```
#include <time.h>
#include "iirutil.hh"
#include "FilterDesign.hh"
#include "filterwiz/FilterFile.hh"
#include <math.h>
#include <string>
#include <iostream>
#include <fstream>
#include <iomanip>

using namespace ligogui;
using namespace filterwiz;
using namespace std;


const double PI = 3.14159265358979323846;
const double fS = 524288.;
const double dT = 17.0;
const bool writefiles = true;
const int kMaxSOS = 7;

typedef unsigned long coeff_list[4][128];
typedef unsigned long parity_list[4][32];

typedef char code_type[41];
typedef code_type code_list[128];

const code_list code = {
  // List of coefficients and micro code
  // bit encoded, LSB last in list, spaces ignored
  //
  //        history file address
  //        *     load input/output
  //        *     *history write enable
  //        *     * accumulator reset
  //        *     *  accumulator load
  //        *     *    multiplier shift: 00-no shift; 01-17b shift; 1X-34b shift
  //        *     *    * input selection: 00-input,lsb; 01-input,msb; 1X-history
  //        *     *    *    coefficients
  //        *     *    *    *                          iteration: value
  "00000 00000 0000 0000 00000000000000000", // code   0: g, msb
  "00000 00000 0010 0001 00000000000000000", // code   1: g, lsb
  "00000 10000 0000 0101 00000000000000000", // code   2: g, msb

  "00000 10000 0000 0110 00000000000000000", // code   3: b20, lsb
  "00000 10001 0000 1010 00000000000000000", // code   4: b20, msb
  "00000 10001 0000 0010 00000000000000000", // code   5: b20, lsb
  "00000 00000 0100 0110 00000000000000000", // code   6: b20, msb
  "00000 00000 0000 0110 00000000000000000", // code   7: b10, lsb
  "00000 00001 0000 1010 00000000000000000", // code   8: b10, msb
  "00000 00001 0000 0010 00000000000000000", // code   9: b10, lsb
  "00000 00000 0000 0110 00000000000000000", // code  10: b10, msb

  "00000 10010 0000 0110 00000000000000000", // code  11: -lb(c00)
```

```
  "00000 10010 0000 1010 00000000000000000", // code  12: a20, lsb
  "00000 10011 0000 0010 00000000000000000", // code  13: a20, msb
  "00000 10011 0001 0010 00000000000000000", // code  14: a20, lsb
  "00000 00010 0000 0110 00000000000000000", // code  15: a20, msb
  "00000 00010 0000 0110 00000000000000000", // code  16: a10, lsb
  "00000 00011 0000 1010 00000000000000000", // code  17: a10, msb
  "00000 00011 0000 0010 00000000000000000", // code  18: a10, lsb
  "00000 10010 0000 0110 00000000000000000", // code  19: a10, msb

  "00000 10010 0000 0110 00000000000000000", // code  20: b21, lsb
  "00000 10011 0000 1010 00000000000000000", // code  21: b21, msb
  "00000 10011 0000 0010 00000000000000000", // code  22: b21, lsb
  "00000 00010 0100 0110 00000000000000000", // code  23: b21, msb
  "00000 00010 0000 0110 00000000000000000", // code  24: b11, lsb
  "00000 00011 0000 1010 00000000000000000", // code  25: b11, msb
  "00000 00011 0000 0010 00000000000000000", // code  26: b11, lsb
  "00000 00000 0000 0110 00000000000000000", // code  27: b11, msb

  "00000 10100 0000 0110 00000000000000000", // code  28: -lb(c01)
  "00000 10100 0000 1010 00000000000000000", // code  29: a21, lsb
  "00000 10101 0000 0010 00000000000000000", // code  30: a21, msb
  "00000 10101 0001 0010 00000000000000000", // code  31: a21, lsb
