. CCRMA's info on Direct form I
2. CMSIS DSP documentation on fixed-point IIR filters
3. CMSIS DSP implementation of IIR filter arm_biquad_cascade_df1_q15.c

[4]:
```c
/* IIR Filter API */
#include <stdint.h>

typedef int16_t q15_t;
typedef int64_t q63_t;
typedef int32_t q31_t;
```

```c
typedef struct iir_instance_q15
{
    q15_t    *coeff; /* b0, b1, b2, a1, a2... */
    q15_t    *state; /* x1, x2, y1, y2, ... */
    uint32_t nstage;
    int32_t  shift;
} iir_q15_t;

void generic_iir_q15(iir_q15_t     *self,
                     q15_t         *dst,
                     const q15_t   *src,
                     uint32_t      frames);

/* IIR Filter Implementation */

static q15_t q15_rsat(q63_t x, int32_t shift);

void generic_iir_q15(iir_q15_t     *self,
                     q15_t         *dst,
                     const q15_t   *src,
                     uint32_t      frames)
{
    const uint32_t nstage = self->nstage;
    const int32_t shift = self->shift;

    const q15_t *coeff = self->coeff;
    q15_t *state = self->state;

    for (uint32_t i = 0; i < nstage; ++i) {

        const q31_t b0 = coeff[0];
        const q31_t b1 = coeff[1];
        const q31_t b2 = coeff[2];
        const q31_t a1 = coeff[3];
        const q31_t a2 = coeff[4];

        q31_t x0, x1, x2, y0, y1, y2;

        x1 = state[0];
        x2 = state[2];
        y1 = state[3];
        y2 = state[4];

        for (uint32_t j = 0; j < frames; ++j) {
            q63_t acc;

            x0 = src[j];
```

```
            acc  = (b0 * x0);
            acc += (b1 * x1);
            acc += (b2 * x2);
            acc -= (a1 * y1);
            acc -= (a2 * y2);
            y0 = q15_rsat(acc, shift);

            x2 = x1; /* new x[n-1] */
            x1 = x0; /* new x[n-2] */
            y2 = y1; /* new y[n-1] */
            y1 = y0; /* new y[n-2] */

            dst[j] = y0;
        }

        state[0] = x1;
        state[1] = x2;
        state[2] = y1;
        state[3] = y2;

        state += 4;
        coeff += 5;
    }
}

static q15_t q15_rsat(q63_t x, int32_t shift)
{
    const q63_t vmax =  32767;
    const q63_t vmin = -32768;

    /* Q1.15 * Q1.15 -> Q2.30 */
    /* Round to nearest Q1.15 */
    x += (x < 0) ? vmin : vmax;

    /* Rescale to Q1.15 */
    x >>= shift;

    /* Saturate result */
    x = (x >= vmax) ? vmax : x;
    x = (x <= vmin) ? vmin : x;

    return (q15_t)x;
}

#define TESTING (1)
```

```c
#if (TESTING == 1)
//%cflags: -Wall -Wextra -lm

#include <math.h>
#include <stdio.h>
#include <inttypes.h>

double qminv(int m, int n);
double qmaxv(int m, int n);
long ftoq(double x, int m, int n);
double qtof(long x, int m, int n);
double clip(double v, double vmin, double vmax);

#define FRM  (1)
#define FRN  (15)
#define BUF  (32)
#define IIR_STAGE (4)

int main()
{
/*<autogen-iir>*/
#define NSTAGE (2)
q15_t iir_coeff[5 * NSTAGE] = {802,802,0,-5068,0,8192,9857,8192,-1394,6672};
q15_t iir_state[4 * NSTAGE] = {0};
uint32_t iir_stage = NSTAGE;
int32_t iir_shift = 13;
iir_q15_t iir_filter = {iir_coeff, iir_state, iir_stage, iir_shift};
/*</autogen-iir>*/

    q15_t ibuf[BUF] = {0};
    q15_t obuf[BUF] = {0};

    for (int i = 0; i < 3; ++i) {
        if (i == 0)
            ibuf[0] = 32767; // 0.99999...
