Application Note: AN00209

xCORE-200 DSP Elements Library

The application note gives an overview of using the xCORE-200 DSP Elements Library.

Required tools and libraries

- xTIMEcomposer Tools - Version 14.2.0 and above
- XMOS DSP library - Version 3.0.0 and above

Required hardware

This application note is designed to run on any XMOS xCORE-200 multicore microcontroller or the XMOS simulator.

Prerequisites

- This document assumes familiarity with the XMOS xCORE architecture, xCONNECT interconnect communication, the XMOS tool chain and the xC language. Documentation related to these aspects are linked to in the appendix §A.
- For descriptions of XMOS related terms found in this document please see the XMOS Glossary[1].

[1] http://www.xmos.com/published/glossary

1 Introduction

The XMOS xCORE-200 DSP Elements Library provides foundation Digital Signal Processing functions.

The library is split into the following modules :

- Adaptive Filters - dsp_adaptive.h
- Filtering - dsp_filters.h
- Basic Math - dsp_math.h
- Matrix Math - dsp_matrix.h
- Statistics - dsp_statistics.h
- Vectors - dsp_vector.h
- FFT and inverse FFT - dsp_fft.h
- Downsample by 3 - dsp_ds3.h
- Oversample by 3 - dsp_os3.h

The library supports variable Q formats from Q8 to Q31, using the macros in the following header file : dsp_qformat.h.

The Downsample and Oversample functions are fixed Q31 format.

2 Getting Started

Ensure you are in the xTIMEcomposer edit perspective by clicking on the Edit perspective button on the left hand side toolbar.

In the Project Explorer view you will see the following applications :

- Adaptive Filters - app_adaptive
- Filtering - app_filters
- Basic Math - app_math
- Matrix Math - app_matrix
- Statistics - app_statistics
- Vectors (Real and Complex) - app_vector
- FFT and inverse FFT - app_fft
- FFT Processing of signals received through a double buffer - app_fft_double_buf
- Downsample by 3 - app_ds3
- Oversample by 3 - app_os3

The applications contain code to generate the simulation data and call all of the functions in each module and print the results in the xTIMEcomposer console.

2.1 Build the applications

To build the application, select 'Project -> Build Project' in the menu, or click the 'Build' button on the toolbar. The output from the compilation process will be visible on the console. Note that some applications offer different configurations. Particular Build Configurations can be selected by clicking the small arrow next to the Build Icon (Hammer).

2.2 Create and launch a run configuration

To run or debug the application, you have to initially create a run or debug configuration. The xTIMEcomposer allows multiple configurations to exist, thus allowing you to store configurations for running on different targets/with different runtime options and arguments. Right-click on the generated binary in the *Project Explorer* view, and select *Run As -> Run Configurations*. To debug, select *Run As -> Debug Configurations*.

In the resulting dialog, double click on *xCORE Application*, then perform the following operations:

- On the *Main* tab select the desired Hardware Target, or check the *simulator* option.
- Select the Run button to launch the application.

The results will be displayed in the xTIMEcomposer *console* tab.

3 Using The DSP Library In Other Applications

3.1 Makefile Additions For These Examples

To start using the DSP Library, you need to add lib_dsp to your Makefile:

```
USED_MODULES = ... lib_dsp ...
```

3.2 Including The Library Into Your Source Code

In order to use any of these modules and the Q formats it is only necessary to include the following header file:

```
#include "dsp.h"
```

APPENDIX A - References

XMOS Tools User Guide

http://www.xmos.com/published/xtimecomposer-user-guide

XMOS xCORE Programming Guide

http://www.xmos.com/published/xmos-programming-guide

XMOS DSP Library

http://www.xmos.com/support/libraries/lib_dsp

XMOS Sample Rate Conversion Library

https://www.xmos.com/published/lib_src-userguide

xCORE-200: The XMOS XS2 Architecture (ISA)

https://www.xmos.com/published/xs2-isa-specification

APPENDIX B - Full Source Code Listings

This section includes the source code for all of the example programs.

B.1 Adaptive Filtering Functions

```
// Copyright (c) 2015-2016, XMOS Ltd, All rights reserved
// XMOS DSP Library - Adaptive Filtering Function Test Program
// Uses Q24 format

// Include files
#include <stdio.h>
#include <xs1.h>
#include <print.h>
#include <dsp.h>

// Define constants

#define Q_M               1
#define Q_N               31

#define FIR_FILTER_LENGTH     160

void print31( int32_t x ) {if(x >=0) printf("+%f ",F31(x)); else printf("%f ",F31(x));}


// Declare global variables and arrays
int32_t fir_coeffs[] = // 161 taps
{
  Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
  Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
  Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
  Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
  Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
  Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
  Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
  Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
  Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
  Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
  Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
  Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
  Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
  Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
  Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
  Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
  Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
  Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
  Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
  Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
  Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
  Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
  Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
  Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
  Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
  Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
  Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
  Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
  Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
  Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
  Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
  Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
  Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
  Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
  Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
  Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
  Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
  Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
  Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
  Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
};


int32_t fir_state[FIR_FILTER_LENGTH];
```

```
int32_t lms_coeffs[FIR_FILTER_LENGTH];
int32_t nlms_coeffs[FIR_FILTER_LENGTH];

int main(void)
{
  int32_t c;
  int32_t x;
  int32_t err;

              // Apply LMS Filter
  for( c = 5; c <= 5; c *= 2 )
  {
    printf( "LMS %u\n", c );
    for( int32_t i = 0; i < FIR_FILTER_LENGTH; ++i ) lms_coeffs[i] = fir_coeffs[i];
    for( int32_t i = 0; i < FIR_FILTER_LENGTH; ++i ) fir_state[i] = 0;
    for( int32_t i = 0; i < c+30; ++i )
    {
      x = dsp_adaptive_lms( Q31(0.08), Q31(0.10), &err, lms_coeffs, fir_state, c, Q31(0.01), Q_N );
      print31( x ); print31( err ); printf( "\n" );
    }
  }

              // Apply Normalized LMS Filter
  for( c = 5; c <= 5; c *= 2 )
  {
    printf( "\nNormalized LMS %u\n", c );
    for( int32_t i = 0; i < FIR_FILTER_LENGTH; ++i ) nlms_coeffs[i] = fir_coeffs[i];
    for( int32_t i = 0; i < FIR_FILTER_LENGTH; ++i ) fir_state[i] = 0;
    for( int32_t i = 0; i < c+30; ++i )
    {
      x = dsp_adaptive_nlms( Q31(0.08), Q31(0.10), &err, nlms_coeffs, fir_state, c, Q31(0.01), Q_N );
      print31( x ); print31( err ); printf( "\n" );
    }
  }

  return (0);
}
```

B.2 Fixed Coefficient Filtering Functions

```
// Copyright (c) 2015-2016, XMOS Ltd, All rights reserved
// XMOS DSP Library - Filtering Functions Test Program
// Uses Q24 format

// Include files
#include <stdio.h>
#include <xs1.h>
#include <dsp.h>

// Define constants

#define Q_M               8
#define Q_N               24

#define FIR_FILTER_LENGTH     30
#define IIR_STATE_LENGTH      4
#define IIR_CASCADE_DEPTH     3
#define SAMPLE_LENGTH         50

#define INTERP_FILTER_LENGTH  160

void print31( int32_t x ) {if(x >=0) printf("+%f ",F31(x)); else printf("%f ",F31(x));}

// Declare global variables and arrays
int32_t  Src[] = { Q24(.11), Q24(.12), Q24(.13), Q24(.14), Q24(.15), Q24(.16), Q24(.17), Q24(.18), Q24(.19),
   ↪ Q24(.20),
               Q24(.21), Q24(.22), Q24(.23), Q24(.24), Q24(.25), Q24(.26), Q24(.27), Q24(.28), Q24(.29), Q24
                  ↪ (.30),
               Q24(.31), Q24(.32), Q24(.33), Q24(.34), Q24(.35), Q24(.36), Q24(.37), Q24(.38), Q24(.39), Q24
                  ↪ (.40),
               Q24(.41), Q24(.42), Q24(.43), Q24(.44), Q24(.45), Q24(.46), Q24(.47), Q24(.48), Q24(.49), Q24
                  ↪ (.50),
               Q24(.51), Q24(.52), Q24(.53), Q24(.54), Q24(.55), Q24(.56), Q24(.57), Q24(.58), Q24(.59), Q24
                  ↪ (.60)};

int32_t        Dst[INTERP_FILTER_LENGTH];

int32_t firCoeffs[] = { Q24(.11), Q24(.12), Q24(.13), Q24(.14), Q24(.15), Q24(.16), Q24(.17), Q24(.18), Q24
   ↪ (.19), Q24(.20),
               Q24(.21), Q24(.22), Q24(.23), Q24(.24), Q24(.25), Q24(.26), Q24(.27), Q24(.28), Q24(.29), Q24
                  ↪ (.30),
               Q24(.31), Q24(.32), Q24(.33), Q24(.34), Q24(.35), Q24(.36), Q24(.37), Q24(.38), Q24(.39), Q24
                  ↪ (.40)};

int32_t firCoeffsInt[] =
{
    Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
    Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
    Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
    Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
    Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
    Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
    Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
    Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
    Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
    Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
    Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
    Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
    Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
    Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
    Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
    Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
    Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
    Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
    Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
    Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
    Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
    Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
    Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
    Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
    Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
    Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
    Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
```

```
    Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
    Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
    Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
    Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
    Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
    Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
    Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
    Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
    Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
    Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
    Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
    Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
    Q31(+0.0391607),Q31(+0.0783215),Q31(+0.0191607),Q31(+0.01531791),Q31(-0.03098222),
};

int32_t iirCoeffs[] = { Q24(.11), Q24(.12), Q24(.13), Q24(.14), Q24(.15),
                    Q24(.21), Q24(.22), Q24(.23), Q24(.24), Q24(.25),
                    Q24(.31), Q24(.32), Q24(.33), Q24(.34), Q24(.35)};

//int filterState[FIR_FILTER_LENGTH];
int32_t filterState[INTERP_FILTER_LENGTH];

int32_t inter_coeff[INTERP_FILTER_LENGTH];
int32_t decim_coeff[INTERP_FILTER_LENGTH];
int32_t decim_input[16];

int overhead_time;

int main(void)
{
  int32_t i, j, c, r, x, y;
  const int print_cycles = 0;

  int32_t start_time, end_time, overhead_time, cycles_taken;
  timer tmr;
  tmr :> start_time;
  tmr :> end_time;
  overhead_time = end_time - start_time;
               // Initiaize FIR filter state array
  for (i = 0; i < FIR_FILTER_LENGTH; i++)
  {
    filterState[i] = 0;
  }

  // Apply FIR filter and store filtered data
  for (i = 0; i < SAMPLE_LENGTH; i++)
  {
    TIME_FUNCTION(Dst[i] =
      dsp_filters_fir (Src[i],                 // Input data sample to be filtered
                       firCoeffs,              // Pointer to filter coefficients
                       filterState,            // Pointer to filter state array
                       FIR_FILTER_LENGTH,      // Filter length
                       Q_N);                   // Q Format N
    );
  }

  if(print_cycles) {
    printf("cycles taken for executing dsp_filters_fir of length %d: %d\n", FIR_FILTER_LENGTH, cycles_taken);
  }


  printf ("FIR Filter Results\n");
  for (i = 0; i < SAMPLE_LENGTH; i++)
  {
      printf ("Dst[%d] = %lf\n", i, F24 (Dst[i]));
  }

               // Initiaize IIR filter state array
  for (i = 0; i < IIR_STATE_LENGTH; i++)
  {
    filterState[i] = 0;
  }

  // Apply IIR filter and store filtered data
  for (i = 0; i < SAMPLE_LENGTH; i++)
```

```
{
  TIME_FUNCTION(Dst[i] =
    dsp_filters_biquad (Src[i],              // Input data sample to be filtered
                        iirCoeffs,           // Pointer to filter coefficients
                        filterState,         // Pointer to filter state array
                        Q_N);                // Q Format N
  );
}

if(print_cycles) {
  printf("cycles taken for executing dsp_filters_biquad: %d\n", cycles_taken);
}

printf ("\nIIR Biquad Filter Results\n");
for (i = 0; i < SAMPLE_LENGTH; i++)
{
    printf ("Dst[%d] = %lf\n", i, F24 (Dst[i]));
}

// Initiaize IIR filter state array
for (i = 0; i < (IIR_CASCADE_DEPTH * IIR_STATE_LENGTH); i++)
{
  filterState[i] = 0;
}

// Apply IIR filter and store filtered data
for (i = 0; i < SAMPLE_LENGTH; i++)
{
  TIME_FUNCTION(Dst[i] =
    dsp_filters_biquads (Src[i],             // Input data sample to be filtered
                         iirCoeffs,          // Pointer to filter coefficients
                         filterState,        // Pointer to filter state array
                         IIR_CASCADE_DEPTH,  // Number of cascaded sections
                         Q_N);               // Q Format N
  );
}

if(print_cycles) {
  printf("cycles taken for executing dsp_filters_biquads (%d cascaded Biquads): %d\n", IIR_CASCADE_DEPTH,
    ↪ cycles_taken);
}

printf ("\nCascaded IIR Biquad Filter Results\n");
for (i = 0; i < SAMPLE_LENGTH; i++)
{
    printf ("Dst[%d] = %lf\n", i, F24 (Dst[i]));
}



printf ("\nInterpolation\n");
for( r = 2; r <= 8; ++r )
{
  c = 8;
  printf( "INTERP taps=%u L=%u\n", c*r, r );

  i = 0;
  for( y = 0; y < c; ++y )
    for( x = 0; x < r; ++x )
      inter_coeff[x*c+y] = firCoeffsInt[i++];

  for( i = 0; i < 160; ++i )
    filterState[i] = 0;

  for( i = 0; i < c; ++i )
  {
    dsp_filters_interpolate( Q31(0.1), inter_coeff, filterState, c*r, r, Dst, 31 );
    for( j = 0; j < r; ++j )
      print31( Dst[j] );
    printf( "\n" );
  }
}


printf ("\nDecimation\n");
```

```
  for( int32_t r = 2; r <= 8; ++r )
  {
    for( i = 0; i < 16; ++i )
      decim_input[i] = Q31(0.1);
    printf( "DECIM taps=%02u M=%02u\n", 32, r );
    for( i = 0; i < 160; ++i )
      filterState[i] = 0;
    for( i = 0; i < 32/r; ++i )
    {
      x = dsp_filters_decimate( decim_input, firCoeffsInt, filterState, 32, r, 31 );
      print31( x );
      if( (i&7) == 7 )
        printf( "\n" );
    }
    if( (--i&7) != 7 )
      printf( "\n" );
  }

  return (0);
}
```