  "00000 00100 0000 0110 00000000000000000", // code  32: a21, msb
  "00000 00100 0000 0110 00000000000000000", // code  33: a11, lsb
  "00000 00101 0000 1010 00000000000000000", // code  34: a11, msb
  "00000 00101 0000 0010 00000000000000000", // code  35: a11, lsb
  "00000 10100 0000 0110 00000000000000000", // code  36: a11, msb

  "00000 10100 0000 0110 00000000000000000", // code  37: b22, lsb
  "00000 10101 0000 1010 00000000000000000", // code  38: b22, msb
  "00000 10101 0000 0010 00000000000000000", // code  39: b22, lsb
  "00000 00100 0100 0110 00000000000000000", // code  40: b22, msb
  "00000 00100 0000 0110 00000000000000000", // code  41: b12, lsb
  "00000 00101 0000 1010 00000000000000000", // code  42: b12, msb
  "00000 00101 0000 0010 00000000000000000", // code  43: b12, lsb
  "00000 00000 0000 0110 00000000000000000", // code  44: b12, msb

  "00000 10110 0000 0110 00000000000000000", // code  45: -lb(c02)
  "00000 10110 0000 1010 00000000000000000", // code  46: a22, lsb
  "00000 10111 0000 0010 00000000000000000", // code  47: a22, msb
  "00000 10111 0001 0010 00000000000000000", // code  48: a22, lsb
  "00000 00110 0000 0110 00000000000000000", // code  49: a22, msb
  "00000 00110 0000 0110 00000000000000000", // code  50: a12, lsb
  "00000 00111 0000 1010 00000000000000000", // code  51: a12, msb
  "00000 00111 0000 0010 00000000000000000", // code  52: a12, lsb
  "00000 10110 0000 0110 00000000000000000", // code  53: a12, msb

  "00000 10110 0000 0110 00000000000000000", // code  54: b23, lsb
  "00000 10111 0000 1010 00000000000000000", // code  55: b23, msb
  "00000 10111 0000 0010 00000000000000000", // code  56: b23, lsb
  "00000 00110 0100 0110 00000000000000000", // code  57: b23, msb
  "00000 00110 0000 0110 00000000000000000", // code  58: b13, lsb
  "00000 00111 0000 1010 00000000000000000", // code  59: b13, msb
  "00000 00111 0000 0010 00000000000000000", // code  60: b13, lsb
```

```
   "00000 00000 0000 0110 00000000000000000", // code  61: b13, msb

   "00000 11000 0000 0110 00000000000000000", // code  62: -lb(c03)
   "00000 11000 0000 1010 00000000000000000", // code  63: a23, lsb
   "00000 11001 0000 0010 00000000000000000", // code  64: a23, msb
   "00000 11001 0001 0010 00000000000000000", // code  65: a23, lsb
   "00000 01000 0000 0110 00000000000000000", // code  66: a23, msb
   "00000 01000 0000 0110 00000000000000000", // code  67: a13, lsb
   "00000 01001 0000 1010 00000000000000000", // code  68: a13, msb
   "00000 01001 0000 0010 00000000000000000", // code  69: a13, lsb
   "00000 11000 0000 0110 00000000000000000", // code  70: a13, msb

   "00000 11000 0000 0110 00000000000000000", // code  71: b24, lsb
   "00000 11001 0000 1010 00000000000000000", // code  72: b24, msb
   "00000 11001 0000 0010 00000000000000000", // code  73: b24, lsb
   "00000 01000 0100 0110 00000000000000000", // code  74: b24, msb
   "00000 01000 0000 0110 00000000000000000", // code  75: b14, lsb
   "00000 01001 0000 1010 00000000000000000", // code  76: b14, msb
   "00000 01001 0000 0010 00000000000000000", // code  77: b14, lsb
   "00000 00000 0000 0110 00000000000000000", // code  78: b14, msb

   "00000 11010 0000 0110 00000000000000000", // code  79: -lb(c04)
   "00000 11010 0000 1010 00000000000000000", // code  80: a24, lsb