        else
            ibuf[0] = 0;

        /* Process IIR filter */
        generic_iir_q15(&iir_filter, obuf, ibuf, BUF);

        /* Print input and output buffers */
        for (int j = 0; j < BUF; ++j) {
            printf("% 6d \t% 6d \t% 0.8f \t% 0.8f\n",
                   ibuf[j], obuf[j],
                   qtof(ibuf[j], FRM, FRN), qtof(obuf[j], FRM, FRN));
        }
```

```
    }
}

long ftoq(double x, int m, int n)
{
    return (long)(rintl(ldexp(clip(x, qminv(m, n), qmaxv(m,n)), n)));
}

double qtof(long x, int m, int n)
{
     return clip(ldexp((double)x, -n), qminv(m, n), qmaxv(m, n));
}

double qmaxv(int m, int n)
{
    const double one = 1.0;
    return ldexp(one, m - 1) - ldexp(one, -n);
}

double qminv(int m, int n)
{
    (void)(n);
    return ldexp(-1.0, m - 1);
}

double clip(double v, double vmin, double vmax)
{
    return fmax(vmin, fmin(v, vmax));
}

#endif
```

```
32767   32117   0.99996948      0.98013306
    0   32767   0.00000000      0.99996948
    0   12189   0.00000000      0.37197876
    0  -24618   0.00000000     -0.75128174
    0  -14121   0.00000000     -0.43093872
    0   17651   0.00000000      0.53866577
    0   14508   0.00000000      0.44274902
    0  -11912   0.00000000     -0.36352539
    0  -13848   0.00000000     -0.42260742
    0    7349   0.00000000      0.22427368
    0   12533   0.00000000      0.38247681
    0   -3857   0.00000000     -0.11770630
    0  -10868   0.00000000     -0.33166504
    0    1295   0.00000000      0.03952026
    0    9075   0.00000000      0.27694702
```

```
0     493   0.00000000      0.01504517
0   -7312   0.00000000     -0.22314453
0   -1650   0.00000000     -0.05035400
0    5678   0.00000000      0.17327881
0    2314   0.00000000      0.07061768
0   -4235   0.00000000     -0.12924194
0   -2610   0.00000000     -0.07965088
0    3009   0.00000000      0.09182739
0    2641   0.00000000      0.08059692
0   -2006   0.00000000     -0.06121826
0   -2497   0.00000000     -0.07620239
0    1212   0.00000000      0.03698730
0    2243   0.00000000      0.06845093
0    -610   0.00000000     -0.01861572
0   -1935   0.00000000     -0.05905151
0     171   0.00000000      0.00521851
0    1609   0.00000000      0.04910278
0     988   0.00000000      0.03015137
0      32   0.00000000      0.00097656
0    -804   0.00000000     -0.02453613
0    -167   0.00000000     -0.00509644
0     630   0.00000000      0.01922607
0     247   0.00000000      0.00753784
0    -476   0.00000000     -0.01452637
0    -287   0.00000000     -0.00875854
0     342   0.00000000      0.01043701
0     295   0.00000000      0.00900269
0    -233   0.00000000     -0.00711060
0    -284   0.00000000     -0.00866699
0     145   0.00000000      0.00442505
0     259   0.00000000      0.00790405
0     -79   0.00000000     -0.00241089
0    -229   0.00000000     -0.00698853
0      29   0.00000000      0.00088501
0     195   0.00000000      0.00595093
0      13   0.00000000      0.00039673
0    -161   0.00000000     -0.00491333
0     -42   0.00000000     -0.00128174
0     127   0.00000000      0.00387573
0      59   0.00000000      0.00180054
0     -98   0.00000000     -0.00299072
0     -69   0.00000000     -0.00210571
0      72   0.00000000      0.00219727
0      72   0.00000000      0.00219727
0     -51   0.00000000     -0.00155640
0     -72   0.00000000     -0.00219727
0      33   0.00000000      0.00100708
0      68   0.00000000      0.00207520
```

```
0     -20   0.00000000     -0.00061035
0    -666   0.00000000     -0.02032471
0    -173   0.00000000     -0.00527954
0     516   0.00000000      0.01574707
0     232   0.00000000      0.00708008
0    -385   0.00000000     -0.01174927
0    -259   0.00000000     -0.00790405
0     273   0.00000000      0.00833130
0     261   0.00000000      0.00796509
0    -182   0.00000000     -0.00555420
0    -248   0.00000000     -0.00756836
0     110   0.00000000      0.00335693
0     224   0.00000000      0.00683594
0     -56   0.00000000     -0.00170898
0    -196   0.00000000     -0.00598145
0      16   0.00000000      0.00048828
0     166   0.00000000      0.00506592
0      19   0.00000000      0.00057983
0    -136   0.00000000     -0.00415039
0     -43   0.00000000     -0.00131226
0     107   0.00000000      0.00326538
0      57   0.00000000      0.00173950
0     -82   0.00000000     -0.00250244
0     -65   0.00000000     -0.00198364
0      59   0.00000000      0.00180054
0      66   0.00000000      0.00201416
0     -41   0.00000000     -0.00125122
0     -65   0.00000000     -0.00198364
0      26   0.00000000      0.00079346
0      61   0.00000000      0.00186157
0     -15   0.00000000     -0.00045776
0     -57   0.00000000     -0.00173950
0       6   0.00000000      0.00018311
```