B.3 Math Functions

```
// Copyright (c) 2015-2016, XMOS Ltd, All rights reserved
// XMOS DSP Library - Math Functions Test Program
// Uses Q24 format

#include <stdio.h>
#include <print.h>
#include <xs1.h>
#include <dsp.h>
#include <math.h>
#include <stdint.h>
#include <stdlib.h>

#define CHECK_RESULTS 1
#define PRINT_ERROR_TOO_BIG 1
#define EXIT_ON_ERROR_TOO_BIG 0
#define TEST_ALL_INPUTS 0

#define PRINT_CYCLE_COUNT 0

#if PRINT_CYCLE_COUNT
#define DIVIDE_STRESS 1 // execute divide on other cores to get worst case performance.
#else
#define DIVIDE_STRESS 0
#endif

#define STEPS 10

// errors from -15..+15
#define ERROR_RANGE 31
typedef struct {
    int32_t errors[ERROR_RANGE];
    int32_t smaller_than_min_error;
    int32_t greater_than_max_error;
    int32_t max_positive_error;
    int32_t max_negative_error;
    int32_t num_checked;
} error_s;

typedef enum {
  PV_OFF=0,
  PV_ON=1
} print_values_t;

typedef enum {
  PC_OFF=0,
  PC_ON=1
} print_cycles_t;

typedef enum {
  LINEAR=0,
  EXPONENTIAL=1
} stimulus_t;

/*
 * Report Errors within the Error Range. Top and Bottom or range contain saturated values
 */
int32_t report_errors(int32_t min_error, int32_t max_error, error_s *e) {
    int32_t result = 1; // PASS
    int32_t half_range = ERROR_RANGE/2;
    for(int32_t i=0; i<ERROR_RANGE; i++) {
        int32_t error = -half_range+i;
        unsigned num_errors = e->errors[i];

        if(num_errors) {
            if (error > max_error || error < min_error) {
              printf("FAIL:");
            }
            printf("Cases Error == %d: %d, percentage: %5.2f%\n",error, num_errors, 100.0*num_errors/e->
              ↪ num_checked);
        }
    }
    if (e->smaller_than_min_error ) {
        printf("Errors smaller than min_error %d: %d\n",min_error,e->smaller_than_min_error);
```

```
        result = 0;
    }
    if (e->greater_than_max_error ) {
        printf("Errors smaller than min_error %d: %d\n",max_error,e->greater_than_max_error);
        result = 0;
    }
    printf("Maximum Positive Error: %d\n",e->max_positive_error);
    printf("Maximum Negative Error: %d\n",e->max_negative_error);
    printf("Number of values checked: %d\n", e->num_checked);

    return result;
}
void reset_errors(error_s *e) {
    for(int32_t i=0; i<ERROR_RANGE; i++) {
        e->errors[i] = 0;
    }
    e->smaller_than_min_error = 0;
    e->greater_than_max_error = 0;
    e->max_positive_error = 0;
    e->max_negative_error = 0;
    e->num_checked = 0;
}

/* Function to check a result against a expected result and store error statistics
 *
 * \param[in]      result
 * \param[in]      expected result
 * \param[in]      max absolute error
 * \param[in,out]  pointer to error struct object
 * \returns        true if check passed
*/

int32_t check_result(int32_t result, int32_t expected, int32_t min_error, int32_t max_error, error_s *e) {
    static int32_t half_range = ERROR_RANGE/2;
    uint32_t error_found=0;

    int32_t error = result - expected;

    // Save max positive and negative error
    if (error < e->max_negative_error) e->max_negative_error = error;
    if (error > e->max_positive_error) e->max_positive_error = error;

    if (error > max_error) {
        e->greater_than_max_error++;
        printf("ERROR: error %d is greater than max_error %d\n",error, max_error);
        error_found = 1;
    } else if(error < min_error) {
        e->smaller_than_min_error++;
        printf("ERROR: error %d is smaller than min_error %d\n",error, min_error);
        error_found = 1;
    }

    if (error_found) {
    //if (error > max_error) {
#if PRINT_ERROR_TOO_BIG
        printf("result is 0x%x, Expected is 0x%x\n",result, expected);
        printf("\n");
#if EXIT_ON_ERROR_TOO_BIG
        report_errors(min_error, max_error, e);
        exit (0);
#endif
#endif
    }

    int bin_idx = error+half_range;
    if(bin_idx>=0 && bin_idx<ERROR_RANGE) {
       e->errors[bin_idx]++; // increment the error counter
    }

    e->num_checked++;

    return error_found;
}

int overhead_time;
```

```
//Todo: Test if this performs as well as the conversion macros in dsp_qformat.h
inline int32_t qs(double d, const int32_t q_format) {
  return (int32_t)((int64_t)((d) * ((uint64_t)1 << (q_format+20)) + (1<<19)) >> 20);
}

void test_multipliation_and_division() {
    const int32_t q_format = 24; // location of the decimal point. Gives 8 digits of precision after
      ↪ conversion to floating point.
    q8_24 result, expected;
    error_s err;
    reset_errors(&err);

    printf("Test Multiplication and Division\n");
    printf("--------------------------------\n");
    printf("Note: All calculations are done in Q8.24 format. That gives 7 digits of precision after the
      ↪ decimal point\n");
    printf("Note: Maximum double representation of Q8.24 format: %.8f\n\n", F24(0x7FFFFFFF));

    double f0, f1;
    f0 = 11.3137085;
    f1 = 11.3137085;
    // Multiply the square root of 128 (maximum double representation of Q8.24)
    printf ("Multiplication (%.8f x %.8f): %.8f\n\n",f0, f1, F24(dsp_math_multiply(Q24(f0), Q24(f1), q_format)
      ↪ ));

    printf ("Multiplication (11.4 x 11.4). Will overflow!: %.8f\n\n", F24(dsp_math_multiply(Q24(11.4), Q24
      ↪ (11.4), q_format)));;

    printf ("Saturated Multiplication (11.4 x 11.4): %.8f\n\n", F24(dsp_math_multiply_sat(Q24(11.4), Q24(11.4)
      ↪ , q_format)));;

    /*
    The result of 0.0005 x 0.0005 is 0.00000025. But this number is not representable as a binary.
    The closest representation in Q8.24 format is (4/2^24) = 0.000000238418579
    printf rounds this to 0.0000002 because the formatting string to printf specifies 8 digits of precision
      ↪ after the decimal point.
    This is the maximum precision that can be achieved with the 24 fractional bits of the Q8.24 format.
    */
    printf ("Multiplication of small numbers (0.0005 x 0.0005): %.8f\n\n", F24(dsp_math_multiply(Q24(0.0005),
      ↪ Q24(0.0005), q_format)));;


    double dividend, divisor;

    dividend = 1.123456; divisor = -128;
    result = dsp_math_divide(Q24(dividend), Q24(divisor), q_format);
    printf ("Signed Division %.8f / %.8f): %.8f\n\n",dividend, divisor, F24(result));
    expected = Q24(dividend/divisor);
    check_result(result, expected, -1, 1, &err);

    dividend = -1.123456; divisor = -128;
    result = dsp_math_divide(Q24(dividend), Q24(divisor), q_format);
    printf ("Signed Division %.8f / %.8f): %.8f\n\n",dividend, divisor, F24(result));
    expected = Q24(dividend/divisor);
    check_result(result, expected, -1, 1, &err);

    dividend = -1.123456; divisor = 127.9999999;
    result = dsp_math_divide(Q24(dividend), Q24(divisor), q_format);
    printf ("Signed Division %.8f / %.8f): %.8f\n\n",dividend, divisor, F24(result));
    expected = Q24(dividend/divisor);
    check_result(result, expected, -1, 1, &err);

    dividend = 1.123456; divisor = 127.9999999;
    result = dsp_math_divide(Q24(dividend), Q24(divisor), q_format);
    printf ("Signed Division %.8f / %.8f): %.8f\n\n",dividend, divisor, F24(result));
    expected = Q24(dividend/divisor);
    check_result(result, expected, -1, 1, &err);

    result = dsp_math_divide_unsigned(Q24(dividend), Q24(divisor), q_format);
    printf ("Division %.8f / %.8f): %.8f\n\n",dividend, divisor, F24(result));
    expected = Q24(dividend/divisor);
    check_result(result, expected, -1, 1, &err);

    printf("Error report from test_multipliation_and_division:\n");
```

```
    report_errors(-1, 1, &err);
    printf("\n");

}


q8_24 execute(int func, q8_24 x, unsigned &cycles_taken) {
    int32_t start_time, end_time;
    timer tmr;
    // uint32_t cycles_taken; // absolute positive values
    q8_24 result;
    // even func index executes the fixed point functions.
    switch(func) {
    case  0: {TIME_FUNCTION(result = dsp_math_exp(x)); return result;}
    case  1: {TIME_FUNCTION(result = Q24(exp(F24(x)))); return result;}
    case  2: {TIME_FUNCTION(result = dsp_math_log(x)); return result;}
    case  3: {TIME_FUNCTION(result = Q24(log(F24(x)))); return result;}
    case  4: {TIME_FUNCTION(result = dsp_math_sqrt(x)); return result;}
    case  5: {TIME_FUNCTION(result = Q24(sqrt(F24(x)))); return result;}
    case  6: {TIME_FUNCTION(result = dsp_math_sin(x)); return result;}
    case  7: {TIME_FUNCTION(result = Q24(sin(F24(x)))); return result;}
    case  8: {TIME_FUNCTION(result = dsp_math_cos(x)); return result;}
    case  9: {TIME_FUNCTION(result = Q24(cos(F24(x)))); return result;}
    case  10: {TIME_FUNCTION(result = dsp_math_atan(x)); return result;}
    case  11: {TIME_FUNCTION(result = Q24(atan(F24(x)))); return result;}
    case  12: {TIME_FUNCTION(result = dsp_math_sinh(x)); return result;}
    case  13: {TIME_FUNCTION(result = Q24(sinh(F24(x)))); return result;}
    case  14: {TIME_FUNCTION(result = dsp_math_cosh(x)); return result;}
    case  15: {TIME_FUNCTION(result = Q24(cosh(F24(x)))); return result;}

    }
    return INT32_MIN;
}

int test_input_range(int func, char name[], int min, int max, stimulus_t exponential, int minerror, int
  ↪ maxerror,
                     int permille, print_values_t print_values) {
    int fail = 0;
    int done=0;
    int done_after_next_iteration=0;
    int i=0;
    unsigned cycles_float, cycles_fixed;
    unsigned worst_cycles_fixed=0, worst_cycles_float=0;

    int32_t worst_cycles_input;

    error_s err;
    reset_errors(&err);

    int64_t x = (int64_t) min;  // use double precision to handle overflow

    printf("- Testing %s\n",name);

    while(!done) {
        timer t;
        int z, zc;

        zc = execute(func|1,x, cycles_float);
        if (zc == INT32_MIN) {
            printf("%s returned INT32_MIN\n",name);
        }
        z = execute(func,x, cycles_fixed);
        if(print_values) {
          printf("%s(%.7f) == %.7f\n",name,F24(x), F24(z));
        }

        check_result(z, zc, minerror, maxerror, &err);

        // Performance
#if PRINT_EVERY_CYCLE_COUNT
        perf_ratio = dsp_math_divide(cycles_float, cycles_fixed, 24);
        printf("Cycles taken to execute %s: %d\n", name, cycles_fixed);
        printf("%s is %.2f times faster than it's floating point equivalent\n", name, F24(perf_ratio));
#endif
```

```
        if(cycles_fixed>worst_cycles_fixed) {
            worst_cycles_fixed = cycles_fixed;
            worst_cycles_input = x;
            worst_cycles_float = cycles_float;
        }

        i++;
        // update x
        if(exponential) {
            if(x==-1) {
                x = 0;   // transition to 0
            } else if(x==0) {
                x = 1;   // transition to positive
            } else if(x<0) {
                x >>= 1; //shift down towards -1
            } else { // positive values
                x <<= 1; // shift up towards max
            }
        } else {
            x = (int64_t) min + (((unsigned)(max - min))>>STEPS) * i;
        }

        // handle done
        if(done_after_next_iteration) done = 1;
        if(x >= (int64_t) max) {
            x = max;
            done_after_next_iteration = 1;
        }
    }

    report_errors(minerror, maxerror, &err);
    unsigned half_range = ERROR_RANGE/2;
    // number of errors -1, 0, +1
    unsigned small_errors = err.errors[half_range-1] + err.errors[half_range] + err.errors[half_range+1];
    //
    unsigned achieved_permille = (1000 * small_errors) / err.num_checked;
    if(achieved_permille < permille) {
        printf("FAIL: only ");
    }
    printf("%d per thousand in one bit error\n", achieved_permille);

#if PRINT_CYCLE_COUNT
        printf("Worst case cycles for executing %s was measured for input %.7f: %d\n", name, F24(
          ↪ worst_cycles_input), worst_cycles_fixed);
        perf_ratio = dsp_math_divide(worst_cycles_float, worst_cycles_fixed, 24);
        printf("%s is %.2f times faster than it's floating point equivalent\n", name, F24(perf_ratio));
#endif

    printstr("\n"); // Dilimiter

    return fail;
}

/*
 * Test the valid input rn=anges
 */
void test_single_input_functions() {

    int fail=0;

    printf("Test Exponential and Logarithmic Functions\n");
    printf("------------------------------------------\n");
    fail += test_input_range(0,"dsp_math_exp", INT32_MIN, Q24(log(127)), LINEAR, -26,  5, 987, PV_OFF);
    fail += test_input_range(2,"dsp_math_log", 1,         INT32_MAX,     LINEAR, -2,  2, 926, PV_OFF);

    printf("Test Squareroot\n");
    printf("---------------\n");
    fail += test_input_range(4,"dsp_math_sqrt", 0,  INT32_MAX,     EXPONENTIAL, -1,  1, 1000, PV_OFF);

    printf("Test Trigonometric Functions\n");
    printf("----------------------------\n");
    fail += test_input_range(6,"dsp_math_sin", -PI_Q8_24, PI_Q8_24,      LINEAR, -1,  1, 1000, PV_OFF);
    fail += test_input_range(8,"dsp_math_cos", -PI_Q8_24, PI_Q8_24,      LINEAR, -1,  1, 1000, PV_OFF);
    fail += test_input_range(10,"dsp_math_atan", INT32_MIN,       INT32_MAX,     1, -1,  EXPONENTIAL, 984,
      ↪ PV_OFF);
```

```
    fail += test_input_range(12,"dsp_math_sinh", -11*ONE_Q8_24>>1, 11*ONE_Q8_24>>1, LINEAR, -40, 40, 726,
      ↪ PV_OFF);  // Should aim for -4 4 800
    fail += test_input_range(14,"dsp_math_cosh", -11*ONE_Q8_24>>1, 11*ONE_Q8_24>>1, LINEAR, -40, 40, 711,
      ↪ PV_OFF);  // Should aim for -4 4, 800

    if (fail != 0) {
        printf("Total failures %d\n", fail);
    }
}

void test_math(void)
{

    int32_t start_time, end_time;
    timer tmr;
    tmr :> start_time;
    tmr :> end_time;
    overhead_time = end_time - start_time;

    printf("Test example for Math functions\n");
    printf("==============================\n");

    test_multipliation_and_division();

    test_single_input_functions();

    exit (0);
}

void divide() {
    int32_t divisor = 3;
    int32_t result = 0x7FFFFFFF;;
    while(1) {
        result = dsp_math_divide(result, divisor, 24);
        if(result==0) result = 0x7FFFFFFF;
    }
}

int main(void) {
    par {
        test_math();
#if DIVIDE_STRESS
        divide();
        divide();
        divide();
        divide();
#endif
    }
    return 0;
}
```

B.4 Matrix Functions

```
// Copyright (c) 2015-2016, XMOS Ltd, All rights reserved
// XMOS DSP Library - Matrix Functions Test Program
// Uses Q24 format

// Include files
#include <stdio.h>
#include <xs1.h>
#include <dsp.h>
#include <math.h>

// Define constants

#define Q_M               8
#define Q_N               24

// Declare global variables and arrays
int32_t  Src1[] = { Q24(.11), Q24(.12), Q24(.13),
                Q24(.21), Q24(.22), Q24(.23),
                Q24(.31), Q24(.32), Q24(.33)};
int32_t  Src2[] = { Q24(.41), Q24(.42), Q24(.43),
                Q24(.51), Q24(.52), Q24(.53),
                Q24(.61), Q24(.62), Q24(.63)};
int32_t         Dst[3*3];

int main(void)
{

    printf ("Matrix multiplication: R = X * Y\n");
    // Input vector
    /*
    int32_t input_vector[2] = {
            Q24(1),
            Q24(1)
    }; // 45 degree vector
    */

    int32_t input_vector[2] = {
            Q24(sqrt(3)/2),
            Q24(0.5)
    }; // 30 degree vector

    printf ("Input column vector X[2] (30 degrees from x axis):\n");
    printf ("%.8f\n", F24 (input_vector[0]));
    printf ("%.8f\n", F24 (input_vector[1]));

    int32_t rotated_vector[2];
    // rotate by 90 degrees (pi/2)
    q8_24 alpha = Q24(1.5707963268);
    cos(alpha);
    int32_t rotation_matrix[4] = {
            dsp_math_cos(alpha), -dsp_math_sin(alpha),
            dsp_math_sin(alpha), dsp_math_cos(alpha),
    };

    dsp_matrix_mulm (rotation_matrix,  // 'input_matrix_X':  Pointer to source data array X
            input_vector,                 // 'input_matrix_Y':  Pointer to source data array Y
            rotated_vector,               // 'result_matrix_R': Pointer to the resulting data array
            2,                            // 'rows_X':    Number of rows in X
            1,                            // 'cols_Y':    Number of columns in Y
            2,                            // 'cols_X_rows_Y'
            24);                          // 'q_format':        Fixed point format, the number of bits making
              ↪  up fractional part

    printf ("Result of multiplying column vector X[2] with rotation matrix Y[2][2] (90 degrees rotation):\n");
    printf ("%.8f\n", F24 (rotated_vector[0]));
    printf ("%.8f\n", F24 (rotated_vector[1]));
                                                  // Matrix negation: R = -X
  dsp_matrix_negate (Src1,                    // 'input_matrix_X':  Pointer/reference to source data
                         Dst,                     // 'result_matrix_R': Pointer to the resulting 2-dimensional
                           ↪  data array
                         3,         // 'row_count':       Number of rows in input and output matrices
                         3);        // 'column_count':    Number of columns in input and output matrices
  printf ("\n");
```

```
printf ("Matrix negation: R = -X\n");
printf ("%lf, %lf, %lf\n", F24 (Dst[0]), F24 (Dst[1]), F24 (Dst[2]));
printf ("%lf, %lf, %lf\n", F24 (Dst[3]), F24 (Dst[4]), F24 (Dst[5]));
printf ("%lf, %lf, %lf\n", F24 (Dst[6]), F24 (Dst[7]), F24 (Dst[8]));
printf ("\n");

                                        // Matrix / scalar addition: R = X + a
dsp_matrix_adds (Src1,                  // 'input_matrix_X':  Pointer/reference to source data
                 Q24(2.),                   // 'scalar_value_A':  Scalar value to add to each 'input'
                   ↪ element
                 Dst,                       // 'result_matrix_R': Pointer to the resulting 2-dimensional
                   ↪  data array
                 3,           // 'row_count':       Number of rows in input and output matrices
                 3);          // 'column_count':    Number of columns in input and output matrices

printf ("Matrix / scalar addition: R = X + a\n");
printf ("%lf, %lf, %lf\n", F24 (Dst[0]), F24 (Dst[1]), F24 (Dst[2]));
printf ("%lf, %lf, %lf\n", F24 (Dst[3]), F24 (Dst[4]), F24 (Dst[5]));
printf ("%lf, %lf, %lf\n", F24 (Dst[6]), F24 (Dst[7]), F24 (Dst[8]));
printf ("\n");

                                        // Matrix / scalar multiplication: R = X * a
dsp_matrix_muls (Src1,                  // 'input_matrix_X':  Pointer/reference to source data
                 Q24(2.),                   // 'scalar_value_A':  Scalar value to multiply each 'input'
                   ↪ element by
                 Dst,                       // 'result_matrix_R': Pointer to the resulting 2-dimensional
                   ↪  data array
                 3,           // 'row_count':       Number of rows in input and output matrices
                 3,           // 'column_count':    Number of columns in input and output matrices
                 Q_N);                      // 'q_format':        Fixed point format, the number of bits
                   ↪  making up fractional part

printf ("Matrix / scalar multiplication: R = X + a\n");
printf ("%lf, %lf, %lf\n", F24 (Dst[0]), F24 (Dst[1]), F24 (Dst[2]));
printf ("%lf, %lf, %lf\n", F24 (Dst[3]), F24 (Dst[4]), F24 (Dst[5]));
printf ("%lf, %lf, %lf\n", F24 (Dst[6]), F24 (Dst[7]), F24 (Dst[8]));
printf ("\n");

                                        // Matrix / matrix addition: R = X + Y
dsp_matrix_addm (Src1,                  // 'input_matrix_X':  Pointer to source data array X
                 Src2,                      // 'input_matrix_Y':  Pointer to source data array Y
                 Dst,                       // 'result_matrix_R': Pointer to the resulting 2-dimensional
                   ↪  data array
                 3,           // 'row_count':       Number of rows in input and output matrices
                 3);          // 'column_count':    Number of columns in input and output matrices

printf ("Matrix / matrix addition: R = X + Y\n");
printf ("%lf, %lf, %lf\n", F24 (Dst[0]), F24 (Dst[1]), F24 (Dst[2]));
printf ("%lf, %lf, %lf\n", F24 (Dst[3]), F24 (Dst[4]), F24 (Dst[5]));
printf ("%lf, %lf, %lf\n", F24 (Dst[6]), F24 (Dst[7]), F24 (Dst[8]));
printf ("\n");

                                        // Matrix / matrix subtraction: R = X - Y
dsp_matrix_subm (Src1,                  // 'input_matrix_X':  Pointer to source data array X
                 Src2,                      // 'input_matrix_Y':  Pointer to source data array Y
                 Dst,                       // 'result_matrix_R': Pointer to the resulting 2-dimensional
                   ↪  data array
                 3,           // 'row_count':       Number of rows in input and output matrices
                 3);          // 'column_count':    Number of columns in input and output matrices

printf ("Matrix / matrix subtraction: R = X - Y\n");
printf ("%lf, %lf, %lf\n", F24 (Dst[0]), F24 (Dst[1]), F24 (Dst[2]));
printf ("%lf, %lf, %lf\n", F24 (Dst[3]), F24 (Dst[4]), F24 (Dst[5]));
printf ("%lf, %lf, %lf\n", F24 (Dst[6]), F24 (Dst[7]), F24 (Dst[8]));
printf ("\n");

                                        // Matrix / matrix multiplication: R = X * Y

                                        // Matrix transposition
dsp_matrix_transpose (Src1,             // 'input_matrix_X':  Pointer to source data array
                      Dst,                  // 'result_matrix_R': Pointer to the resulting 2-dimensional
                        ↪  data array
                      3,       // 'row_count':       Number of rows in input and output matrices
                      3,       // 'column_count':    Number of columns in input and output matrices
```

```
                              Q_N);                 // 'q_format':        Fixed point format, the number of bits
                                ↪  making up fractional part

  printf ("Matrix transposition\n");
  printf ("%lf, %lf, %lf\n", F24 (Dst[0]), F24 (Dst[1]), F24 (Dst[2]));
  printf ("%lf, %lf, %lf\n", F24 (Dst[3]), F24 (Dst[4]), F24 (Dst[5]));
  printf ("%lf, %lf, %lf\n", F24 (Dst[6]), F24 (Dst[7]), F24 (Dst[8]));
  printf ("\n");

  return (0);
}
```