   "00000 11011 0000 0010 00000000000000000", // code  81: a24, msb
   "00000 11011 0001 0010 00000000000000000", // code  82: a24, lsb
   "00000 01010 0000 0110 00000000000000000", // code  83: a24, msb
   "00000 01010 0000 0110 00000000000000000", // code  84: a14, lsb
   "00000 01011 0000 1010 00000000000000000", // code  85: a14, msb
   "00000 01011 0000 0010 00000000000000000", // code  86: a14, lsb
   "00000 11010 0000 0110 00000000000000000", // code  87: a14, msb

   "00000 11010 0000 0110 00000000000000000", // code  88: b25, lsb
   "00000 11011 0000 1010 00000000000000000", // code  89: b25, msb
   "00000 11011 0000 0010 00000000000000000", // code  90: b25, lsb
   "00000 01010 0100 0110 00000000000000000", // code  91: b25, msb
   "00000 01010 0000 0110 00000000000000000", // code  92: b15, lsb
   "00000 01011 0000 1010 00000000000000000", // code  93: b15, msb
   "00000 01011 0000 0010 00000000000000000", // code  94: b15, lsb
   "00000 00000 0000 0110 00000000000000000", // code  95: b15, msb

   "00000 11100 0000 0110 00000000000000000", // code  96: -lb(c05)
   "00000 11100 0000 1010 00000000000000000", // code  97: a25, lsb
   "00000 11101 0000 0010 00000000000000000", // code  98: a25, msb
   "00000 11101 0001 0010 00000000000000000", // code  99: a25, lsb
   "00000 01100 0000 0110 00000000000000000", // code 100: a25, msb
   "00000 01100 0000 0110 00000000000000000", // code 101: a15, lsb
   "00000 01101 0000 1010 00000000000000000", // code 102: a15, msb
   "00000 01101 0000 0010 00000000000000000", // code 103: a15, lsb
   "00000 11100 0000 0110 00000000000000000", // code 104: a15, msb

   "00000 11100 0000 0110 00000000000000000", // code 105: b26, lsb
   "00000 11101 0000 1010 00000000000000000", // code 106: b26, msb
   "00000 11101 0000 0010 00000000000000000", // code 107: b26, lsb
   "00000 01100 0100 0110 00000000000000000", // code 108: b26, msb
```

```
  "00000 01100 0000 0110 0000000000000000", // code 109: b16, lsb
  "00000 01101 0000 1010 0000000000000000", // code 110: b16, msb
  "00000 01101 0000 0010 0000000000000000", // code 111: b16, lsb
  "00000 00000 0000 0110 0000000000000000", // code 112: b16, msb

  "00000 11110 0000 0110 0000000000000000", // code 113: -lb(c06)
  "00000 11110 0000 1010 0000000000000000", // code 114: a26, lsb
  "00000 11111 0000 0010 0000000000000000", // code 115: a26, msb
  "00000 11111 0001 0010 0000000000000000", // code 116: a26, lsb
  "00000 01110 0000 0110 0000000000000000", // code 117: a26, msb
  "00000 01110 0000 0110 0000000000000000", // code 118: a16, lsb
  "00000 01111 0000 1010 0000000000000000", // code 119: a16, msb
  "00000 01111 0000 0010 0000000000000000", // code 120: a16, lsb
  "00000 11110 0000 0110 0000000000000000", // code 121: a16, msb

  "00000 11110 0000 0110 0000000000000000", // code 122: --
  "00000 11111 0000 1010 0000000000000000", // code 123: --
  "00000 11111 0000 0010 0000000000000000", // code 124: --
  "00000 01110 1100 0000 0000000000000000", // code 125: --
  "00000 01110 0000 0000 0000000000000000", // code 126: --
  "00000 00000 0000 0000 0000000000000000"  // code 127: g,lsb
  };


unsigned long convert_coeff (double a, const char* p = "lsb")
{
  const int q = 1.0;
  unsigned long long c = 0;
  bool lsb = strcmp (p, "lsb") == 0;

  double aa = fabs (a);
// too large?
