# Filter C Programming

## (4A) Waveform Generator

Young Won Lim
4/7/20

Please send corrections (or suggestions) to youngwlim@hotmail.com.
This document was produced by using LibreOffice.

# Based on

Introduction to Signal Processing

S. J. Ofranidis

# Sinusoidal Generators

```
for n = 0, 1, 2, . . . do:
    w0 = (2 R cos ω0 ) w1 − R^2 w2 + δ(n)
    y = (R sin ω0 ) w1
    w2 = w1
    w1 = w0
```

```
for n = 0, 1, 2, . . . do:
    w0 = (2 R cos ω0 ) w1 − R^2 w2 + δ(n)
    y = w0 - (R sin ω0 ) w1
    w2 = w1
    w1 = w0
```

# Sinusoidal Generators

```
for each input sample x do:
        y1 = a w1 − b w2 + x
        y2 = a w2 + b w1
        w1 = y1
        w2 = y2
```

```
for n = 0, 1, 2, . . . do:
        v0 = δ(n)
        y = w0 = w4 + b0 v0 + b1 v1 + b2 v2 + b3 v3
        delay ( 3, v )
        delay ( 4, w )
```

# Sinusoidal Generators

```
for n = 0, 1, . . . , D−1 do:
    w0 = bn
    delay (D, w)
```

```
repeat forever :
    w0 = wD
    delay (D, w)
```

```
for n = 0, 1, 2, . . . do:
    w0 = wD + δ(n)
    y = b0 w0 + b1 w1 + · · · + bD−1 wD−1
    delay (D, w)
```

# Sinusoidal Generators

```
for n = 0, 1, . . . , D−1 do:
    ∗p = bn
    cdelay (D, w, &p)
```

```
repeat forever :
    ∗p = tap(D, w, p, D)
    cdelay (D, w, &p)
```

```
for n = 0, 1, . . . , D−1 do:
    w[q]= bn
    cdelay2 (D, &q)
```

# Sinusoidal Generators

```
repeat forever :
    w[q]= tap2 (D, w, q, D)
    cdelay2 (D, & q)
```

```
repeat forever :
    output y = w[q]
    cdelay2 (D − 1 , &q)
    cdelay2 (D − 1 , &q)
```

```
repeat forever :
    output y = w[q]
    cdelay2 (D − 1 , &q)
    cdelay2 (D − 1 , &q)
    cdelay2 (D − 1 , &q)
    cdelay2 (D − 1 , &q)
```

# Sinusoidal Generators

```
repeat forever :
    output y = w[q]
    gdelay2 (D − 1 , c, &q)
```

```
repeat forever :
    i = q
    output y = w[i]
    gdelay2 (D − 1 , c, &q)
```

```
repeat forever :
    i = q
    output y = Aw[i]
    gdelay2 (D − 1 , DF, & q)
```

# Sinusoidal Generators

```
for each input sample x do:
        y = x + awD
        w0 = x
        delay (D, w)
```

```
for each input sample x do:
        sD = tap (D, w , p, D)
        y = x + asD
        ∗p = x
        cdelay (D, w , &p)
```

# Sinusoidal Generators

for each time instant n and input sample x do:

compute current notch $\omega_0 = \omega_1 + \omega_2 \sin(\omega_{sweep} n)$

$w_0 = bx + 2b \cos\omega_0 \, w_1 - (2b - 1) w_2$

$y = w_0 - 2 \cos\omega_0 \, w_1 + w_2$

$w_2 = w_1$

$w_1 = w_0$

# Sinusoidal Generators

```
for each input sample x do:
    w0 = x + awD
    y = −aw0 + wD
    delay (D, w)
```

```
for each input sample x do:
    sD = tap (D, w , p, D)
    s0 = x + asD
    y = −as0 + sD
    ∗p = s0
    cdelay (D, w , &p)
```

# Sinusoidal Generators

```
for each input sample x do:
    x1 = plain (D1 , w1 , &p1 , a1 , x)
    x2 = plain (D2 , w2 , &p2 , a2 , x)
    x3 = plain (D3 , w3 , &p3 , a3 , x)
    x4 = plain (D4 , w4 , &p4 , a4 , x)
    x5 = b1 x1 + b2 x2 + b3 x3 + b4 x4
    x6 = allpass (D5 , w5 , &p5 , a5 , x5)
    y = allpass (D6 , w6 , &p6 , a6 , x6)
```

# Sinusoidal Generators S

```
for each input sample x do:
       u = can (M, a, M, b , v, wD )
       y = x + u
       w0 = y
       delay (D, w)

for each input sample x do:
       s D = tap (D, w , p, D)
       u = can (M, a , M, b , v , sD )
       y = x + u
       ∗p = y
       cdelay (D, w , &p)
```

# Sinusoidal Generators S

```
for each input sample x do:
        y = wD
        w0 = x + awD
        delay (D, w)

for each input sample x do:
        sD = tap (D, w , p, D)
        y = sD
        ∗p = x + asD
        cdelay (D, w , &p)
```