B.5 Statistics Functions

```
// Copyright (c) 2015-2016, XMOS Ltd, All rights reserved
// XMOS DSP Library - Statistics Functions Test Program
// Uses Q24 format

// Include files
#include <stdio.h>
#include <xs1.h>
#include <dsp.h>

// Define constants

#define Q_M               8
#define Q_N               24

#define SAMPLE_LENGTH         50
#define SHORT_SAMPLE_LENGTH   5

// Declare global variables and arrays
int32_t  Src[] = { Q24(.11), Q24(.12), Q24(.13), Q24(.14), Q24(.15), Q24(.16), Q24(.17), Q24(.18), Q24(.19),
  ↪ Q24(.20),
               Q24(.21), Q24(.22), Q24(.23), Q24(.24), Q24(.25), Q24(.26), Q24(.27), Q24(.28), Q24(.29), Q24
                 ↪ (.30),
               Q24(.31), Q24(.32), Q24(.33), Q24(.34), Q24(.35), Q24(.36), Q24(.37), Q24(.38), Q24(.39), Q24
                 ↪ (.40),
               Q24(.41), Q24(.42), Q24(.43), Q24(.44), Q24(.45), Q24(.46), Q24(.47), Q24(.48), Q24(.49), Q24
                 ↪ (.50),
               Q24(.51), Q24(.52), Q24(.53), Q24(.54), Q24(.55), Q24(.56), Q24(.57), Q24(.58), Q24(.59), Q24
                 ↪ (.60)};
int32_t  Src2[] = { Q24(.51), Q24(.52), Q24(.53), Q24(.54), Q24(.55)};
int32_t           Dst[SAMPLE_LENGTH];

int main(void)
{
  int32_t result;

  result =
    dsp_vector_mean (Src,                       // Input vector
                         SAMPLE_LENGTH,             // Vector length
                         Q_N);                      // Q Format N

  printf ("Vector Mean = %lf\n", F24 (result));

  result =
    dsp_vector_power (Src,                      // Input vector
                          SAMPLE_LENGTH,            // Vector length
                          Q_N);                     // Q Format N

  printf ("Vector Power (sum of squares) = %lf\n", F24 (result));

  result =
    dsp_vector_rms (Src,                        // Input vector
                        SAMPLE_LENGTH,              // Vector length
                        Q_N);                       // Q Format N

  printf ("Vector Root Mean Square = %lf\n", F24 (result));

  result =
    dsp_vector_dotprod (Src,                    // Input vector 1
                            Src2,                   // Input vector 2
                            SHORT_SAMPLE_LENGTH,    // Vector length
                            Q_N);                   // Q Format N

  printf ("Vector Dot Product = %lf\n", F24 (result));

  return (0);
}
```

B.6 Vector Functions

```
// Copyright (c) 2015-2016, XMOS Ltd, All rights reserved
// XMOS DSP Library - Vector Functions Test Program
// Uses Q24 format

#include <stdio.h>
#include <xs1.h>
#include <dsp.h>

#define Q_M                   8
#define Q_N                   24

#define SAMPLE_LENGTH         50
#define SHORT_SAMPLE_LENGTH   5

#define COMPLEX_VECTOR_LENGTH 12


int32_t  Src[] = { Q24(.11), Q24(.12), Q24(.13), Q24(.14), Q24(.15), Q24(.16), Q24(.17), Q24(.18), Q24(.19),
  ↪ Q24(.20),
                   Q24(.21), Q24(.22), Q24(.23), Q24(.24), Q24(.25), Q24(.26), Q24(.27), Q24(.28), Q24(.29),
                     ↪ Q24(.30),
                   Q24(.31), Q24(.32), Q24(.33), Q24(.34), Q24(.35), Q24(.36), Q24(.37), Q24(.38), Q24(.39),
                     ↪ Q24(.40),
                   Q24(.41), Q24(.42), Q24(.43), Q24(.44), Q24(.45), Q24(.46), Q24(.47), Q24(.48), Q24(.49),
                     ↪ Q24(.50),
                   Q24(.51), Q24(.52), Q24(.53), Q24(.54), Q24(.55), Q24(.56), Q24(.57), Q24(.58), Q24(.59),
                     ↪ Q24(.60)};
int32_t  Src2[] = { Q24(.51), Q24(.52), Q24(.53), Q24(.54), Q24(.55), Q24(.56), Q24(.57), Q24(.58), Q24(.59),
  ↪ Q24(.60)};
int32_t  Src3[] = { Q24(.61), Q24(.62), Q24(.63), Q24(.64), Q24(.65), Q24(.66), Q24(.67), Q24(.68), Q24(.69),
  ↪ Q24(.70)};
int32_t  Dst[SAMPLE_LENGTH];

int32_t A_real[] = {Q24(1.), Q24(2.), Q24(3.), Q24(4.), Q24(3.), Q24(4.), Q24(1.), Q24(2.), Q24(1.), Q24(3.),
  ↪ Q24(5.), Q24(7.)};
int32_t A_imag[] = {Q24(5.), Q24(6.), Q24(7.), Q24(8.), Q24(7.), Q24(8.), Q24(5.), Q24(6.), Q24(2.), Q24(4.),
  ↪ Q24(6.), Q24(8.)};
int32_t B_real[] = {Q24(3.), Q24(4.), Q24(5.), Q24(6.), Q24(7.), Q24(1.), Q24(3.), Q24(7.), Q24(5.), Q24(6.),
  ↪ Q24(5.), Q24(6.)};
int32_t B_imag[] = {Q24(5.), Q24(6.), Q24(7.), Q24(8.), Q24(9.), Q24(3.), Q24(5.), Q24(9.), Q24(7.), Q24(8.),
  ↪ Q24(7.), Q24(8.)};
int32_t C_real[COMPLEX_VECTOR_LENGTH];
int32_t C_imag[COMPLEX_VECTOR_LENGTH];


int main(void)
{
  int32_t result;
  int32_t i;

  result =
    dsp_vector_minimum (Src,                    // Input vector
                        SAMPLE_LENGTH);         // Vector length

  printf ("Minimum location = %d\n", result);
  printf ("Minimum = %lf\n", F24 (Src[result]));

  result =
    dsp_vector_maximum (Src,                    // Input vector
                        SAMPLE_LENGTH);         // Vector length

  printf ("Maximum location = %d\n", result);
  printf ("Maximum = %lf\n", F24 (Src[result]));

  dsp_vector_negate (Src,                       // Input vector
                     Dst,                       // Output vector
                     SHORT_SAMPLE_LENGTH);      // Vector length

  printf ("Vector Negate Result\n");
  for (i = 0; i < SHORT_SAMPLE_LENGTH; i++)
  {
    printf ("Dst[%d] = %lf\n", i, F24 (Dst[i]));
  }
```

```
dsp_vector_abs (Src,                          // Input vector
                Dst,                              // Output vector
                SHORT_SAMPLE_LENGTH);             // Vector length

printf ("Vector Absolute Result\n");
for (i = 0; i < SHORT_SAMPLE_LENGTH; i++)
{
  printf ("Dst[%d] = %lf\n", i, F24 (Dst[i]));
}

dsp_vector_adds (Src,                         // Input vector
                 Q24(2.),                         // Input scalar
                 Dst,                             // Output vector
                 SHORT_SAMPLE_LENGTH);            // Vector length

printf ("Vector / scalar addition Result\n");
for (i = 0; i < SHORT_SAMPLE_LENGTH; i++)
{
  printf ("Dst[%d] = %lf\n", i, F24 (Dst[i]));
}

dsp_vector_muls (Src,                         // Input vector
                 Q24(2.),                         // Input scalar
                 Dst,                             // Output vector
                 SHORT_SAMPLE_LENGTH,             // Vector length
                 Q_N);                            // Q Format N

printf ("Vector / scalar multiplication Result\n");
for (i = 0; i < SHORT_SAMPLE_LENGTH; i++)
{
  printf ("Dst[%d] = %lf\n", i, F24 (Dst[i]));
}

dsp_vector_addv (Src,                         // Input vector
                 Src2,                            // Input vector 2
                 Dst,                             // Output vector
                 SHORT_SAMPLE_LENGTH);            // Vector length

printf ("Vector / vector addition Result\n");
for (i = 0; i < SHORT_SAMPLE_LENGTH; i++)
{
  printf ("Dst[%d] = %lf\n", i, F24 (Dst[i]));
}

dsp_vector_subv (Src,                         // Input vector
                 Src2,                            // Input vector 2
                 Dst,                             // Output vector
                 SHORT_SAMPLE_LENGTH);            // Vector length

printf ("Vector / vector subtraction Result\n");
for (i = 0; i < SHORT_SAMPLE_LENGTH; i++)
{
  printf ("Dst[%d] = %lf\n", i, F24 (Dst[i]));
}

dsp_vector_mulv (Src,                         // Input vector
                 Src2,                            // Input vector 2
                 Dst,                             // Output vector
                 SHORT_SAMPLE_LENGTH,             // Vector length
                 Q_N);                            // Q Format N

printf ("Vector / vector multiplication Result\n");
for (i = 0; i < SHORT_SAMPLE_LENGTH; i++)
{
  printf ("Dst[%d] = %lf\n", i, F24 (Dst[i]));
}

dsp_vector_mulv_adds (Src,                    // Input vector
                      Src2,                       // Input vector 2
                      Q24(2.),                    // Input scalar
                      Dst,                        // Output vector
                      SHORT_SAMPLE_LENGTH,        // Vector length
                      Q_N);                       // Q Format N
```

```
  printf ("Vector multiplication and scalar addition Result\n");
  for (i = 0; i < SHORT_SAMPLE_LENGTH; i++)
  {
    printf ("Dst[%d] = %lf\n", i, F24 (Dst[i]));
  }

  dsp_vector_muls_addv (Src,                    // Input vector
                        Q24(2.),                    // Input scalar
                        Src2,                       // Input vector 2
                        Dst,                        // Output vector
                        SHORT_SAMPLE_LENGTH,        // Vector length
                        Q_N);                       // Q Format N

  printf ("Vector / Scalar multiplication and vector addition Result\n");
  for (i = 0; i < SHORT_SAMPLE_LENGTH; i++)
  {
    printf ("Dst[%d] = %lf\n", i, F24 (Dst[i]));
  }

  dsp_vector_muls_subv (Src,                    // Input vector
                        Q24(2.),                    // Input scalar
                        Src2,                       // Input vector 2
                        Dst,                        // Output vector
                        SHORT_SAMPLE_LENGTH,        // Vector length
                        Q_N);                       // Q Format N

  printf ("Vector / Scalar multiplication and vector subtraction Result\n");
  for (i = 0; i < SHORT_SAMPLE_LENGTH; i++)
  {
    printf ("Dst[%d] = %lf\n", i, F24 (Dst[i]));
  }

  dsp_vector_mulv_addv (Src,                    // Input vector
                        Src2,                       // Input vector 2
                        Src3,                       // Input vector 2
                        Dst,                        // Output vector
                        SHORT_SAMPLE_LENGTH,        // Vector length
                        Q_N);                       // Q Format N

  printf ("Vector / Vector multiplication and vector addition Result\n");
  for (i = 0; i < SHORT_SAMPLE_LENGTH; i++)
  {
    printf ("Dst[%d] = %lf\n", i, F24 (Dst[i]));
  }

  dsp_vector_mulv_subv (Src,                    // Input vector
                        Src2,                       // Input vector 2
                        Src3,                       // Input vector 2
                        Dst,                        // Output vector
                        SHORT_SAMPLE_LENGTH,        // Vector length
                        Q_N);                       // Q Format N

  printf ("Vector / Vector multiplication and vector subtraction Result\n");
  for (i = 0; i < SHORT_SAMPLE_LENGTH; i++)
  {
    printf ("Dst[%d] = %lf\n", i, F24 (Dst[i]));
  }


  dsp_vector_mulv_complex (A_real, A_imag, B_real, B_imag, C_real, C_imag, COMPLEX_VECTOR_LENGTH, Q_N);

  printf ("Complex vector / Vector multiplication Result\n");
  printf ("C_real = ");
  for (int i = 0; i < COMPLEX_VECTOR_LENGTH; i++)
    printf ("%lf, ", F24 (C_real[i]));
  printf ("\n");
  printf ("C_imag = ");
  for (int i = 0; i < COMPLEX_VECTOR_LENGTH; i++)
    printf ("%lf, ", F24 (C_imag[i]));
  printf ("\n");


  return (0);
}
```