  if (aa > 2.0*q) {
    c = 0x3FFFFFFFULL;
  }
  else {
  // convert bit by bit starting at the MSB
    aa += exp (-log(2.0)*34); // for correct rounding
  //int n = lsb ? 34 : 17;
    int n = 34;
    for (int i = n-1; i >= 0; --i) {
      if (aa >= q) {
        c |= 1ULL << i;
        aa -= q;
      }
      aa *= 2.0;
    }
  }
// form negative if necessary
  if (a < 0.0) {
    c = -c;
  }
```

```
  // mask higher order bits
  if (lsb) {
    c &= 0x1FFFFULL; // get last 17 bits
  }
  else {
    unsigned long long mask = 0x3FFFFULL;
    c = (c & (mask << 17)) >> 17; // get bits 18 thru 35
  }
  return c;
}

unsigned long convert_shiftadjust (double g)
{
  double gain = fabs (g);
  int n = (1 << 3) - 1;
  for (int i = -(1 << 3) + 1; i <= 0; ++i) {
    if (gain > exp (log(2.0) * i)) {
      n = -i;
    }
    else {
      break;
    }
  }
  return n & 0xF;
}

unsigned long convert_code (const char* code)
{
  unsigned long c = 0;
  for (const char* p = code; *p; ++p) {
    if (!isspace (*p)) {
      c <<= 1;
      c |= (*p == '1') ? 1 : 0;
    }
  }
  return c & 0xFFFC0000;
}


int main(int argc, char **argv)
{

  if (argc <= 1) {
    cout << "Usage: coeffgen 'filterfile' {'filter'}" << endl;
    return 1;
  }
  FilterFile ff;
  const FilterModule* mod = 0;
```

```
    if (!ff.read (argv[1])) {
      cerr << "Unable to open filter file " << argv[1] << endl;
      return 1;
    }

    coeff_list coeffs;
    memset (coeffs, 0, sizeof (coeff_list));
    parity_list parity;
    memset (parity, 0, sizeof (parity_list));

    for (int fil = 0; fil < 4; ++fil) {
    // get coefficients
      if ((argc > fil + 2) && (mod = ff.find (argv[fil+2]))) {
        double fS = mod->getFSample();
        if (!(*mod)[0].valid()) {
          cerr << "Filter not valid at section 0 of " <<
              argv[fil+2] << endl;
        }
        else {
          cout << "Use filter " << (*mod)[0].getName() << " of " <<
              argv[fil+2] << endl;
          bool err = false;
          const FilterDesign& ds = (*mod)[0].filter();

          int order = iirorder (ds.get());
          if (order < 0) {
            cerr << "Not an IIR filter" << endl;
            err = true;
            order = 0;
          }
          int nba = 1;
          double* coeff = new double [1 + 4*order];
          if (!err) {
            if (!iir2z (ds.get(), nba, coeff, "o")) {
              cerr << "Unable to obtain online filter coefficients" <<
                  endl;
              err = true;
            }
          }
          int sosnum = (nba - 1) / 4;
          if (!err) {
            if ((sosnum < 1) || (sosnum > kMaxSOS)) {
              cerr << "Invalid number of SOSs " << sosnum << endl;
              err = true;
            }
          }
          if (!err) {
            cout << "Filter " << fil << ": Creating " <<
                (*mod)[0].getName() << " of " << argv[fil+2] << endl;
            for (int j = 0; j < sosnum; ++j) {
              coeffs[fil][3+j*17] = coeffs[fil][5+j*17] =
                  convert_coeff ( coeff[4+j*4], "lsb");
              coeffs[fil][4+j*17] = coeffs[fil][6+j*17] =
```

```
                convert_coeff ( coeff[4+j*4], "msb");
            coeffs[fil][7+j*17] = coeffs[fil][9+j*17] =
                convert_coeff ( coeff[3+j*4], "lsb");
            coeffs[fil][8+j*17] = coeffs[fil][10+j*17]=