# Sinusoidal Generators S

```
for each input sample x do:
        y = b0 x + b1 w1 D + b2 w2 D
        u 2 = can (G2 , w2 D )
        w20 = w1 D + u2
        delay (D2 , w2)
        u 1 = can (G1 , w1 D)
        w10 = x + u1
        delay (D1 , w1)
```

# Sinusoidal Generators S

```
for each input sample x do:
    s1 = tap (D1 + D2 , w , p, D1)
    s2 = tap (D1 + D2 , w , p, D1 + D2)
    y = b0 x + b1 s1 + b2 s2
    s0 = x + a1 s1 + a2 s2
    ∗p = s0
    cdelay (D1 + D2 , w , &p)
```

# Sinusoidal Generators S

```
for each input sample x do:
    c = λc 1 + ( 1 − λ)|x|
    c1 = c
    g = fir(M, h , w , f (c))
    y = Gx
```

# Sinusoidal Generators S

```
for each input sample x do:
        y = aw1 + bx
        w1 = y
```

```
for each input sample x do:
        y = −a1 w1 − a2 w2 + Gx
        w2 = w1
        w1 = y
```

```
for each input sample x do:
        w0 = 0 . 9615 w10 + 0 . 9807 x
        y = w0 − w10
        delay ( 10 , w )
```

# Sinusoidal Generators S

```
for each input sample x do:
    w 0 = 0 . 9391 w 50 + 0 . 0305 x
    y = w 0 + w 50
    delay ( 50 , w )
```

```
for each input sample x do:
    s 50 = tap ( 50 , w , p, 50 )
    s 0 = 0 . 9391 s 50 + 0 . 0305 x
    y = s 0 + s 50
    ∗p = s 0
    cdelay ( 50 , w , & p)
```

# Sinusoidal Generators S

```
for each of the ND inputs x do:
    w0 = wD + x/N
    delay (D, w)

for each of the ND inputs x do:
    ∗p = tap (D, w , p, D)+x/N
    cdelay (D, w , &p)

for n = 0 , 1 , . . . , D − 1,
    ŷ(n)= wD−n

for n = 0 , 1 , . . . , D − 1,
    ŷ(n)= tap(D, w , p, D−n)
```

# Sinusoidal Generators S

```
double *b, *w, *p;                                  definition of b[n] is not shown
b = (double *) calloc(D, sizeof(double));           (D+ 1 ) -dimensional
w = (double *) calloc(D+1, sizeof(double));         initialize circular pointer
p = w;                                              initialization part
for (n=0; n<D; n++) {                               fill buffer with b[n] 's
        *p = b[n];                                  current output
        printf("%lf\n", *p);                        update circular delay line
        cdelay(D, w, &p);                           steady state part
}                                                   first state = last state
for (n=D; n<Ntot; n++) {                            current output
        *p = tap(D, w, p, D);                       update circular delay line
        printf("%lf\n", *p);
        cdelay(D, w, &p);
}
```

# Sinusoidal Generators S

```
for (n=0; n<D; n++) {              initialization part
      w[0] = b[n];                 fill buffer with b[n] ’s
      printf("%lf\n", w[0]);       current output
      delay(D, w);                 update linear delay line
}                                  steady state part
for (n=D; n<Ntot; n++) {           first state = last state
      w[0] = w[D];                 current output
      printf("%lf\n", w[0]);       update linear delay line
      delay(D, w);
}
```

# Sinusoidal Generators S

```
/* gdelay2.c - generalized circular delay with real-
valued shift */
void gdelay2(int D, double c, double *q)
{
        *q -= c;                // c =shift, q =offset index
                                // decrement by c
        If (*q < 0)
                *q += D+1;
        if (*q > D)
                *q -= D+1;
}
```

# Sinusoidal Generators S

```
/* wavgen.c - wavetable generator (truncation method) */
void gdelay2();
double wavgen(int D, double *w, double A, doulbe F, double *q)
{
        double y;
        int i;
        i = (int) (*q);
        truncate down
        y = A * w[i];
        shift c = DF
        gdelay2(D-1, D*F, q);
        return y;
}
```

# Sinusoidal Generators S

```
/* wavgenr.c - wavetable generator (rounding method) */
void gdelay2();
double wavgenr(int D, doulbe *w, A, F, doulbe *q)
{
        double y;
        int k;
        k = (int) (*q + 0.5);
        // usage: y = wavgenr(D, w, A, F, &q);
        // D = wavetable length
        // A = amplitude, F = frequency, q = offset index
        // round
        y = A * w[k];
        gdelay2(D-1, D*F, q);
        // shift c = DF
        return y;
}
```