B.7 FFT and inverse FFT

Note: The method for processing two real signals with a single complex FFT was improved. It now requires only half the memory. See Build Configurations tworeals and tworeals_int16_buf.

```
// Copyright (c) 2015-2016, XMOS Ltd, All rights reserved
// XMOS DSP Library - Example to use FFT and inverse FFT

#include <stdio.h>
#include <xs1.h>
#include <xclib.h>
#include <dsp.h>
#include <stdint.h>

#define TRACE_VALUES 1
#if TRACE_VALUES
#define PRINT_FFT_INPUT 1
#define PRINT_FFT_OUTPUT 1
#define PRINT_IFFT_OUTPUT 1
#define N_FFT_POINTS 32
#define PRINT_CYCLE_COUNT 0
#else
#define PRINT_FFT_INPUT 0
#define PRINT_FFT_OUTPUT 0
#define PRINT_IFFT_OUTPUT 0
#define N_FFT_POINTS 4096
#define PRINT_CYCLE_COUNT 1
#endif

#define INPUT_FREQ N_FFT_POINTS/8

#ifndef INT16_BUFFERS
#define INT16_BUFFERS 0 // disabled by default. int32_t buffers is default
#endif

#ifndef TWOREALS
#define TWOREALS 0 // Processing two real signals with a single complex FFT. Disabled by default
#endif

#if TWOREALS
int32_t do_tworeals_fft_and_ifft();
#else
int32_t do_complex_fft_and_ifft();
#endif

// Array holding one complex signal or two real signals
#if INT16_BUFFERS
dsp_complex_short_t data[N_FFT_POINTS];
#else
dsp_complex_t  data[N_FFT_POINTS];
#endif


/**
 * Experiments with functions that generate sine and cosine signals with a defined number of points
 **/
// Macros to ease the use of the sin_N and cos_N functions
// Note: 31-clz(N) == log2(N) when N is power of two
#define SIN(M, N) sin_N(M, 31-clz(N), dsp_sine_ ## N)
#define COS(M, N) cos_N(M, 31-clz(N), dsp_sine_ ## N)

int32_t sin_N(int32_t x, int32_t log2_points_per_cycle, const int32_t sine[]);
int32_t cos_N(int32_t x, int32_t log2_points_per_cycle, const int32_t sine[]);

// generate sine signal with a configurable number of samples per cycle
#pragma unsafe arrays
int32_t sin_N(int32_t x, int32_t log2_points_per_cycle, const int32_t sine[]) {
    // size of sine[] must be equal to num_points!
    int32_t num_points = (1<<log2_points_per_cycle);
    int32_t half_num_points = num_points>>1;

    x = x & (num_points-1); // mask off the index

    switch (x >> (log2_points_per_cycle-2)) { // switch on upper two bits
```

```
       // upper two bits determine the quadrant.
       case 0: return sine[x];
       case 1: return sine[half_num_points-x];
       case 2: return -sine[x-half_num_points];
       case 3: return -sine[num_points-x];
    }
    return 0; // unreachable
}

// generate cosine signal with a configurable number of samples per cycle
#pragma unsafe arrays
int32_t cos_N(int32_t x, int32_t log2_points_per_cycle, const int32_t sine[]) {
    int32_t quarter_num_points = (1<<(log2_points_per_cycle-2));
    return sin_N(x+(quarter_num_points), log2_points_per_cycle, sine); // cos a = sin(a + 2pi/4)
}

int main( void )
{

#if TWOREALS
    do_tworeals_fft_and_ifft();
#else
    do_complex_fft_and_ifft();
#endif
    return 0;
};

#if INT16_BUFFERS
#define RIGHT_SHIFT 17  // shift down to Q14. Q15 would cause overflow.
#else
#define RIGHT_SHIFT 1   // shift down to Q30. Q31 would cause overflow
#endif

void print_data_array() {
#if INT16_BUFFERS
    printf("re,      im         \n");
    for(int32_t i=0; i<N_FFT_POINTS; i++) {
        printf( "%.5f, %.5f\n", F14(data[i].re), F14(data[i].im));
    }
#else
    printf("re,           im         \n");
    for(int32_t i=0; i<N_FFT_POINTS; i++) {
        printf( "%.10f, %.10f\n", F30(data[i].re), F30(data[i].im));
    }
#endif
}

#if TWOREALS
void generate_tworeals_test_signal(int32_t N, int32_t test) {
    switch(test) {
     case 0: {
         printf("++++ Test %d: %d point FFT of two real signals:: re0: %d Hz cosine, re1: %d Hz cosine\n"
                 ,test,N,INPUT_FREQ,INPUT_FREQ);
         for(int32_t i=0; i<N; i++) {
             data[i].re = COS(i, 8) >> RIGHT_SHIFT;
             data[i].im = COS(i, 8) >> RIGHT_SHIFT;
         }
         break;
     }
     case 1: {
         printf("++++ Test %d: %d point FFT of two real signals:: re0: %d Hz sine, re1: %d Hz sine\n"
                 ,test,N,INPUT_FREQ,INPUT_FREQ);
         for(int32_t i=0; i<N; i++) {
             data[i].re = SIN(i, 8) >> RIGHT_SHIFT;
             data[i].im = SIN(i, 8) >> RIGHT_SHIFT;
         }
         break;
     }
     case 2: {
         printf("++++ Test %d: %d point FFT of two real signals:: re0: %d Hz sine, re1: %d Hz cosine\n"
                 ,test,N,INPUT_FREQ,INPUT_FREQ);
         for(int32_t i=0; i<N; i++) {
             data[i].re = SIN(i, 8) >> RIGHT_SHIFT;
             data[i].im = COS(i, 8) >> RIGHT_SHIFT;
         }
```

```
            break;
        }
        case 3: {
            printf("++++ Test %d: %d point FFT of two real signals:: re0: %d Hz cosine, re1: %d Hz sine\n"
                    ,test,N,INPUT_FREQ,INPUT_FREQ);
            for(int32_t i=0; i<N; i++) {
                data[i].re = COS(i, 8) >> RIGHT_SHIFT;
                data[i].im = SIN(i, 8) >> RIGHT_SHIFT;
            }
            break;
        }
        }
#if PRINT_FFT_INPUT
    printf("Generated Two Real Input Signals:\n");
    print_data_array();
#endif
}

int32_t do_tworeals_fft_and_ifft() {
    timer tmr;
    uint32_t start_time, end_time, overhead_time, cycles_taken;
    tmr :> start_time;
    tmr :> end_time;
    overhead_time = end_time - start_time;
#if INT16_BUFFERS
    printf("FFT/iFFT of two real signals of type int16_t\n");
#else
    printf("FFT/iFFT of two real signals of type int32_t\n");
#endif
    printf("============================================\n");

    for(int32_t test=0; test<4; test++) {
        generate_tworeals_test_signal(N_FFT_POINTS, test);

        tmr :> start_time;
#if INT16_BUFFERS
        dsp_complex_t tmp_data[N_FFT_POINTS]; // tmp buffer to enable 32-bit FFT/iFFT
        dsp_fft_short_to_long(data, tmp_data, N_FFT_POINTS); // convert into tmp buffer
        dsp_fft_bit_reverse(tmp_data, N_FFT_POINTS);
        dsp_fft_forward(tmp_data, N_FFT_POINTS, FFT_SINE(N_FFT_POINTS));
        dsp_fft_split_spectrum(tmp_data, N_FFT_POINTS);
        dsp_fft_long_to_short(tmp_data, data, N_FFT_POINTS); // convert from tmp buffer
#else
        dsp_fft_bit_reverse(data, N_FFT_POINTS);
        dsp_fft_forward(data, N_FFT_POINTS, FFT_SINE(N_FFT_POINTS));
        dsp_fft_split_spectrum(data, N_FFT_POINTS);
#endif
        tmr :> end_time;
        cycles_taken = end_time-start_time-overhead_time;
#if PRINT_CYCLE_COUNT
        printf("Cycles taken for %d point FFT of two purely real signals: %d\n", N_FFT_POINTS, cycles_taken);
#endif

#if PRINT_FFT_OUTPUT
        // Print forward complex FFT results
        //printf( "First half of Complex FFT output spectrum of Real signal 0 (cosine):\n");
        printf( "FFT output of two half spectra. Second half could be discarded due to symmetry without
          ↪ loosing information\n");
        printf( "spectrum of first signal in data[0..N/2-1]. spectrum of second signa in data[N/2..N-1]:\n");

        print_data_array();
#if 0
        printf( "First half of Complex FFT output spectrum of Real signal 1 (sine):\n");
        printf("re,           im         \n");
        for(int32_t i=0; i<N_FFT_POINTS; i++) {
            printf( "%.10f, %.10f\n", F30(data[i].re), F30(data[i].im));
        }
#endif
#endif

        tmr :> start_time;
#if INT16_BUFFERS
        dsp_fft_short_to_long(data, tmp_data, N_FFT_POINTS); // convert into tmp buffer
        dsp_fft_merge_spectra(tmp_data, N_FFT_POINTS);
        dsp_fft_bit_reverse(tmp_data, N_FFT_POINTS);
```

```
        dsp_fft_inverse(tmp_data, N_FFT_POINTS, FFT_SINE(N_FFT_POINTS));
        dsp_fft_long_to_short(tmp_data, data, N_FFT_POINTS); // convert from tmp buffer
#else
        dsp_fft_merge_spectra(data, N_FFT_POINTS);
        dsp_fft_bit_reverse(data, N_FFT_POINTS);
        dsp_fft_inverse(data, N_FFT_POINTS, FFT_SINE(N_FFT_POINTS));
#endif

        tmr :> end_time;
        cycles_taken = end_time-start_time-overhead_time;
#if PRINT_CYCLE_COUNT
        printf("Cycles taken for iFFT: %d\n", cycles_taken);
#endif

#if PRINT_IFFT_OUTPUT
        printf( "////// Time domain signal after dsp_fft_inverse\n");
        print_data_array();
#endif
        printf("\n" ); // Test delimiter
    }

    printf( "DONE.\n" );
    return 0;
}
# else  // Complex Signals
void generate_complex_test_signal(int32_t N, int32_t test) {
    switch(test) {
    case 0: {
        printf("++++ Test 0: %d point FFT/iFFT of complex signal:: Real: %d Hz cosine, Imag: 0\n"
                ,N_FFT_POINTS,N_FFT_POINTS/8);
        for(int32_t i=0; i<N; i++) {
            data[i].re = COS(i, 8) >> RIGHT_SHIFT;
            data[i].im = 0;
        }
        break;
    }
    case 1: {
        printf("++++ Test 1: %d point FFT/iFFT of complex signal:: Real: %d Hz sine, Imag: 0\n"
                ,N_FFT_POINTS,N_FFT_POINTS/8);
        for(int32_t i=0; i<N; i++) {
            data[i].re = SIN(i, 8) >> RIGHT_SHIFT;
            data[i].im = 0;
        }
        break;
    }
    case 2: {
        printf("++++ Test 2: %d point FFT/iFFT of complex signal: Real: 0, Imag: %d Hz cosine\n"
                ,N_FFT_POINTS,N_FFT_POINTS/8);
        for(int32_t i=0; i<N; i++) {
            data[i].re = 0;
            data[i].im = COS(i, 8) >> RIGHT_SHIFT;
        }
        break;
    }
    case 3: {
        printf("++++ Test 3: %d point FFT/iFFT of complex signal: Real: 0, Imag: %d Hz sine\n"
                ,N_FFT_POINTS,N_FFT_POINTS/8);
        for(int32_t i=0; i<N; i++) {
            data[i].re = 0;
            data[i].im = SIN(i, 8) >> RIGHT_SHIFT;
        }
        break;
    }
    }
#if PRINT_FFT_INPUT
    printf("Generated Signal:\n");
    print_data_array();
#endif
}

int32_t do_complex_fft_and_ifft() {
    timer tmr;
    uint32_t start_time, end_time, overhead_time, cycles_taken;
    tmr :> start_time;
    tmr :> end_time;
```

```
    overhead_time = end_time - start_time;

#if INT16_BUFFERS
    printf("FFT/iFFT of a complex signal of type int16_t\n");
#else
    printf("FFT/iFFT of a complex signal of type int32_t\n");
#endif
    printf("=========================================\n");

    for(int32_t test=0; test<4; test++) {
        generate_complex_test_signal(N_FFT_POINTS, test);

        tmr :> start_time;

#if INT16_BUFFERS
        dsp_complex_t tmp_data[N_FFT_POINTS]; // tmp buffer to enable 32-bit FFT/iFFT
        // convert into int32_t temporary buffer
        dsp_fft_short_to_long(data, tmp_data, N_FFT_POINTS);
        // 32 bit FFT
        dsp_fft_bit_reverse(tmp_data, N_FFT_POINTS);
        dsp_fft_forward(tmp_data, N_FFT_POINTS, FFT_SINE(N_FFT_POINTS));
        // convert back into int16_t buffer
        dsp_fft_long_to_short(tmp_data, data, N_FFT_POINTS);
#else
        // 32 bit FFT
        dsp_fft_bit_reverse(data, N_FFT_POINTS);
        dsp_fft_forward(data, N_FFT_POINTS, FFT_SINE(N_FFT_POINTS));
#endif
        tmr :> end_time;
        cycles_taken = end_time-start_time-overhead_time;


#if PRINT_CYCLE_COUNT
        printf("Cycles taken for %d point FFT of complex signal: %d\n", N_FFT_POINTS, cycles_taken);
#endif


#if PRINT_FFT_OUTPUT
        printf( "FFT output:\n");
        print_data_array();
#endif

        tmr :> start_time;
#if INT16_BUFFERS
        // convert into int32_t temporary buffer
        dsp_fft_short_to_long(data, tmp_data, N_FFT_POINTS);
        // 32 bit iFFT
        dsp_fft_bit_reverse(tmp_data, N_FFT_POINTS);
        dsp_fft_inverse(tmp_data, N_FFT_POINTS, FFT_SINE(N_FFT_POINTS));
        // convert back into int16_t buffer
        dsp_fft_long_to_short(tmp_data, data, N_FFT_POINTS);
#else
        // 32 bit iFFT
        dsp_fft_bit_reverse(data, N_FFT_POINTS);
        dsp_fft_inverse(data, N_FFT_POINTS, FFT_SINE(N_FFT_POINTS));
#endif
        tmr :> end_time;
        cycles_taken = end_time-start_time-overhead_time;
#if PRINT_CYCLE_COUNT
        printf("Cycles taken for iFFT: %d\n", cycles_taken);
#endif

#if PRINT_IFFT_OUTPUT
        printf( "////// Time domain signal after dsp_fft_inverse\n");
        print_data_array();
#endif
        printf("\n" ); // Test delimiter
    }

    printf( "DONE.\n" );
    return 0;

}
#endif
```

B.8 FFT Processing of signals received through a double buffer

```
// Copyright (c) 2016, XMOS Ltd, All rights reserved

#include <stdio.h>
#include <xs1.h>
#include <dsp.h>
#include <stdlib.h>
#include <math.h>

/**
Example showing FFT processing of a configurable number of input signals received through a double buffer.
-----------------------------------------------------------------------------------------------------------


This example shows how two tasks can implement a double buffering
mechanism accessing a shared memory area.

The task produce_samples fills one buffer with a stereo signal
whilst the do_fft task processes the other buffer
"in place" which means the buffer is also used to store the real part of the result
Tasks access these buffers via *movable* pointers. These pointers can
be safely transferred between tasks without any race conditions
between the tasks using them.

When the the produce_samples taks has finished filling the buffer, it calls the
swap function of the interface to synchronise with the do_fft task and swap pointers (buffers)

.. aafig::

   +---------------+                    +------------------+
   |  `do_fft`     |   `bufswap_i`      | `produce_samples`|
   |               +<------------------>|                  |
   |   +---------+ |   `interface`      |   +---------+    |
   |   |`movable`| |                    |   |`movable`|    |
   |   |`pointer`| |                    |   |`pointer`|    |
   |   +----+----+ |                    |   +----+----+    |
   |        |      |                    |        |         |
   +--------|------+                    +--------|---------+
            |              `MEMORY`              |
            |          +------+------+           |
            |          |`buf1`|`buf2`|           |
            +--------->+      |      +<----------+
                       |      |      |
                       |      |      |
                       +------+------+
**/

/** Global configuration defines **/

#ifndef NUM_CHANS
// Number of input channels
#define NUM_CHANS 4
#endif

#ifndef N_FFT_POINTS
// FFT Points
#define N_FFT_POINTS 256
#endif

#ifndef INT16_BUFFERS
#define INT16_BUFFERS 0 // Disabled by default
#endif

#ifndef TWOREALS
#define TWOREALS 0 // Processing two real signals with a single complex FFT. Disabled by default
#endif

#define SAMPLE_FREQ 48000
#define SAMPLE_PERIOD_CYCLES XS1_TIMER_HZ/SAMPLE_FREQ

#ifndef CHECK_TIMING
#define CHECK_TIMING 0
#endif
```

```
#ifndef PRINT_INPUTS_AND_OUTPUTS
#define PRINT_INPUTS_AND_OUTPUTS 0
#endif

/****/

/** Declaration of Data Types and Memory Buffers **/

// Array holding one complex signal or two real signals
#if INT16_BUFFERS
#define SIGNAL_ARRAY_TYPE dsp_complex_short_t
#define OUTPUT_SUM_TYPE int32_t // double precision variable to avoid overflow in addition
#else
#define SIGNAL_ARRAY_TYPE dsp_complex_t
#define OUTPUT_SUM_TYPE int64_t // double precision variable to avoid overflow in addition
#endif

#if TWOREALS
#define NUM_SIGNAL_ARRAYS NUM_CHANS/2
#else
#define NUM_SIGNAL_ARRAYS NUM_CHANS
#endif

/** Union to store blocks of samples for multiple digital signals
*/
typedef union {
    SIGNAL_ARRAY_TYPE data[NUM_SIGNAL_ARRAYS][N_FFT_POINTS];  // time domain or frequency domain signals
    SIGNAL_ARRAY_TYPE half_spectra[NUM_SIGNAL_ARRAYS*2][N_FFT_POINTS/2];  // frequency domain half spectra
} multichannel_sample_block_s ;
/****/

/** Print Functions **/
void print_signal(SIGNAL_ARRAY_TYPE signal[N_FFT_POINTS]) {
#if INT16_BUFFERS
       printf("re,      im         \n");
       for(int32_t i=0; i<N_FFT_POINTS; i++) {
           //printf( "0x%x, 0x%x\n", signal[i].re, signal[i].im);
           printf( "%.5f, %.5f\n", F14(signal[i].re), F14(signal[i].im));
       }

#else
       printf("re,      im         \n");
       for(int32_t i=0; i<N_FFT_POINTS; i++) {
           printf( "%.8f, %.8f\n", F24(signal[i].re), F24(signal[i].im));
       }
#endif
}

void print_buffer(multichannel_sample_block_s *buffer) {
    for(int32_t c=0; c<NUM_SIGNAL_ARRAYS; c++) {
        print_signal(buffer->data[c]);
    }
}
/****/


/**
 The interface between the two tasks is a single transaction that swaps
 the movable pointers. It has an argument that is a reference to a
 movable pointer. Since it is a reference the server side of the
 connection can update the argument.
*/

interface bufswap_i {
  void swap(multichannel_sample_block_s * movable &x);
};

/**
 The do_fft task takes as arguments the server end of an interface
 connection and the initial buffer it is going to process. It is
 initialized by creating a movable pointer to this buffer and then
 processing it.
*/
#if !(XCC_VERSION_MAJOR >= 1403)
static SIGNAL_ARRAY_TYPE output[N_FFT_POINTS];
```

```
#endif

void do_fft(server interface bufswap_i input,
        multichannel_sample_block_s * initial_buffer)
{
  multichannel_sample_block_s * movable buffer = initial_buffer;

  timer tmr; uint32_t start_time, end_time, overhead_time;
  tmr :> start_time;
  tmr :> end_time;
  overhead_time = end_time - start_time;

#if INT16_BUFFERS
  printf("%d Point FFT Processing of %d int16_t signals received through a double buffer\n"
                ,N_FFT_POINTS,NUM_CHANS);
#else
  printf("%d Point FFT Processing of %d int32_t signals received through a double buffer\n"
                ,N_FFT_POINTS,NUM_CHANS);
#endif

  /** The main loop of the filling task waits for a swap transaction with
      the other task and implements the swap of pointers. After that it
      fills the new buffer it has been given:
  */
  while (1) {
    // swap buffers
    select {
      case input.swap(multichannel_sample_block_s * movable &input_buf):
        // Swapping uses the 'move' operator. This operator transfers the
        // pointer to a new variable, setting the original variable to null.
        // The 'display_buffer' variable is a reference, so updating it will
        // update the pointer passed in by the other task.
        multichannel_sample_block_s * movable tmp;
        tmp = move(input_buf);
        input_buf = move(buffer);
        buffer = move(tmp);

#if PRINT_INPUTS_AND_OUTPUTS
        print_buffer(buffer);
#endif

        tmr :> start_time;

        // Do FFTs
        for(int32_t a=0; a<NUM_SIGNAL_ARRAYS; a++) {
            // process the new buffer "in place"
    #if INT16_BUFFERS
            dsp_complex_t tmp_buffer[N_FFT_POINTS];
            dsp_fft_short_to_long(buffer->data[a], tmp_buffer, N_FFT_POINTS); // convert into tmp buffer
            dsp_fft_bit_reverse(tmp_buffer, N_FFT_POINTS);
            dsp_fft_forward(tmp_buffer, N_FFT_POINTS, FFT_SINE(N_FFT_POINTS));
    #if TWOREALS
            dsp_fft_split_spectrum(tmp_buffer, N_FFT_POINTS);

    #endif
            dsp_fft_long_to_short(tmp_buffer, buffer->data[a], N_FFT_POINTS); // convert from tmp buffer
    ////// 32 bit buffers
    #else
            dsp_fft_bit_reverse(buffer->data[a], N_FFT_POINTS);
            dsp_fft_forward(buffer->data[a], N_FFT_POINTS, FFT_SINE(N_FFT_POINTS));
    #if TWOREALS
            dsp_fft_split_spectrum(buffer->data[a], N_FFT_POINTS);
    #endif

    #endif

        }

        // Process the frequency domain of all NUM_CHANS channels.
        // 1. Lowpass
        // cutoff frequency = (Fs/N_FFT_POINTS * cutoff_index)
        // With cutoff_index = N_FFT_POINTS/M:
        // cutoff frequency = Fs/M

        uint32_t cutoff_idx = N_FFT_POINTS/4;
```

```
        // 2. Calculate average per frequency bin into the output signal array.

        // To calculate the average over all channels.
        // To divide by NUM_CHANS, shift down throughout the loop log2(NUM_CHANS) times to avoid overflow.
        uint32_t log_num_chan = log2(NUM_CHANS);
        uint32_t step = NUM_CHANS/log_num_chan;
        uint shift_idx = step;

#if (XCC_VERSION_MAJOR >= 1403)
        static SIGNAL_ARRAY_TYPE output[N_FFT_POINTS];
#endif

    #if TWOREALS
        for(unsigned i = 0; i < N_FFT_POINTS/2; i++) {
          OUTPUT_SUM_TYPE output_re = 0;
          OUTPUT_SUM_TYPE output_im = 0;
          for(int32_t c=0; c<NUM_CHANS; c++) {
             if(i>=cutoff_idx) {
               buffer->half_spectra[c][i].re = 0;
               buffer->half_spectra[c][i].im = 0;
             }
             output_re += buffer->half_spectra[c][i].re;
             output_im += buffer->half_spectra[c][i].im;
          }
          output[i].re = (OUTPUT_SUM_TYPE) output_re/NUM_CHANS; // average
          output[i].im = (OUTPUT_SUM_TYPE) output_im/NUM_CHANS; // average
        }
    #else // Complex
        for(unsigned i = 0; i < N_FFT_POINTS/2; i++) {
          OUTPUT_SUM_TYPE output_re = 0, output_re_ri = 0;
          OUTPUT_SUM_TYPE output_im = 0, output_im_ri = 0;
          uint32_t ri = N_FFT_POINTS-i; // reverse index
          for(int32_t c=0; c<NUM_CHANS; c++) {
            if(i>=cutoff_idx) {
              buffer->data[c][i].re = 0;
              buffer->data[c][i].im = 0;
              if(i>0) {
                buffer->data[c][N_FFT_POINTS-i].re = 0;
                buffer->data[c][N_FFT_POINTS-i].im = 0;
              }
            }
            output_re += buffer->data[c][i].re;
            output_im += buffer->data[c][i].im;
            if(i>0) {
              output_re_ri += buffer->data[c][ri].re;
              output_im_ri += buffer->data[c][ri].im;
            }
          }

          output[i].re = (OUTPUT_SUM_TYPE) output_re/NUM_CHANS; // average
          output[i].im = (OUTPUT_SUM_TYPE) output_im/NUM_CHANS; // average
          if(i>0) {
            output[ri].re = (OUTPUT_SUM_TYPE) output_re_ri/NUM_CHANS; // average
            output[ri].im = (OUTPUT_SUM_TYPE) output_im_ri/NUM_CHANS; // average
          }
        }
    #endif

        // Todo: Add iFFT

        tmr :> end_time;
        int32_t cycles_taken = end_time-start_time-overhead_time;

#if CHECK_TIMING
#if TWOREALS
        printf("%d Point FFT processing of %d real sequences 'in place' in double buffer %x took %d cycles\n"
                ,N_FFT_POINTS,NUM_CHANS,buffer,cycles_taken);
#else
        printf("%d Point FFT processing of %d complex sequences 'in place' in double buffer %x took %d cycles\
           ↪ n"
                ,N_FFT_POINTS,NUM_CHANS,buffer,cycles_taken);
#endif
        if(cycles_taken>SAMPLE_PERIOD_CYCLES*N_FFT_POINTS) {
            printf("Timing Check ERROR: Max allowed cycles at Fs %d Hz is %d\n",SAMPLE_FREQ,
              ↪ SAMPLE_PERIOD_CYCLES*N_FFT_POINTS);
```

```
        } else {
            printf("Timing Check PASS: Max allowed cycles at Fs %d Hz is %d\n",SAMPLE_FREQ,
              ↪ SAMPLE_PERIOD_CYCLES*N_FFT_POINTS);
        }
#endif

#if PRINT_INPUTS_AND_OUTPUTS
        print_buffer(buffer);
#endif
        printf("Processed output signal\n");
        print_signal(output);

        break;
    }
    // fill the buffer with data
  }
}

/** Utility functions for signal generation **/
int32_t scaled_sin(q8_24 x) {
   q8_24 y = dsp_math_sin(x);
#if INT16_BUFFERS
   y >>= 10; // convert to Q14
#else
#endif
   return y;
}
int32_t scaled_cos(q8_24 x) {
   q8_24 y = dsp_math_cos(x);
#if INT16_BUFFERS
   y >>= 10; // convert to Q14
#else
#endif
   return y;
}
/****/

/**
 The displaying task takes the other end of the interface connection
 and its initial buffer as arguments. It also creates a movable pointer
 to that buffer.
**/

#define MAX_SAMPLE_PERIODS 1

void produce_samples(client interface bufswap_i filler,
        multichannel_sample_block_s * initial_buffer) {
  multichannel_sample_block_s * movable buffer = initial_buffer;
  timer tmr;
  int32_t t;
  static int32_t counter;
  int32_t done = 0;

  tmr :> t;
  /** The main loop of the display task first calls the 'swap' transaction,
      which synchronizes with the fill task and updates the 'buffer' pointer
      to the new swapped memory location. After that it calls an auxiliary
      'display' function to do the actual displaying. This function is
      application dependent and not defined here.
   **/
  while(!done) {
    //fill the next buffer
    for(int32_t a=0; a<NUM_SIGNAL_ARRAYS; a++) {
      // points per cycle. divide by power of two to ensure signals fit into the FFT window
      int32_t ppc = N_FFT_POINTS/(1<<a);
      printf("Points Per Cycle is %d\n", ppc);

      for(int32_t i=0; i<N_FFT_POINTS; i++) {
        // generate input signals

        // Equation: x = 2pi * i/ppc = 2pi * ((i%ppc) / ppc))
        q8_24 factor = ((i%ppc) << 24) / ppc; // factor is always < Q24(1)
        q8_24 x = dsp_math_multiply(PI2_Q8_24, factor, 24);

#if TWOREALS
```

```
        buffer->data[a][i].re = scaled_sin(x);
        buffer->data[a][i].im = scaled_cos(x);
#else
        buffer->data[a][i].re = scaled_sin(x);
        buffer->data[a][i].im = 0;
#endif

      }

      // wait until next sample period
      tmr when timerafter(t+SAMPLE_PERIOD_CYCLES) :> t;
    }

    // swap buffers
    filler.swap(buffer);
    counter++;

    if(counter == MAX_SAMPLE_PERIODS) {
        exit(0);
    }

  }
}

/**
 The application runs both of these tasks in parallel using a 'par'
 statement. The two global buffers are passed into
 the two tasks:
**/

// make global to enforce 64 bit alignment
multichannel_sample_block_s buffer0;
multichannel_sample_block_s buffer1;

int main() {

  interface bufswap_i bufswap;
  par {
      produce_samples(bufswap, &buffer1);
      do_fft(bufswap, &buffer0);
  }
  return 0;
}
```