                convert_coeff ( coeff[3+j*4], "msb");
            coeffs[fil][12+j*17]= coeffs[fil][14+j*17]=
                convert_coeff (-coeff[2+j*4], "lsb");
            coeffs[fil][13+j*17]= coeffs[fil][15+j*17]=
                convert_coeff (-coeff[2+j*4], "msb");
            coeffs[fil][16+j*17]= coeffs[fil][18+j*17]=
                convert_coeff (-coeff[1+j*4], "lsb");
            coeffs[fil][17+j*17]= coeffs[fil][19+j*17]=
                convert_coeff (-coeff[1+j*4], "msb");
          }
          double gain = 1.0;
          double gainerr = 1.0;
          for (int j = sosnum - 1; j >= 0; --j) {
            double dcgain = (1.0+coeff[3+j*4]+coeff[4+j*4])/
                (1.0+coeff[1+j*4]+coeff[2+j*4]);
            coeffs[fil][11+j*17]= convert_shiftadjust (gainerr/dcgain);
            double shiftgain = exp (log(2.0) * coeffs[fil][11+j*17]);
            gain = gain / shiftgain ;
            gainerr = gainerr / dcgain * shiftgain;
            cout <<
                "  Section " << j << ": dc gain is " << 1.0/dcgain <<
                "  shift adj. is <<" << coeffs[fil][11+j*17] <<
                "  cumm. gain err is " << gainerr << endl;
          }
          gain = coeff[0] / gain;
          cout << "  Remaining filter gain is " << setprecision(12) <<
              gain << setprecision(6) << endl;
          if (fabs (gain) >= 2.0) {
            cerr << "Gain out of range " << gain <<
                " expected <2.0   TRUNCATED!" << endl;
          }
          coeffs[fil][127] = coeffs[fil][1] = convert_coeff (gain, "lsb");
          coeffs[fil][0]   = coeffs[fil][2] = convert_coeff (gain, "msb");
        }
        delete [] coeff;
      }
    }
    // fill in micro code
    for (int j = 0; j < 128; ++j) {
      coeffs[fil][j] |= convert_code (code[j]);
      //printf ("0x%08lx  ", coeffs[fil][j]);
      //if (j % 4 == 3) printf ("\n");
    }
  }
  cout << endl << endl;

// write memory initialization file (generic)
  for (int i = 0; i < 4*16; ++i) {
    cout << "        INIT_" << setfill('0') << setw(2) << right << hex << i;
    cout << " : bit_vector(255 downto 0) :=" << endl;
```

```
    cout << "           X\"";
    for (int j = 7; j >= 0; --j) {
      cout << setfill('0') << setw(8) << hex <<
          coeffs[i / 16][8*(i % 16)+j];
    }
    cout << "\";" << endl;
  }
  cout << endl << endl;

  // write generic map
  //   for (int i = 0; i < 4*16; ++i) {
  //      cout << "       INIT_" << setfill('0') << setw(2) << right << hex << i;
  //      cout << " => X\"";
  //      for (int j = 7; j >= 0; --j) {
  //         cout << setfill('0') << setw(8) << hex <<
  //            coeffs[i / 16][8*(i % 16)+j];
  //      }
  //      cout << "\"," << endl;
  //   }
  //   cout << endl << endl;

  // write memory initialization file (attributes)
  for (int i = 0; i < 4*16; ++i) {
    cout << "   attribute INIT_" << setfill('0') << setw(2) << right <<
       hex << i;
    cout << " : string;" << endl;
    cout << "   attribute INIT_" << setfill('0') << setw(2) << right <<
       hex << i;
    cout << " of U_RAM : label is" << endl;
    cout << "      \"";
    for (int j = 7; j >= 0; --j) {
      cout << setfill('0') << setw(8) << hex <<
          coeffs[i / 16][8*(i % 16)+j];
    }
    cout << "\";" << endl;
  }

  return 0;
}
```