# Sinusoidal Generators S

```
/* wavgeni.c - wavetable generator (interpolation method) */
void gdelay2();
double wavgeni(int D, double *w, A, F, double *q)
{
        double y;
        int i, j;
        i = (int) *q;
        j = (i + 1) % D;
        // usage: y = wavgeni(D, w, A, F, &q);
        // D = wavetable length
        // A = amplitude, F = frequency, q = offset index
        // interpolate between w[i], w[j]
        y = A * (w[i] + (*q - i) * (w[j] - w[i]));
        gdelay2(D-1, D*F, q);
        return y;
}
```

# Sinusoidal Generators S

```
/* sine.c - sine wavetable of length D */
#include <math.h>
double sine(int D, int i)
{
	double pi = 4 * atan(1.0);
	return sin(2 * pi * i / D);
}

/* square.c - square wavetable of length D, with D1 ones */
double square(int D1, int I)
{
	if (i < D1)
		return 1;
	else
		return 0;
}
```

# Sinusoidal Generators S

```
/* trapez.c - trapezoidal wavetable: D1 rising, D2 steady */
double trapez(int D, int D1, int D2, int i)
{
        if (i < D1)
                return i/(double) D1;
        else
                if (i < D1+D2)
                        return 1;
                else
                        return (D - i)/(double) (D - D1 - D2);
}
```

# Sinusoidal Generators S

```
double *w;
w = (double *) calloc(D, sizeof(double));        // use: D = 1000
q1 = q2 = q3 = q4 = q5 = 0;                       // initialize q s
for (i=0; i<D; i++) {
      w[q1] = sine(D, i);                 // load wavetable with a sinusoid
      gdelay2(D-1, 1.0, &q1);             // may need the cast w[(int)q1]
}
gdelay2(D-1, -m*D*F2, &q4);                    // reset q4 = mDF 2
gdelay2(D-1, m*D*F2, &q5);                     // reset q5 = −mDF 2
for (n=0; n<Ntot; n++) {                       // use: A = 1, N tot = 1000
      y1[n] = wavgen(D, w, A, F1, &q1);   // use: F 1 = 1.0 /D
      y2[n] = wavgen(D, w, A, F1, &q2);   // use: F 2 = 5.0 /D
      y3[n] = wavgen(D, w, A, F1, &q3);   // use: F 3 = 10.5 /D
      y4[n] = wavgen(D, w, A, F1, &q4);   // use: F 4 = F2
      y5[n] = wavgen(D, w, A, F1, &q5);   // use: F 5 = F2
}
```

# Sinusoidal Generators S

```
double *w, *wenv, q, qenv;
w = (double *) calloc(D, sizeof(double));         // use: D = 1000
wenv = (double *) calloc(D, sizeof(double));      // allocate wavetables

q = qenv = 0;                                     // initialize offsets

for (i=0; i<D; i++) {                             // load wavetables:
      w[q] = sine(D, i);                          // may need the cast w[(int)q]
      wenv[qenv] = trapez(D, D/4, 0, i);          // triangular envelope
      gdelay2(D-1, 1.0, &q);                      // or, cdelay2(D-1, &q);
      gdelay2(D-1, 1.0, &qenv);
}
Fenv = 1.0 / Ntot;                                // use: N tot = 1000 or 2000
                                                  // envelope frequency
for (n=0; n<Ntot; n++) {
      A[n] = wavgen(D, wenv, Aenv, Fenv, &qenv);  // use: A env = 1 . 0
      y[n] = wavgen(D, w, A[n], F, &q);           // use: F = 0 . 01
}
```

# Sinusoidal Generators S

```
q = qenv = 0;                                   // initialize offsets
for (i=0; i<D; i++) {                           // load wavetables
        w[q] = sine(D, i);                      // sinusoidal signal
        wenv[qenv] = 1 + 0.25 * sine(D, i);     // sinusoidal envelope
        gdelay2(D-1, 1.0, &q);                  // or cdely2(D-1, &q)
        gdelay2(D-1, 1.0, &qenv);
}
```

# Sinusoidal Generators S

```
double *w, *wm;
w = (double *) calloc(D, sizeof(double));
wm = (double *) calloc(D, sizeof(double));

q = qm = 0;

for (i=0; i<D; i++) {                              // load wavetables
    w[q] = sine(D, i);                             // signals: y 1 (n), y 2 (n), y 3 (n)
    /* w[q] = square(D/2, i); */                   // signal: y 4 (n)
    gdelay2(D-1, 1.0, &q);

    wm[qm] = sine(D, i);                           // signal: y 1 (n)
    /* wm[qm] = 2 * square(D/2, i) - 1; */         // signal: y 2 (n)
    /* wm[qm] = trapez(D, D, 0, i); */             // signals: y 3 (n), y 4 (n)
    gdelay2(D-1, 1.0, &qm);
}
```

# Sinusoidal Generators S

```
for (n=0; n<Ntot; n++) {                                    // use: N tot = 1000
    F[n] = Fc + wavgen(D, wm, Am, Fm, &qm);
    y[n] = wavgen(D, w, A, F[n], &q);                       // use: A = 1
}
```

**References**

[1] S. J. Ofranidis , Introduction to Signal Processing