B.9 Downsample by 3 Function

```
// Copyright (c) 2016, XMOS Ltd, All rights reserved
// XMOS DSP Library - Downsample by factor of 3 example
// Uses Signed 32b Integer format

// Include files
#include <stdio.h>
#include <xs1.h>
#include <dsp.h>
#include <stdint.h>

#define NUM_CHANNELS  2
#define NUM_OUTPUT_SAMPLES  128

// 0db 1KHz 48KHz sinewave - pure tone test
const int32_t s1k_0db_48[1024] = {0, 277499757, 550251420, 813588109, 1063004099, 1294231806, 1503314879,
  ↪ 1686675721, 1841177179, 1964175542, 2053566235, 2107819915, 2126008198, 2107819915, 2053566235,
  ↪ 1964175414, 1841177051, 1686675593, 1503314751, 1294231678, 1063003971, 813587981, 550251228, 277499565,
  ↪  -185, -277499949, -550251612, -813588301, -1063004291, -1294231934, -1503315007, -1686675849,
  ↪ -1841177179, -1964175542, -2053566235, -2107819915, -2126008198, -2107819915, -2053566107, -1964175286,
  ↪ -1841176923, -1686675465, -1503314623, -1294231550, -1063003779, -813587789, -550251036, -277499405,
  ↪ 371, 277500141, 550251804, 813588493, 1063004419, 1294232062, 1503315135, 1686675977, 1841177307,
  ↪ 1964175670, 2053566363, 2107819915, 2126008198, 2107819787, 2053566107, 1964175286, 1841176923,
  ↪ 1686675337, 1503314495, 1294231422, 1063003651, 813587597, 550250908, 277499213, -557, -277500301,
  ↪ -550251996, -813588621, -1063004611, -1294232318, -1503315263, -1686676105, -1841177435, -1964175670,
  ↪ -2053566363, -2107820043, -2126008198, -2107819787, -2053566107, -1964175158, -1841176795, -1686675337,
  ↪ -1503314367, -1294231166, -1063003459, -813587469, -550250716, -277499021, 743, 277500493, 550252124,
  ↪ 813588813, 1063004739, 1294232446, 1503315391, 1686676233, 1841177563, 1964175798, 2053566491,
  ↪ 2107820043, 2126008198, 2107819787, 2053565979, 1964175158, 1841176667, 1686675209, 1503314239,
  ↪ 1294231038, 1063003331, 813587277, 550250524, 277498829, -929, -277500685, -550252316, -813589005,
  ↪ -1063004931, -1294232574, -1503315519, -1686676361, -1841177563, -1964175798, -2053566491, -2107820043,
  ↪ -2126008198, -2107819787, -2053565979, -1964175030, -1841176539, -1686675081, -1503314111, -1294230910,
  ↪ -1063003139, -813587085, -550250332, -277498669, 1115, 277500877, 550252508, 813589197, 1063005123,
  ↪ 1294232702, 1503315647, 1686676361, 1841177691, 1964175926, 2053566491, 2107820043, 2126008198,
  ↪ 2107819787, 2053565851, 1964175030, 1841176539, 1686674953, 1503313983, 1294230782, 1063003011,
  ↪ 813586957, 550250140, 277498477, -1301, -277501037, -550252700, -813589325, -1063005251, -1294232830,
  ↪ -1503315775, -1686676489, -1841177819, -1964175926, -2053566619, -2107820043, -2126008198, -2107819659,
  ↪ -2053565851, -1964174902, -1841176411, -1686674825, -1503313855, -1294230654, -1063002819, -813586765,
  ↪ -550250012, -277498285, 1486, 277501229, 550252892, 813589517, 1063005443, 1294232958, 1503315903,
  ↪ 1686676617, 1841177819, 1964176054, 2053566619, 2107820171, 2126008198, 2107819659, 2053565851,
  ↪ 1964174902, 1841176283, 1686674697, 1503313727, 1294230526, 1063002691, 813586573, 550249820, 277498093,
  ↪  -1672, -277501421, -550253020, -813589709, -1063005571, -1294233086, -1503316031, -1686676745,
  ↪ -1841177947, -1964176182, -2053566619, -2107820171, -2126008198, -2107819659, -2053565723, -1964174774,
  ↪ -1841176155, -1686674569, -1503313471, -1294230398, -1063002499, -813586445, -550249628, -277497933,
  ↪ 1858, 277501613, 550253212, 813589837, 1063005763, 1294233342, 1503316159, 1686676873, 1841178075,
  ↪ 1964176182, 2053566747, 2107820171, 2126008198, 2107819659, 2053565723, 1964174646, 1841176155,
  ↪ 1686674441, 1503313343, 1294230142, 1063002371, 813586253, 550249436, 277497741, -2044, -277501805,
  ↪ -550253404, -813590029, -1063005891, -1294233470, -1503316287, -1686677001, -1841178203, -1964176310,
  ↪ -2053566747, -2107820171, -2126008198, -2107819659, -2053565723, -1964174646, -1841176027, -1686674441,
  ↪ -1503313215, -1294230014, -1063002179, -813586061, -550249244, -277497549, 2230, 277501965, 550253596,
  ↪ 813590221, 1063006083, 1294233598, 1503316415, 1686677129, 1841178203, 1964176310, 2053566875,
  ↪ 2107820171, 2126008198, 2107819659, 2053565595, 1964174518, 1841175899, 1686674313, 1503313087,
  ↪ 1294229886, 1063002051, 813585933, 550249116, 277497357, -2416, -277502157, -550253788, -813590349,
  ↪ -1063006211, -1294233726, -1503316543, -1686677257, -1841178331, -1964176438, -2053566875, -2107820171,
  ↪ -2126008198, -2107819531, -2053565595, -1964174518, -1841175899, -1686674185, -1503312959, -1294229758,
  ↪ -1063001859, -813585741, -550248924, -277497197, 2602, 277502349, 550253916, 813590541, 1063006403,
  ↪ 1294233854, 1503316671, 1686677385, 1841178459, 1964176438, 2053566875, 2107820299, 2126008198,
  ↪ 2107819531, 2053565467, 1964174390, 1841175771, 1686674057, 1503312831, 1294229630, 1063001731,
  ↪ 813585549, 550248732, 277497005, -2787, -277502541, -550254108, -813590733, -1063006531, -1294233982,
  ↪ -1503316799, -1686677385, -1841178587, -1964176566, -2053567003, -2107820299, -2126008198, -2107819531,
  ↪ -2053565467, -1964174390, -1841175643, -1686673929, -1503312703, -1294229502, -1063001539, -813585357,
  ↪ -550248540, -277496813, 2973, 277502701, 550254300, 813590861, 1063006723, 1294234238, 1503316927,
  ↪ 1686677513, 1841178587, 1964176566, 2053567003, 2107820299, 2126008198, 2107819531, 2053565467,
  ↪ 1964174262, 1841175515, 1686673801, 1503312575, 1294229374, 1063001411, 813585229, 550248348, 277496621,
  ↪  -3159, -277502893, -550254492, -813591053, -1063006851, -1294234366, -1503317055, -1686677641,
  ↪ -1841178715, -1964176694, -2053567003, -2107820299, -2126008198, -2107819531, -2053565339, -1964174262,
  ↪ -1841175515, -1686673673, -1503312447, -1294229118, -1063001219, -813585037, -550248220, -277496461,
  ↪ 3345, 277503085, 550254684, 813591245, 1063007043, 1294234494, 1503317183, 1686677769, 1841178843,
  ↪ 1964176822, 2053567131, 2107820299, 2126008198, 2107819403, 2053565339, 1964174134, 1841175387,
  ↪ 1686673545, 1503312319, 1294228990, 1063001091, 813584845, 550248028, 277496269, -3531, -277503277,
  ↪ -550254812, -813591373, -1063007171, -1294234622, -1503317311, -1686677897, -1841178843, -1964176822,
  ↪ -2053567131, -2107820427, -2126008198, -2107819403, -2053565339, -1964174006, -1841175259, -1686673417,
  ↪ -1503312191, -1294228862, -1063000899, -813584717, -550247836, -277496077, 3717, 277503437, 550255004,
  ↪ 813591565, 1063007363, 1294234750, 1503317439, 1686678025, 1841178971, 1964176950, 2053567259,
```

```
  ↪ 2107820427, 2126008198, 2107819403, 2053565211, 1964174006, 1841175131, 1686673417, 1503312063,
  ↪ 1294228734, 1063000771, 813584525, 550247644, 277495885, -3903, -277503629, -550255196, -813591757,
  ↪ -1063007491, -1294234878, -1503317567, -1686678153, -1841179099, -1964176950, -2053567259, -2107820427,
  ↪ -2126008198, -2107819403, -2053565211, -1964173878, -1841175131, -1686673289, -1503311935, -1294228606,
  ↪ -1063000579, -813584333, -550247452, -277495693, 4088, 277503821, 550255388, 813591885, 1063007683,
  ↪ 1294235006, 1503317695, 1686678281, 1841179227, 1964177078, 2053567259, 2107820427, 2126008198,
  ↪ 2107819403, 2053565083, 1964173878, 1841175003, 1686673161, 1503311807, 1294228478, 1063000451,
  ↪ 813584205, 550247324, 277495533, -4274, -277504013, -550255580, -813592077, -1063007811, -1294235262,
  ↪ -1503317823, -1686678281, -1841179227, -1964177078, -2053567387, -2107820427, -2126008198, -2107819275,
  ↪ -2053565083, -1964173750, -1841174875, -1686673033, -1503311679, -1294228222, -1063000259, -813584013,
  ↪ -550247132, -277495341, 4460, 277504173, 550255708, 813592269, 1063008003, 1294235390, 1503317951,
  ↪ 1686678409, 1841179355, 1964177206, 2053567387, 2107820555, 2126008198, 2107819275, 2053565083,
  ↪ 1964173750, 1841174875, 1686672905, 1503311551, 1294228094, 1063000131, 813583821, 550246940, 277495149,
  ↪  -4646, -277504365, -550255900, -813592397, -1063008131, -1294235518, -1503318079, -1686678537,
  ↪ -1841179483, -1964177206, -2053567387, -2107820555, -2126008198, -2107819275, -2053564955, -1964173622,
  ↪ -1841174747, -1686672777, -1503311423, -1294227966, -1062999939, -813583693, -550246748, -277494957,
  ↪ 4832, 277504557, 550256092, 813592589, 1063008323, 1294235646, 1503318207, 1686678665, 1841179611,
  ↪ 1964177334, 2053567515, 2107820555, 2126008198, 2107819275, 2053564955, 1964173622, 1841174619,
  ↪ 1686672649, 1503311295, 1294227838, 1062999811, 813583501, 550246556, 277494797, -5018, -277504749,
  ↪ -550256284, -813592781, -1063008451, -1294235774, -1503318335, -1686678793, -1841179611, -1964177462,
  ↪ -2053567515, -2107820555, -2126008198, -2107819275, -2053564955, -1964173494, -1841174491, -1686672521,
  ↪ -1503311167, -1294227710, -1062999619, -813583309, -550246428, -277494605, 5204, 277504909, 550256476,
  ↪ 813592973, 1063008643, 1294235902, 1503318463, 1686678921, 1841179739, 1964177462, 2053567643,
  ↪ 2107820555, 2126008198, 2107819147, 2053564827, 1964173366, 1841174491, 1686672521, 1503311039,
  ↪ 1294227582, 1062999427, 813583181, 550246236, 277494413, -5389, -277505101, -550256604, -813593101,
  ↪ -1063008771, -1294236158, -1503318719, -1686679049, -1841179867, -1964177590, -2053567643, -2107820555,
  ↪ -2126008198, -2107819147, -2053564827, -1964173366, -1841174363, -1686672393, -1503310911, -1294227454,
  ↪ -1062999299, -813582989, -550246044, -277494221, 5575, 277505293, 550256796, 813593293, 1063008963,
  ↪ 1294236286, 1503318847, 1686679177, 1841179867, 1964177590, 2053567643, 2107820683, 2126008198,
  ↪ 2107819147, 2053564699, 1964173238, 1841174235, 1686672265, 1503310783, 1294227198, 1062999107,
  ↪ 813582797, 550245852, 277494061, -5761, -277505485, -550256988, -813593485, -1063009091, -1294236414,
  ↪ -1503318975, -1686679305, -1841179995, -1964177718, -2053567771, -2107820683, -2126008198, -2107819147,
  ↪ -2053564699, -1964173238, -1841174107, -1686672137, -1503310655, -1294227070, -1062998979, -813582669,
  ↪ -550245660, -277493869, 5947, 277505645, 550257180, 813593613, 1063009283, 1294236542, 1503319103,
  ↪ 1686679305, 1841180123, 1964177718, 2053567771, 2107820683, 2126008198, 2107819147, 2053564699,
  ↪ 1964173110, 1841174107, 1686672009, 1503310527, 1294226942, 1062998787, 813582477, 550245532, 277493677,
  ↪  -6133, -277505837, -550257372, -813593805, -1063009411, -1294236670, -1503319231, -1686679433,
  ↪ -1841180251, -1964177846, -2053567771, -2107820683, -2126008198, -2107819147, -2053564571, -1964173110,
  ↪ -1841173979, -1686671881, -1503310399, -1294226814, -1062998659, -813582285, -550245340, -277493485,
  ↪ 6319, 277506029, 550257500, 813593997, 1063009603, 1294236798, 1503319359, 1686679561, 1841180251,
  ↪ 1964177846, 2053567899, 2107820683, 2126008198, 2107819019, 2053564571, 1964172982, 1841173851,
  ↪ 1686671753, 1503310271, 1294226686, 1062998467, 813582157, 550245148, 277493325, -6505, -277506221,
  ↪ -550257692, -813594125, -1063009731, -1294236926, -1503319487, -1686679689, -1841180379, -1964177974,
  ↪ -2053567899, -2107820811, -2126008198, -2107819019, -2053564571, -1964172982, -1841173851, -1686671625,
  ↪ -1503310143, -1294226558, -1062998339, -813581965, -550244956, -277493133, 6691, 277506381, 550257884,
  ↪ 813594317, 1063009923, 1294237182, 1503319615, 1686679817, 1841180507, 1964178102, 2053568027,
  ↪ 2107820811, 2126008198, 2107819019, 2053564443, 1964172854, 1841173723, 1686671497, 1503310015,
  ↪ 1294226302, 1062998147, 813581773, 550244764, 277492941, -6876, -277506573, -550258076, -813594509,
  ↪ -1063010051, -1294237310, -1503319743, -1686679945, -1841180635, -1964178102, -2053568027, -2107820811,
  ↪ -2126008198, -2107819019, -2053564443, -1964172726, -1841173595, -1686671497, -1503309887, -1294226174,
  ↪ -1062998019, -813581581, -550244636, -277492749, 7062, 277506765, 550258268, 813594637, 1063010243,
  ↪ 1294237438, 1503319871, 1686680073, 1841180635, 1964178230, 2053568027, 2107820811, 2126008198,
  ↪ 2107819019, 2053564315, 1964172726, 1841173467, 1686671369, 1503309759, 1294226046, 1062997827,
  ↪ 813581453, 550244444, 277492589, -7248, -277506957, -550258460, -813594829, -1063010435, -1294237566,
  ↪ -1503319999, -1686680201, -1841180763, -1964178230, -2053568155, -2107820811, -2126008198, -2107818891,
  ↪ -2053564315, -1964172598, -1841173467, -1686671241, -1503309631, -1294225918, -1062997699, -813581261,
  ↪ -550244252, -277492397, 7434, 277507149, 550258588, 813595021, 1063010563, 1294237694, 1503320127,
  ↪ 1686680201, 1841180891, 1964178358, 2053568155, 2107820939, 2126008198, 2107818891, 2053564315,
  ↪ 1964172598, 1841173339, 1686671113, 1503309503};
// -6db 4/5KHz 48KHz sinewaves - intermodulation test
const int32_t im4k5k_m6dB_48[1024] = {0, 1190523165, 1967042101, 2065749272, 1466758218, 396719369,
  ↪ -759249949, -1601426338, -1859774933, -1484644677, -651982717, 314985730, 1073741631, 1388727394,
  ↪ 1207792408, 662838586, -45, -527684994, -759250333, -677022646, -393017066, -81733807, 107267448,
  ↪ 116782205, 674, -116781093, -107266968, 81733575, 393016170, 677021430, 759249181, 527684258, -40,
  ↪ -662838010, -1207791512, -1388726370, -1073740928, -314985410, 651982525, 1484644165, 1859774293,
  ↪ 1601425826, 759249629, -396719401, -1466758090, -2065749144, -1967041845, -1190523037, 153, 1190523293,
  ↪ 1967042229, 2065749528, 1466758346, 396719305, -759250269, -1601426978, -1859775573, -1484645189,
  ↪ -651982909, 314986018, 1073742399, 1388728418, 1207793432, 662839162, -130, -527685730, -759251485,
  ↪ -677023862, -393017962, -81734031, 107267984, 116783325, 2022, -116779981, -107266432, 81733351,
  ↪ 393015242, 677020150, 759247965, 527683522, -125, -662837498, -1207790488, -1388725474, -1073740160,
  ↪ -314985154, 651982333, 1484643525, 1859773653, 1601425186, 759249309, -396719465, -1466757962,
  ↪ -2065748888, -1967041589, -1190522781, 307, 1190523549, 1967042485, 2065749784, 1466758474, 396719241,
  ↪ -759250589, -1601427490, -1859776341, -1484645829, -651983101, 314986274, 1073743167, 1388729442,
  ↪ 1207794328, 662839674, -151, -527686466, -759252701, -677025078, -393018794, -81734263, 107268520,
  ↪ 116784437, 3370, -116778805, -107265896, 81733119, 393014346, 677018934, 759246813, 527682818, -210,
```

```
↪ -662836922, -1207789592, -1388724450, -1073739456, -314984866, 651982141, 1484643013, 1859773013,
↪ 1601424674, 759248925, -396719497, -1466757706, -2065748632, -1967041461, -1190522653, 460, 1190523677,
↪ 1967042741, 2065750040, 1466758602, 396719177, -759250909, -1601428002, -1859776981, -1484646341,
↪ -651983293, 314986594, 1073743935, 1388730466, 1207795352, 662840250, -237, -527687170, -759253853,
↪ -677026422, -393019690, -81734487, 107269064, 116785581, 4718, -116777685, -107265352, 81732887,
↪ 393013450, 677017654, 759245661, 527682082, -296, -662836410, -1207788696, -1388723426, -1073738688,
↪ -314984610, 651981949, 1484642501, 1859772245, 1601424034, 759248605, -396719625, -1466757578,
↪ -2065748376, -1967041205, -1190522397, 614, 1190523933, 1967042997, 2065750168, 1466758730, 396719145,
↪ -759251229, -1601428642, -1859777621, -1484646853, -651983549, 314986882, 1073744703, 1388731490,
↪ 1207796248, 662840826, -322, -527687906, -759255005, -677027638, -393020586, -81734719, 107269600,
↪ 116786693, 6067, -116776573, -107264816, 81732663, 393012554, 677016438, 759244445, 527681378, -381,
↪ -662835834, -1207787672, -1388722402, -1073737920, -314984322, 651981693, 1484641989, 1859771605,
↪ 1601423522, 759248285, -396719657, -1466757450, -2065748120, -1967040949, -1190522269, 768, 1190524061,
↪ 1967043253, 2065750424, 1466758858, 396719049, -759251613, -1601429154, -1859778389, -1484647493,
↪ -651983741, 314987138, 1073745471, 1388732514, 1207797144, 662841338, -407, -527688642, -759256221,
↪ -677028854, -393021514, -81734951, 107270136, 116787805, 7415, -116775461, -107264280, 81732431,
↪ 393011722, 677015158, 759243293, 527680642, -466, -662835258, -1207786776, -1388721378, -1073737216,
↪ -314983970, 651981501, 1484641349, 1859770965, 1601422882, 759247965, -396719721, -1466757322,
↪ -2065747992, -1967040693, -1190522141, 921, 1190524189, 1967043381, 2065750680, 1466759114, 396719017,
↪ -759251933, -1601429794, -1859779029, -1484648005, -651983933, 314987458, 1073746239, 1388733538,
↪ 1207798168, 662841914, -493, -527689378, -759257373, -677030070, -393022410, -81735175, 107270680,
↪ 116788957, 8763, -116774349, -107263736, 81732207, 393010826, 677013942, 759242141, 527679874, -488,
↪ -662834746, -1207785752, -1388720354, -1073736448, -314983714, 651981309, 1484640837, 1859770197,
↪ 1601422370, 759247645, -396719753, -1466757194, -2065747736, -1967040565, -1190521885, 1075, 1190524445,
↪  1967043637, 2065750936, 1466759242, 396718953, -759252253, -1601430306, -1859779669, -1484648517,
↪ -651984125, 314987746, 1073747007, 1388734562, 1207799064, 662842426, -578, -527690082, -759258525,
↪ -677031350, -393023306, -81735407, 107271152, 116790069, 10111, -116773229, -107263200, 81731975,
↪ 393009898, 677012662, 759240925, 527679170, -573, -662834170, -1207784856, -1388719330, -1073735680,
↪ -314983426, 651981117, 1484640325, 1859769557, 1601421730, 759247325, -396719817, -1466757066,
↪ -2065747480, -1967040309, -1190521757, 1229, 1190524573, 1967043765, 2065751192, 1466759370, 396718889,
↪ -759252573, -1601430946, -1859780309, -1484649157, -651984317, 314988002, 1073747647, 1388735586,
↪ 1207800088, 662843002, -663, -527690850, -759259741, -677032566, -393024202, -81735631, 107271688,
↪ 116791181, 11460, -116772053, -107262728, 81731751, 393009002, 677011446, 759239773, 527678434, -658,
↪ -662833594, -1207783832, -1388718306, -1073734912, -314983138, 651980925, 1484639685, 1859768917,
↪ 1601421218, 759247005, -396719849, -1466756938, -2065747224, -1967040053, -1190521629, 1382, 1190524829,
↪  1967044021, 2065751320, 1466759498, 396718825, -759252893, -1601431458, -1859781077, -1484649669,
↪ -651984509, 314988322, 1073748415, 1388736610, 1207800984, 662843578, -685, -527691554, -759260893,
↪ -677033782, -393025098, -81735863, 107272232, 116792325, 12808, -116770941, -107262184, 81731519,
↪ 393008106, 677010230, 759238621, 527677730, -744, -662833082, -1207782936, -1388717282, -1073734208,
↪ -314982882, 651980733, 1484639173, 1859768277, 1601420578, 759246685, -396719977, -1466756682,
↪ -2065746968, -1967039925, -1190521373, 1536, 1190524957, 1967044277, 2065751576, 1466759626, 396718793,
↪ -759253213, -1601432098, -1859781717, -1484650181, -651984701, 314988578, 1073749183, 1388737506,
↪ 1207802008, 662844090, -770, -527692258, -759262045, -677035062, -393025930, -81736087, 107272768,
↪ 116793445, 14156, -116769829, -107261648, 81731287, 393007210, 677008950, 759237405, 527676994, -829,
↪ -662832506, -1207781912, -1388716258, -1073733440, -314982594, 651980541, 1484638661, 1859767509,
↪ 1601420066, 759246301, -396720009, -1466756554, -2065746840, -1967039669, -1190521245, 1690, 1190525085,
↪  1967044533, 2065751832, 1466759754, 396718697, -759253533, -1601432610, -1859782357, -1484650821,
↪ -651984957, 314988866, 1073749951, 1388738658, 1207802904, 662844666, -855, -527693026, -759263261,
↪ -677036342, -393026858, -81736319, 107273304, 116794557, 15504, -116768709, -107261112, 81731063,
↪ 393006314, 677007734, 759236253, 527676290, -915, -662831994, -1207781016, -1388715234, -1073732672,
↪ -314982242, 651980285, 1484638021, 1859766869, 1601419426, 759245981, -396720073, -1466756426,
↪ -2065746584, -1967039413, -1190521117, 1843, 1190525213, 1967044789, 2065752088, 1466759882, 396718665,
↪ -759253853, -1601433122, -1859783125, -1484651333, -651985149, 314989186, 1073750719, 1388739682,
↪ 1207803800, 662845242, -941, -527693730, -759264413, -677037558, -393027754, -81736551, 107273848,
↪ 116795701, 16853, -116767597, -107260568, 81730831, 393005418, 677006454, 759235101, 527675490, -1000,
↪ -662831418, -1207780120, -1388714210, -1073731904, -314981986, 651980093, 1484637509, 1859766229,
↪ 1601418914, 759245661, -396720105, -1466756298, -2065746328, -1967039157, -1190520861, 1997, 1190525469,
↪  1967044917, 2065752344, 1466760010, 396718601, -759254237, -1601433762, -1859783765, -1484651845,
↪ -651985341, 314989442, 1073751487, 1388740578, 1207804824, 662845754, -1026, -527694466, -759265565,
↪ -677038774, -393028650, -81736775, 107274384, 116796813, 18201, -116766485, -107260032, 81730607,
↪ 393004554, 677005238, 759233885, 527674786, -1021, -662830842, -1207779096, -1388713186, -1073731200,
↪ -314981698, 651979901, 1484636997, 1859765461, 1601418274, 759245341, -396720169, -1466756170,
↪ -2065746072, -1967038901, -1190520733, 2151, 1190525597, 1967045173, 2065752472, 1466760138, 396718537,
↪ -759254557, -1601434402, -1859784405, -1484652357, -651985533, 314989730, 1073752255, 1388741602,
↪ 1207805720, 662846330, -1111, -527695202, -759266781, -677039990, -393029546, -81737007, 107274920,
↪ 116797933, 19549, -116765309, -107259496, 81730375, 393003658, 677003958, 759232733, 527674050, -1107,
↪ -662830330, -1207778200, -1388712290, -1073730432, -314981442, 651979709, 1484636357, 1859764821,
↪ 1601417762, 759245021, -396720201, -1466756042, -2065745944, -1967038773, -1190520477, 2304, 1190525853,
↪  1967045429, 2065752728, 1466760266, 396718473, -759254877, -1601434914, -1859785045, -1484652997,
↪ -651985725, 314990050, 1073752895, 1388742626, 1207806744, 662846842, -1197, -527695938, -759267933,
↪ -677041270, -393030442, -81737231, 107275400, 116799077, 20898, -116764197, -107259016, 81730151,
↪ 393002762, 677002742, 759231581, 527673346, -1192, -662829754, -1207777176, -1388711138, -1073729664,
↪ -314981154, 651979517, 1484635845, 1859764181, 1601417122, 759244701, -396720329, -1466755914,
↪ -2065745688, -1967038517, -1190520349, 2458, 1190525981, 1967045557, 2065752984, 1466760394, 396718441,
↪ -759255197, -1601435426, -1859785685, -1484653509, -651985917, 314990306, 1073753663, 1388743650,
```

```
  ↪ 1207807640, 662847418, -1282, -527696642, -759269085, -677042550, -393031338, -81737463, 107275936,
  ↪ 116800189, 22246, -116763077, -107258480, 81729919, 393001866, 677001526, 759230365, 527672642, -1277,
  ↪ -662829242, -1207776280, -1388710242, -1073728896, -314980866, 651979325, 1484635333, 1859763541,
  ↪ 1601416610, 759244381, -396720361, -1466755786, -2065745432, -1967038389, -1190520221, 2612, 1190526109,
  ↪  1967045813, 2065753240, 1466760650, 396718345, -759255517, -1601436066, -1859786453, -1484654021,
  ↪ -651986109, 314990594, 1073754431, 1388744674, 1207808664, 662847994, -1304, -527697378, -759270301,
  ↪ -677043766, -393032202, -81737687, 107276472, 116801301, 23594, -116761965, -107257944, 81729687,
  ↪ 393000970, 677000246, 759229213, 527671906, -1363, -662828666, -1207775256, -1388709218, -1073728192,
  ↪ -314980546, 651979133, 1484634821, 1859762901, 1601415970, 759243997, -396720425, -1466755658,
  ↪ -2065745176, -1967038133, -1190519965, 2765, 1190526365, 1967046069, 2065753368, 1466760778, 396718313,
  ↪ -759255837, -1601436578, -1859787093, -1484654661, -651986301, 314990882, 1073755199, 1388745698,
  ↪ 1207809560, 662848506, -1389, -527698114, -759271453, -677044982, -393033098, -81737919, 107277016,
  ↪ 116802445, 24942, -116760853, -107257408, 81729463, 393000074, 676999030, 759228061, 527671202, -1448,
  ↪ -662828090, -1207774360, -1388708194, -1073727424, -314980258, 651978941, 1484634181, 1859762133,
  ↪ 1601415458, 759243677, -396720457, -1466755530, -2065744920, -1967037877, -1190519837, 2919, 1190526493,
  ↪  1967046325, 2065753624, 1466760906, 396718249, -759256157, -1601437218, -1859787733, -1484655173,
  ↪ -651986557, 314991170, 1073755967, 1388746722, 1207810456, 662849082, -1474, -527698818, -759272605,
  ↪ -677046262, -393033994, -81738151, 107277552, 116803565, 26291, -116759741, -107256864, 81729231,
  ↪ 392999146, 676997814, 759226845, 527670402, -1533, -662827578, -1207773464, -1388707170, -1073726656,
  ↪ -314979970, 651978685, 1484633669, 1859761493, 1601414818, 759243357, -396720521, -1466755402,
  ↪ -2065744792, -1967037621, -1190519581, 3073, 1190526621, 1967046453, 2065753880, 1466761034, 396718185,
  ↪ -759256541, -1601437858, -1859788501, -1484655685, -651986749, 314991458, 1073756735, 1388747746,
  ↪ 1207811480, 662849658, -1560, -527699586, -759273821, -677047478, -393034890, -81738375, 107278088,
  ↪ 116804677, 27639, -116758557, -107256328, 81729007, 392998314, 676996534, 759225693, 527669698, -1619,
  ↪ -662827002, -1207772440, -1388706146, -1073725952, -314979714, 651978493, 1484633157, 1859760853,
  ↪ 1601414306, 759243037, -396720553, -1466755146, -2065744536, -1967037365, -1190519453, 3226, 1190526877,
  ↪  1967046709, 2065754136, 1466761162, 396718121, -759256861, -1601438370, -1859789141, -1484656325,
  ↪ -651986941, 314991714, 1073757503, 1388748770, 1207812376};

int main(void)
{
    unsafe {
        dsp_ds3_return_code_t return_code = DSP_DS3_NO_ERROR;

        // Input data for both channels
        const int32_t * unsafe input_data[NUM_CHANNELS] = {s1k_0db_48, im4k5k_m6dB_48};

        // Output samples
        int32_t output_data[NUM_CHANNELS];

        // DS3 instances variables
        // -----------------------
        // State and Control structures (one for each channel)
        int                 dsp_ds3_delay[NUM_CHANNELS][DSP_DS3_N_COEFS<<1];
        dsp_ds3_ctrl_t      dsp_ds3_ctrl[NUM_CHANNELS];

        //Init DS3
        for (int i=0; i<NUM_CHANNELS; i++) {

            printf("Init ds3 channel %d\n", i);
            // Process init
            // ------------
            // Set delay line base to ctrl structure
            dsp_ds3_ctrl[i].delay_base = dsp_ds3_delay[i];

            // Init instance
            if (dsp_ds3_init(&dsp_ds3_ctrl[i]) != DSP_DS3_NO_ERROR) {
                printf("Error on init\n");
                return_code = DSP_DS3_ERROR;
            }

            // Sync (i.e. clear data)
            // ----
            if (dsp_ds3_sync(&dsp_ds3_ctrl[i]) != DSP_DS3_NO_ERROR) {
                printf("Error on sync\n");
                return_code = DSP_DS3_ERROR;
            }

        }

        for (int s=0; s<NUM_OUTPUT_SAMPLES; s++) {
            for (int i=0; i<NUM_CHANNELS; i++) {
                // Set input and output data pointers for the DS3
                dsp_ds3_ctrl[i].in_data            = (int *)input_data[i];
                dsp_ds3_ctrl[i].out_data           = (int *)&output_data[i];
```

```
                // Do the sample rate conversion on three input samples
                if (dsp_ds3_proc(&dsp_ds3_ctrl[i]) != DSP_DS3_NO_ERROR) {
                    printf("Error on ds3 process\n");
                    return_code = DSP_DS3_ERROR;
                }
                //for(int j=0; j<3; j++) printf("in = %d\n", *(input_data[i] + j));
                input_data[i] += 3; // Move input pointer on by 3
                printf("%d\n", output_data[i]);
            }
        }
        return (int)return_code;
    }
}
```

B.10 Oversample by 3 Function

```
// Copyright (c) 2016, XMOS Ltd, All rights reserved
// XMOS DSP Library - Upsample by factor of 3 example
// Uses Signed 32b Integer format

// Include files
#include <stdio.h>
#include <xs1.h>
#include <dsp.h>
#include <stdint.h>

#define NUM_CHANNELS  2
#define NUM_OUTPUT_SAMPLES  384

// 0db 1KHz 16KHz sinewave - pure tone test
const int32_t s1k_0db_16[1024] = {0, 820984395, 1516981307, 1982031665, 2145335552, 1982031537, 1516981179,
  ↪ 820984203, -187, -820984587, -1516981435, -1982031665, -2145335552, -1982031537, -1516981051,
  ↪ -820984075, 375, 820984715, 1516981563, 1982031793, 2145335552, 1982031409, 1516980923, 820983883, -562,
  ↪  -820984907, -1516981691, -1982031793, -2145335552, -1982031409, -1516980795, -820983691, 750,
  ↪ 820985099, 1516981819, 1982031921, 2145335552, 1982031281, 1516980667, 820983499, -937, -820985291,
  ↪ -1516982075, -1982032049, -2145335552, -1982031153, -1516980539, -820983371, 1125, 820985419,
  ↪ 1516982203, 1982032049, 2145335552, 1982031153, 1516980411, 820983179, -1312, -820985611, -1516982331,
  ↪ -1982032177, -2145335552, -1982031025, -1516980283, -820982987, 1500, 820985803, 1516982459, 1982032177,
  ↪  2145335552, 1982031025, 1516980155, 820982859, -1687, -820985931, -1516982587, -1982032305,
  ↪ -2145335552, -1982030897, -1516980027, -820982667, 1875, 820986123, 1516982715, 1982032305, 2145335552,
  ↪ 1982030897, 1516979899, 820982475, -2063, -820986315, -1516982843, -1982032433, -2145335552,
  ↪ -1982030769, -1516979771, -820982283, 2250, 820986443, 1516982971, 1982032433, 2145335552, 1982030641,
  ↪ 1516979643, 820982155, -2438, -820986635, -1516983099, -1982032561, -2145335552, -1982030641,
  ↪ -1516979515, -820981963, 2625, 820986827, 1516983227, 1982032689, 2145335552, 1982030513, 1516979387,
  ↪ 820981771, -2813, -820987019, -1516983355, -1982032689, -2145335552, -1982030513, -1516979259,
  ↪ -820981643, 3000, 820987147, 1516983483, 1982032817, 2145335552, 1982030385, 1516979131, 820981451,
  ↪ -3188, -820987339, -1516983611, -1982032817, -2145335552, -1982030385, -1516979003, -820981259, 3375,
  ↪ 820987531, 1516983739, 1982032945, 2145335552, 1982030257, 1516978875, 820981131, -3563, -820987659,
  ↪ -1516983867, -1982032945, -2145335552, -1982030257, -1516978747, -820980939, 3751, 820987851,
  ↪ 1516983995, 1982033073, 2145335552, 1982030129, 1516978619, 820980747, -3938, -820988043, -1516984123,
  ↪ -1982033201, -2145335552, -1982030001, -1516978491, -820980555, 4126, 820988235, 1516984251, 1982033201,
  ↪  2145335552, 1982030001, 1516978363, 820980427, -4313, -820988363, -1516984379, -1982033329,
  ↪ -2145335552, -1982029873, -1516978107, -820980235, 4501, 820988555, 1516984507, 1982033329, 2145335552,
  ↪ 1982029873, 1516977979, 820980043, -4688, -820988747, -1516984635, -1982033457, -2145335552,
  ↪ -1982029745, -1516977851, -820979915, 4876, 820988875, 1516984763, 1982033457, 2145335552, 1982029745,
  ↪ 1516977723, 820979723, -5063, -820989067, -1516984891, -1982033585, -2145335552, -1982029617,
  ↪ -1516977595, -820979531, 5251, 820989259, 1516985019, 1982033585, 2145335552, 1982029489, 1516977467,
  ↪ 820979339, -5438, -820989451, -1516985147, -1982033713, -2145335552, -1982029489, -1516977339,
  ↪ -820979211, 5626, 820989579, 1516985275, 1982033841, 2145335552, 1982029361, 1516977211, 820979019,
  ↪ -5814, -820989771, -1516985403, -1982033841, -2145335552, -1982029361, -1516977083, -820978827, 6001,
  ↪ 820989963, 1516985531, 1982033969, 2145335552, 1982029233, 1516976955, 820978699, -6189, -820990091,
  ↪ -1516985787, -1982033969, -2145335552, -1982029233, -1516976827, -820978507, 6376, 820990283,
  ↪ 1516985915, 1982034097, 2145335552, 1982029105, 1516976699, 820978315, -6564, -820990475, -1516986043,
  ↪ -1982034097, -2145335552, -1982029105, -1516976571, -820978187, 6751, 820990603, 1516986171, 1982034225,
  ↪  2145335552, 1982028977, 1516976443, 820977995, -6939, -820990795, -1516986299, -1982034353,
  ↪ -2145335552, -1982028849, -1516976315, -820977803, 7126, 820990987, 1516986427, 1982034353, 2145335552,
  ↪ 1982028849, 1516976187, 820977611, -7314, -820991179, -1516986555, -1982034481, -2145335552,
  ↪ -1982028721, -1516976059, -820977483, 7502, 820991307, 1516986683, 1982034481, 2145335552, 1982028721,
  ↪ 1516975931, 820977291, -7689, -820991499, -1516986811, -1982034609, -2145335552, -1982028593,
  ↪ -1516975803, -820977099, 7877, 820991691, 1516986939, 1982034609, 2145335552, 1982028593, 1516975675,
  ↪ 820976971, -8064, -820991819, -1516987067, -1982034737, -2145335552, -1982028465, -1516975547,
  ↪ -820976779, 8252, 820992011, 1516987195, 1982034737, 2145335552, 1982028465, 1516975419, 820976587,
  ↪ -8439, -820992203, -1516987323, -1982034865, -2145335552, -1982028337, -1516975291, -820976395, 8627,
  ↪ 820992395, 1516987451, 1982034993, 2145335552, 1982028209, 1516975163, 820976267, -8814, -820992523,
  ↪ -1516987579, -1982034993, -2145335552, -1982028209, -1516975035, -820976075, 9002, 820992715,
  ↪ 1516987707, 1982035121, 2145335552, 1982028081, 1516974907, 820975883, -9190, -820992907, -1516987835,
  ↪ -1982035121, -2145335552, -1982028081, -1516974779, -820975755, 9377, 820993035, 1516987963, 1982035249,
  ↪  2145335552, 1982027953, 1516974523, 820975563, -9565, -820993227, -1516988091, -1982035249,
  ↪ -2145335552, -1982027953, -1516974395, -820975371, 9752, 820993419, 1516988219, 1982035377, 2145335552,
  ↪ 1982027825, 1516974267, 820975179, -9940, -820993547, -1516988347, -1982035377, -2145335552,
  ↪ -1982027697, -1516974139, -820975051, 10127, 820993739, 1516988475, 1982035505, 2145335552, 1982027697,
  ↪ 1516974011, 820974859, -10315, -820993931, -1516988603, -1982035633, -2145335552, -1982027569,
  ↪ -1516973883, -820974667, 10502, 820994123, 1516988731, 1982035633, 2145335552, 1982027569, 1516973755,
  ↪ 820974539, -10690, -820994251, -1516988859, -1982035761, -2145335552, -1982027441, -1516973627,
  ↪ -820974347, 10877, 820994443, 1516988987, 1982035761, 2145335552, 1982027441, 1516973499, 820974155,
  ↪ -11065, -820994635, -1516989115, -1982035889, -2145335552, -1982027313, -1516973371, -820974027, 11253,
  ↪ 820994763, 1516989243, 1982035889, 2145335552, 1982027313, 1516973243, 820973835, -11440, -820994955,
  ↪ -1516989499, -1982036017, -2145335552, -1982027185, -1516973115, -820973643, 11628, 820995147,
  ↪ 1516989627, 1982036145, 2145335552, 1982027057, 1516972987, 820973451, -11815, -820995339, -1516989755,
```

```
  ↪ -1982036145, -2145335552, -1982027057, -1516972859, -820973323, 12003, 820995467, 1516989883,
  ↪ 1982036273, 2145335552, 1982026929, 1516972731, 820973131, -12190, -820995659, -1516990011, -1982036273,
  ↪  -2145335552, -1982026929, -1516972603, -820972939, 12378, 820995851, 1516990139, 1982036401,
  ↪ 2145335552, 1982026801, 1516972475, 820972811, -12565, -820995979, -1516990267, -1982036401,
  ↪ -2145335552, -1982026801, -1516972347, -820972619, 12753, 820996171, 1516990395, 1982036529, 2145335552,
  ↪  1982026673, 1516972219, 820972427, -12941, -820996363, -1516990523, -1982036529, -2145335552,
  ↪ -1982026673, -1516972091, -820972235, 13128, 820996555, 1516990651, 1982036657, 2145335552, 1982026545,
  ↪ 1516971963, 820972107, -13316, -820996683, -1516990779, -1982036785, -2145335552, -1982026417,
  ↪ -1516971835, -820971915, 13503, 820996875, 1516990907, 1982036785, 2145335552, 1982026417, 1516971707,
  ↪ 820971723, -13691, -820997067, -1516991035, -1982036913, -2145335552, -1982026289, -1516971579,
  ↪ -820971595, 13878, 820997195, 1516991163, 1982036913, 2145335552, 1982026289, 1516971451, 820971403,
  ↪ -14066, -820997387, -1516991291, -1982037041, -2145335552, -1982026161, -1516971323, -820971211, 14253,
  ↪ 820997579, 1516991419, 1982037041, 2145335552, 1982026161, 1516971195, 820971083, -14441, -820997707,
  ↪ -1516991547, -1982037169, -2145335552, -1982026033, -1516971067, -820970891, 14628, 820997899,
  ↪ 1516991675, 1982037169, 2145335552, 1982025905, 1516970811, 820970699, -14816, -820998091, -1516991803,
  ↪ -1982037297, -2145335552, -1982025905, -1516970683, -820970507, 15004, 820998283, 1516991931,
  ↪ 1982037425, 2145335552, 1982025777, 1516970555, 820970379, -15191, -820998411, -1516992059, -1982037425,
  ↪  -2145335552, -1982025777, -1516970427, -820970187, 15379, 820998603, 1516992187, 1982037553,
  ↪ 2145335552, 1982025649, 1516970299, 820969995, -15566, -820998795, -1516992315, -1982037553,
  ↪ -2145335552, -1982025649, -1516970171, -820969867, 15754, 820998923, 1516992443, 1982037681, 2145335552,
  ↪  1982025521, 1516970043, 820969675, -15941, -820999115, -1516992571, -1982037681, -2145335552,
  ↪ -1982025521, -1516969915, -820969483, 16129, 820999307, 1516992699, 1982037809, 2145335552, 1982025393,
  ↪ 1516969787, 820969291, -16316, -820999499, -1516992827, -1982037937, -2145335552, -1982025265,
  ↪ -1516969659, -820969163, 16504, 820999627, 1516993083, 1982037937, 2145335552, 1982025265, 1516969531,
  ↪ 820968971, -16692, -820999819, -1516993211, -1982038065, -2145335552, -1982025137, -1516969403,
  ↪ -820968779, 16879, 821000011, 1516993339, 1982038065, 2145335552, 1982025137, 1516969275, 820968651,
  ↪ -17067, -821000139, -1516993467, -1982038193, -2145335552, -1982025009, -1516969147, -820968459, 17254,
  ↪ 821000331, 1516993595, 1982038193, 2145335552, 1982025009, 1516969019, 820968267, -17442, -821000523,
  ↪ -1516993723, -1982038321, -2145335552, -1982024881, -1516968891, -820968075, 17629, 821000651,
  ↪ 1516993851, 1982038321, 2145335552, 1982024753, 1516968763, 820967947, -17817, -821000843, -1516993979,
  ↪ -1982038449, -2145335552, -1982024753, -1516968635, -820967755, 18004, 821001035, 1516994107,
  ↪ 1982038577, 2145335552, 1982024625, 1516968507, 820967563, -18192, -821001227, -1516994235, -1982038577,
  ↪  -2145335552, -1982024625, -1516968379, -820967435, 18380, 821001355, 1516994363, 1982038705,
  ↪ 2145335552, 1982024497, 1516968251, 820967243, -18567, -821001547, -1516994491, -1982038705,
  ↪ -2145335552, -1982024497, -1516968123, -820967051, 18755, 821001739, 1516994619, 1982038833, 2145335552,
  ↪  1982024369, 1516967995, 820966923, -18942, -821001867, -1516994747, -1982038833, -2145335552,
  ↪ -1982024369, -1516967867, -820966731, 19130, 821002059, 1516994875, 1982038961, 2145335552, 1982024241,
  ↪ 1516967739, 820966539, -19317, -821002251, -1516995003, -1982039089, -2145335552, -1982024113,
  ↪ -1516967611, -820966347, 19505, 821002443, 1516995131, 1982039089, 2145335552, 1982024113, 1516967483,
  ↪ 820966219, -19692, -821002571, -1516995259, -1982039217, -2145335552, -1982023985, -1516967355,
  ↪ -820966027, 19880, 821002763, 1516995387, 1982039217, 2145335552, 1982023985, 1516967099, 820965835,
  ↪ -20067, -821002955, -1516995515, -1982039345, -2145335552, -1982023857, -1516966971, -820965707, 20255,
  ↪ 821003083, 1516995643, 1982039345, 2145335552, 1982023857, 1516966843, 820965515, -20443, -821003275,
  ↪ -1516995771, -1982039473, -2145335552, -1982023729, -1516966715, -820965323, 20630, 821003467,
  ↪ 1516995899, 1982039473, 2145335552, 1982023729, 1516966587, 820965131, -20818, -821003659, -1516996027,
  ↪ -1982039601, -2145335552, -1982023601, -1516966459, -820965003, 21005, 821003787, 1516996155,
  ↪ 1982039729, 2145335552, 1982023473, 1516966331, 820964811, -21193, -821003979, -1516996283, -1982039729,
  ↪  -2145335552, -1982023473, -1516966203, -820964619, 21380, 821004171, 1516996411, 1982039857,
  ↪ 2145335552, 1982023345, 1516966075, 820964491, -21568, -821004299, -1516996539, -1982039857,
  ↪ -2145335552, -1982023345, -1516965947, -820964299, 21755, 821004491, 1516996795, 1982039985, 2145335552,
  ↪  1982023217, 1516965819, 820964107, -21943, -821004683, -1516996923, -1982039985, -2145335552,
  ↪ -1982023217, -1516965691, -820963979, 22131, 821004811, 1516997051, 1982040113, 2145335552, 1982023089,
  ↪ 1516965563, 820963787, -22318, -821005003, -1516997179, -1982040113, -2145335552, -1982022961,
  ↪ -1516965435, -820963595, 22506, 821005195, 1516997307, 1982040241, 2145335552, 1982022961, 1516965307,
  ↪ 820963403, -22693, -821005387, -1516997435, -1982040369, -2145335552, -1982022833, -1516965179,
  ↪ -820963275, 22881, 821005515, 1516997563, 1982040369, 2145335552, 1982022833, 1516965051, 820963083,
  ↪ -23068, -821005707, -1516997691, -1982040497, -2145335552, -1982022705, -1516964923, -820962891, 23256,
  ↪ 821005899, 1516997819, 1982040497, 2145335552, 1982022705, 1516964795, 820962763, -23443, -821006027,
  ↪ -1516997947, -1982040625, -2145335552, -1982022577, -1516964667, -820962571, 23631, 821006219,
  ↪ 1516998075, 1982040625, 2145335552, 1982022577, 1516964539, 820962379, -23819, -821006411, -1516998203,
  ↪ -1982040753, -2145335552, -1982022449, -1516964411, -820962187};
// -6db 4/5KHz 16KHz sinewaves - intermodulation test
const int32_t im4k5k_m6dB_16[1024] = {0, 1751257718, -1832991325, 348346744, 1073741504, -1170152739,
  ↪ 314491426, 232757770, 349, -232758042, -314491874, 1170153379, -1073741631, -348347224, 1832991709,
  ↪ -1751257590, -494, 1751257846, -1832990941, 348346232, 1073741376, -1170152227, 314491042, 232757450,
  ↪ 1049, -232758362, -314492322, 1170154019, -1073741631, -348347672, 1832992221, -1751257462, -989,
  ↪ 1751257974, -1832990429, 348345752, 1073741248, -1170151587, 314490594, 232757178, 1749, -232758698,
  ↪ -314492706, 1170154659, -1073741759, -348348152, 1832992605, -1751257206, -1484, 1751258102,
  ↪ -1832990045, 348345304, 1073741184, -1170150947, 314490210, 232756858, 2449, -232758954, -314493154,
  ↪ 1170155299, -1073741887, -348348664, 1832992989, -1751257078, -1979, 1751258358, -1832989661, 348344824,
  ↪  1073741056, -1170150307, 314489762, 232756586, 3148, -232759290, -314493538, 1170155939, -1073742015,
  ↪ -348349144, 1832993373, -1751256950, -2474, 1751258486, -1832989149, 348344312, 1073740992, -1170149667,
  ↪  314489314, 232756266, 3848, -232759546, -314493986, 1170156579, -1073742143, -348349592, 1832993885,
  ↪ -1751256694, -2969, 1751258614, -1832988765, 348343832, 1073740864, -1170149027, 314488930, 232755930,
  ↪ 4548, -232759882, -314494434, 1170157219, -1073742143, -348350072, 1832994269, -1751256694, -3464,
```

```
↪ 1751258870, -1832988381, 348343384, 1073740736, -1170148387, 314488482, 232755674, 5248, -232760138,
↪ -314494818, 1170157731, -1073742271, -348350584, 1832994653, -1751256438, -3958, 1751258998,
↪ -1832987997, 348342904, 1073740672, -1170147747, 314488098, 232755338, 5947, -232760474, -314495266,
↪ 1170158371, -1073742399, -348351032, 1832995165, -1751256310, -4453, 1751259126, -1832987485, 348342392,
↪  1073740544, -1170147235, 314487650, 232755066, 6647, -232760794, -314495650, 1170159011, -1073742527,
↪ -348351512, 1832995549, -1751256182, -4948, 1751259254, -1832987101, 348341912, 1073740480, -1170146595,
↪  314487202, 232754746, 7347, -232761066, -314496098, 1170159651, -1073742655, -348351992, 1832995933,
↪ -1751255926, -5443, 1751259382, -1832986717, 348341464, 1073740352, -1170145955, 314486818, 232754474,
↪ 8046, -232761402, -314496546, 1170160291, -1073742655, -348352472, 1832996445, -1751255798, -5938,
↪ 1751259638, -1832986205, 348340984, 1073740224, -1170145315, 314486370, 232754154, 8746, -232761658,
↪ -314496930, 1170160931, -1073742783, -348352952, 1832996829, -1751255670, -6433, 1751259766,
↪ -1832985821, 348340472, 1073740160, -1170144675, 314485986, 232753818, 9446, -232761994, -314497378,
↪ 1170161571, -1073742911, -348353432, 1832997213, -1751255542, -6928, 1751259894, -1832985437, 348340024,
↪  1073740032, -1170144035, 314485538, 232753562, 10146, -232762250, -314497762, 1170162083, -1073743039,
↪ -348353912, 1832997597, -1751255414, -7423, 1751260022, -1832984925, 348339544, 1073739968, -1170143395,
↪  314485090, 232753226, 10845, -232762586, -314498210, 1170162723, -1073743167, -348354392, 1832998109,
↪ -1751255286, -7917, 1751260278, -1832984541, 348339064, 1073739840, -1170142755, 314484706, 232752970,
↪ 11545, -232762906, -314498658, 1170163363, -1073743167, -348354872, 1832998493, -1751255030, -8412,
↪ 1751260278, -1832984157, 348338552, 1073739712, -1170142243, 314484258, 232752634, 12245, -232763178,
↪ -314499042, 1170164003, -1073743295, -348355352, 1832998877, -1751254902, -8907, 1751260534,
↪ -1832983773, 348338104, 1073739648, -1170141475, 314483874, 232752378, 12945, -232763498, -314499490,
↪ 1170164643, -1073743423, -348355832, 1832999389, -1751254774, -9402, 1751260662, -1832983261, 348337624,
↪  1073739520, -1170140963, 314483426, 232752042, 13644, -232763770, -314499874, 1170165283, -1073743551,
↪ -348356312, 1832999773, -1751254518, -9897, 1751260790, -1832982877, 348337144, 1073739456, -1170140323,
↪  314482978, 232751722, 14344, -232764090, -314500322, 1170165923, -1073743679, -348356792, 1833000157,
↪ -1751254390, -10392, 1751261046, -1832982493, 348336632, 1073739328, -1170139683, 314482594, 232751450,
↪ 15044, -232764362, -314500770, 1170166563, -1073743679, -348357272, 1833000669, -1751254262, -10887,
↪ 1751261174, -1832981981, 348336184, 1073739200, -1170139043, 314482146, 232751130, 15744, -232764682,
↪ -314501154, 1170167075, -1073743807, -348357752, 1833001053, -1751254134, -11382, 1751261302,
↪ -1832981597, 348335704, 1073739136, -1170138403, 314481762, 232750858, 16443, -232765018, -314501602,
↪ 1170167843, -1073743935, -348358232, 1833001437, -1751254006, -11876, 1751261430, -1832981213,
↪ 348335224, 1073739008, -1170137763, 314481314, 232750538, 17143, -232765274, -314501986, 1170168355,
↪ -1073744063, -348358712, 1833001821, -1751253750, -12371, 1751261558, -1832980701, 348334712,
↪ 1073738944, -1170137251, 314480866, 232750266, 17843, -232765610, -314502434, 1170168995, -1073744191,
↪ -348359192, 1833002333, -1751253622, -12866, 1751261814, -1832980317, 348334264, 1073738816,
↪ -1170136483, 314480482, 232749946, 18543, -232765866, -314502882, 1170169635, -1073744191, -348359672,
↪ 1833002717, -1751253494, -13361, 1751261942, -1832979933, 348333784, 1073738688, -1170135971, 314480034,
↪  232749610, 19242, -232766202, -314503266, 1170170275, -1073744319, -348360120, 1833003101, -1751253366,
↪  -13856, 1751262070, -1832979549, 348333304, 1073738624, -1170135331, 314479650, 232749338, 19942,
↪ -232766458, -314503714, 1170170915, -1073744447, -348360632, 1833003613, -1751253110, -14351,
↪ 1751262198, -1832979037, 348332792, 1073738496, -1170134691, 314479202, 232749018, 20642, -232766794,
↪ -314504098, 1170171555, -1073744575, -348361112, 1833003997, -1751253110, -14846, 1751262454,
↪ -1832978653, 348332344, 1073738432, -1170134051, 314478754, 232748746, 21342, -232767130, -314504546,
↪ 1170172067, -1073744703, -348361592, 1833004381, -1751252854, -15341, 1751262582, -1832978269,
↪ 348331864, 1073738304, -1170133411, 314478370, 232748426, 22041, -232767386, -314504994, 1170172835,
↪ -1073744703, -348362040, 1833004893, -1751252726, -15835, 1751262710, -1832977757, 348331384,
↪ 1073738176, -1170132771, 314477922, 232748154, 22741, -232767722, -314505378, 1170173347, -1073744831,
↪ -348362552, 1833005277, -1751252598, -16330, 1751262838, -1832977373, 348330904, 1073738112,
↪ -1170132259, 314477538, 232747834, 23441, -232767978, -314505826, 1170173987, -1073744959, -348363032,
↪ 1833005661, -1751252342, -16825, 1751262966, -1832976989, 348330424, 1073737984, -1170131491, 314477090,
↪  232747498, 24140, -232768314, -314506210, 1170174627, -1073745087, -348363512, 1833006045, -1751252214,
↪  -17320, 1751263222, -1832976477, 348329944, 1073737920, -1170130979, 314476642, 232747242, 24840,
↪ -232768570, -314506658, 1170175267, -1073745215, -348363960, 1833006557, -1751252086, -17815,
↪ 1751263350, -1832976093, 348329464, 1073737792, -1170130339, 314476258, 232746906, 25540, -232768906,
↪ -314507106, 1170175907, -1073745215, -348364472, 1833006941, -1751251958, -18310, 1751263478,
↪ -1832975709, 348328984, 1073737664, -1170129699, 314475810, 232746650, 26240, -232769226, -314507490,
↪ 1170176547, -1073745343, -348364952, 1833007325, -1751251830, -18805, 1751263606, -1832975325,
↪ 348328504, 1073737600, -1170129059, 314475426, 232746314, 26939, -232769498, -314507938, 1170177059,
↪ -1073745471, -348365432, 1833007837, -1751251574, -19299, 1751263734, -1832974813, 348328024,
↪ 1073737472, -1170128419, 314474978, 232746058, 27639, -232769818, -314508322, 1170177827, -1073745599,
↪ -348365880, 1833008221, -1751251446, -19794, 1751263990, -1832974429, 348327544, 1073737408,
↪ -1170127779, 314474530, 232745722, 28339, -232770090, -314508770, 1170178339, -1073745727, -348366392,
↪ 1833008605, -1751251318, -20289, 1751264118, -1832974045, 348327096, 1073737280, -1170127139, 314474146,
↪  232745402, 29039, -232770410, -314509154, 1170178979, -1073745727, -348366872, 1833009117, -1751251190,
↪  -20784, 1751264246, -1832973533, 348326584, 1073737152, -1170126627, 314473698, 232745130, 29738,
↪ -232770682, -314509602, 1170179619, -1073745855, -348367352, 1833009501, -1751250934, -21279,
↪ 1751264374, -1832973149, 348326104, 1073737088, -1170125987, 314473314, 232744810, 30438, -232771002,
↪ -314510050, 1170180259, -1073745983, -348367800, 1833009885, -1751250806, -21774, 1751264630,
↪ -1832972765, 348325624, 1073736960, -1170125347, 314472866, 232744538, 31138, -232771338, -314510434,
↪ 1170180899, -1073746111, -348368312, 1833010269, -1751250678, -22269, 1751264758, -1832972253,
↪ 348325176, 1073736896, -1170124707, 314472418, 232744202, 31838, -232771594, -314510882, 1170181539,
↪ -1073746239, -348368792, 1833010781, -1751250550, -22764, 1751264886, -1832971869, 348324664,
↪ 1073736768, -1170124067, 314472034, 232743946, 32537, -232771930, -314511266, 1170182179, -1073746239,
↪ -348369240, 1833011165, -1751250422, -23258, 1751265014, -1832971485, 348324184, 1073736640,
↪ -1170123427, 314471586, 232743610, 33237, -232772202, -314511714, 1170182819, -1073746367, -348369720,
```

```
  ↪ 1833011549, -1751250166, -23753, 1751265142, -1832971101, 348323704, 1073736576, -1170122787, 314471202,
  ↪  232743290, 33937, -232772522, -314512162, 1170183331, -1073746495, -348370232, 1833012061, -1751250038,
  ↪  -24248, 1751265398, -1832970589, 348323256, 1073736448, -1170122147, 314470754, 232743018, 34637,
  ↪ -232772794, -314512546, 1170183971, -1073746623, -348370712, 1833012445, -1751249910, -24743,
  ↪ 1751265526, -1832970205, 348322744, 1073736384, -1170121635, 314470306, 232742698, 35336, -232773114,
  ↪ -314512994, 1170184611, -1073746751, -348371160, 1833012829, -1751249782, -25238, 1751265654,
  ↪ -1832969821, 348322264, 1073736256, -1170120995, 314469922, 232742426, 36036, -232773450, -314513378,
  ↪ 1170185251, -1073746751, -348371640, 1833013341, -1751249526, -25733, 1751265782, -1832969309,
  ↪ 348321816, 1073736128, -1170120355, 314469474, 232742106, 36736, -232773706, -314513826, 1170185891,
  ↪ -1073746879, -348372152, 1833013725, -1751249398, -26228, 1751266038, -1832968925, 348321336,
  ↪ 1073736064, -1170119715, 314469090, 232741834, 37436, -232774042, -314514274, 1170186531, -1073747007,
  ↪ -348372632, 1833014109, -1751249270, -26723, 1751266166, -1832968541, 348320824, 1073735936,
  ↪ -1170119075, 314468642, 232741514, 38135, -232774298, -314514658, 1170187171, -1073747135, -348373080,
  ↪ 1833014493, -1751249142, -27217, 1751266294, -1832968029, 348320344, 1073735872, -1170118435, 314468194,
  ↪  232741178, 38835, -232774634, -314515106, 1170187811, -1073747263, -348373560, 1833015005, -1751249014,
  ↪  -27712, 1751266422, -1832967645, 348319896, 1073735744, -1170117795, 314467810, 232740922, 39535,
  ↪ -232774890, -314515490, 1170188323, -1073747263, -348374072, 1833015389, -1751248758, -28207,
  ↪ 1751266550, -1832967261, 348319416, 1073735616, -1170117155, 314467362, 232740586, 40234, -232775226,
  ↪ -314515938, 1170188963, -1073747391, -348374552, 1833015773, -1751248630, -28702, 1751266806,
  ↪ -1832966877, 348318904, 1073735552, -1170116643, 314466978, 232740330, 40934, -232775546, -314516386,
  ↪ 1170189603, -1073747519, -348375000, 1833016285, -1751248502, -29197, 1751266934, -1832966365,
  ↪ 348318424, 1073735424, -1170116003, 314466530, 232739994, 41634, -232775818, -314516770, 1170190243,
  ↪ -1073747647, -348375480, 1833016669, -1751248374, -29692, 1751267062, -1832965981, 348317976,
  ↪ 1073735360, -1170115363, 314466082, 232739738, 42334, -232776138, -314517218, 1170190883, -1073747775,
  ↪ -348375992, 1833017053, -1751248246, -30187, 1751267318, -1832965597, 348317496, 1073735232,
  ↪ -1170114723, 314465698, 232739402, 43033, -232776410, -314517602, 1170191523, -1073747775, -348376440,
  ↪ 1833017565, -1751247990, -30682, 1751267318, -1832965085, 348316984, 1073735104, -1170114083, 314465250,
  ↪  232739082, 43733, -232776730, -314518050, 1170192163, -1073747903, -348376920, 1833017949, -1751247862,
  ↪  -31176, 1751267574, -1832964701, 348316504, 1073735040, -1170113443, 314464866, 232738810, 44433,
  ↪ -232777002, -314518498, 1170192675, -1073748031, -348377400, 1833018333, -1751247734};

int main(void)
{
    unsafe {
        dsp_os3_return_code_t return_code = DSP_OS3_NO_ERROR;

        // Input data for both channels
        const int32_t * unsafe input_data[NUM_CHANNELS] = {s1k_0db_16, im4k5k_m6dB_16};

        // OS3 instances variables
        // -----------------------
        // State and Control structures (one per channel)
        int32_t           dsp_os3_delay[NUM_CHANNELS][(DSP_OS3_N_COEFS/DSP_OS3_N_PHASES)<<1];        // Delay
          ↪ line length is 1/3rd of number of coefs as over-sampler by 3 (and double for circular buffer
          ↪ simulation)
        dsp_os3_ctrl_t    dsp_os3_ctrl[NUM_CHANNELS];

        //Init OS3
        for (int i=0; i<NUM_CHANNELS; i++) {

            printf("Init os3 channel %d\n", i);
            // Process init
            // ------------
            // Set delay line base to ctrl structure
            dsp_os3_ctrl[i].delay_base        = (int*)dsp_os3_delay[i];

            // Init instance
            if (dsp_os3_init(&dsp_os3_ctrl[i]) != DSP_OS3_NO_ERROR) {
                printf("Error on init\n");
                return_code = DSP_OS3_ERROR;
            }

            // Sync (i.e. clear data)
            // ----
            if (dsp_os3_sync(&dsp_os3_ctrl[i]) != DSP_OS3_NO_ERROR) {
                printf("Error on sync\n");
                return_code = DSP_OS3_ERROR;
            }

        }

        for (int s=0; s<NUM_OUTPUT_SAMPLES; s++) {
            for (int i=0; i<NUM_CHANNELS; i++) {
                // Run through output samples, we read 1 input sample every 3 output samples
                // This is given when the phase member of the control structure reaches '0'
```

```
                // Check if a new input sample is needed
                // If it is needed read it into the control structure and call function to write to delay line
                // If we have reached end of file, signal it.
                if (dsp_os3_ctrl[i].phase == 0) {
                    dsp_os3_ctrl[i].in_data = *input_data[i];
                    //printf("in = %d\n", (int32_t) *input_data[i]);
                    input_data[i]++;

                    if (dsp_os3_input(&dsp_os3_ctrl[i]) != DSP_OS3_NO_ERROR) {
                        printf("Error on os3 input\n");
                        return_code = DSP_OS3_ERROR;
                    }
                }

                // Call sample rate conversion. Always output a sample on each loop
                if (dsp_os3_proc(&dsp_os3_ctrl[i]) != DSP_OS3_NO_ERROR) {
                    printf("Error on os3 process\n");
                    return_code = -4;
                }
                printf("%d\n", dsp_os3_ctrl[i].out_data);
            }
        }
        return (int)return_code;
    }
}
```

APPENDIX C - Correct Results Listings

This section includes the source code for all of the example programs.

C.1 Adaptive Filtering Functions

```
LMS 5
+0.003133 +0.096867
+0.009405 +0.090595
+0.010949 +0.089051
+0.012192 +0.087808
+0.009736 +0.090264
+0.009765 +0.090235
+0.009793 +0.090207
+0.009822 +0.090178
+0.009851 +0.090149
+0.009880 +0.090120
+0.009909 +0.090091
+0.009938 +0.090062
+0.009966 +0.090034
+0.009995 +0.090005
+0.010024 +0.089976
+0.010053 +0.089947
+0.010082 +0.089918
+0.010110 +0.089890
+0.010139 +0.089861
+0.010168 +0.089832
+0.010197 +0.089803
+0.010225 +0.089775
+0.010254 +0.089746
+0.010283 +0.089717
+0.010312 +0.089688
+0.010340 +0.089660
+0.010369 +0.089631
+0.010398 +0.089602
+0.010426 +0.089574
+0.010455 +0.089545
+0.010484 +0.089516
+0.010512 +0.089488
+0.010541 +0.089459
+0.010570 +0.089430
+0.010598 +0.089402

Normalized LMS 5
+0.003133 +0.096867
+0.010367 +0.089633
+0.012796 +0.087204
+0.014894 +0.085106
+0.013266 +0.086734
+0.014134 +0.085866
+0.014992 +0.085008
+0.015842 +0.084158
+0.016684 +0.083316
+0.017517 +0.082483
+0.018342 +0.081658
+0.019159 +0.080841
+0.019967 +0.080033
```

```
+0.020767 +0.079233
+0.021560 +0.078440
+0.022344 +0.077656
+0.023121 +0.076879
+0.023889 +0.076111
+0.024651 +0.075349
+0.025404 +0.074596
+0.026150 +0.073850
+0.026888 +0.073112
+0.027620 +0.072380
+0.028343 +0.071657
+0.029060 +0.070940
+0.029769 +0.070231
+0.030472 +0.069528
+0.031167 +0.068833
+0.031855 +0.068145
+0.032537 +0.067463
+0.033211 +0.066789
+0.033879 +0.066121
+0.034540 +0.065460
+0.035195 +0.064805
+0.035843 +0.064157
```

C.2 Fixed Coefficient Filtering Functions

```
FIR Filter Results
Dst[0] = 0.012100
Dst[1] = 0.026400
Dst[2] = 0.043000
Dst[3] = 0.062000
Dst[4] = 0.083500
Dst[5] = 0.107600
Dst[6] = 0.134400
Dst[7] = 0.164000
Dst[8] = 0.196500
Dst[9] = 0.232000
Dst[10] = 0.270600
Dst[11] = 0.312400
Dst[12] = 0.357500
Dst[13] = 0.406000
Dst[14] = 0.458000
Dst[15] = 0.513600
Dst[16] = 0.572900
Dst[17] = 0.636000
Dst[18] = 0.703000
Dst[19] = 0.774000
Dst[20] = 0.849100
Dst[21] = 0.928400
Dst[22] = 1.012000
Dst[23] = 1.100000
Dst[24] = 1.192500
Dst[25] = 1.289600
Dst[26] = 1.391400
Dst[27] = 1.498000
Dst[28] = 1.609500
Dst[29] = 1.726000
Dst[30] = 1.802500
Dst[31] = 1.879000
Dst[32] = 1.955500
Dst[33] = 2.032000
Dst[34] = 2.108500
Dst[35] = 2.185000
Dst[36] = 2.261500
Dst[37] = 2.338000
Dst[38] = 2.414500
Dst[39] = 2.491000
Dst[40] = 2.567500
Dst[41] = 2.644000
Dst[42] = 2.720500
Dst[43] = 2.797000
Dst[44] = 2.873500
Dst[45] = 2.950000
Dst[46] = 3.026500
Dst[47] = 3.103000
Dst[48] = 3.179500
Dst[49] = 3.256000

IIR Biquad Filter Results
Dst[0] = 0.012100
Dst[1] = 0.028094
Dst[2] = 0.048748
```

```
Dst[3] = 0.057639
Dst[4] = 0.065582
Dst[5] = 0.071627
Dst[6] = 0.077265
Dst[7] = 0.082561
Dst[8] = 0.087748
Dst[9] = 0.092869
Dst[10] = 0.097964
Dst[11] = 0.103045
Dst[12] = 0.108121
Dst[13] = 0.113194
Dst[14] = 0.118265
Dst[15] = 0.123336
Dst[16] = 0.128407
Dst[17] = 0.133477
Dst[18] = 0.138548
Dst[19] = 0.143618
Dst[20] = 0.148689
Dst[21] = 0.153759
Dst[22] = 0.158830
Dst[23] = 0.163900
Dst[24] = 0.168970
Dst[25] = 0.174041
Dst[26] = 0.179111
Dst[27] = 0.184182
Dst[28] = 0.189252
Dst[29] = 0.194323
Dst[30] = 0.199393
Dst[31] = 0.204463
Dst[32] = 0.209534
Dst[33] = 0.214604
Dst[34] = 0.219675
Dst[35] = 0.224745
Dst[36] = 0.229815
Dst[37] = 0.234886
Dst[38] = 0.239956
Dst[39] = 0.245027
Dst[40] = 0.250097
Dst[41] = 0.255168
Dst[42] = 0.260238
Dst[43] = 0.265308
Dst[44] = 0.270379
Dst[45] = 0.275449
Dst[46] = 0.280520
Dst[47] = 0.285590
Dst[48] = 0.290661
Dst[49] = 0.295731

Cascaded IIR Biquad Filter Results
Dst[0] = 0.000788
Dst[1] = 0.003924
Dst[2] = 0.012215
Dst[3] = 0.027035
Dst[4] = 0.048666
Dst[5] = 0.074798
Dst[6] = 0.103666
Dst[7] = 0.133224
Dst[8] = 0.162755
```

```
Dst[9] = 0.191477
Dst[10] = 0.219245
Dst[11] = 0.245924
Dst[12] = 0.271591
Dst[13] = 0.296325
Dst[14] = 0.320256
Dst[15] = 0.343501
Dst[16] = 0.366176
Dst[17] = 0.388383
Dst[18] = 0.410208
Dst[19] = 0.431725
Dst[20] = 0.452995
Dst[21] = 0.474067
Dst[22] = 0.494982
Dst[23] = 0.515771
Dst[24] = 0.536461
Dst[25] = 0.557073
Dst[26] = 0.577623
Dst[27] = 0.598124
Dst[28] = 0.618587
Dst[29] = 0.639019
Dst[30] = 0.659427
Dst[31] = 0.679817
Dst[32] = 0.700191
Dst[33] = 0.720554
Dst[34] = 0.740908
Dst[35] = 0.761255
Dst[36] = 0.781596
Dst[37] = 0.801932
Dst[38] = 0.822265
Dst[39] = 0.842595
Dst[40] = 0.862924
Dst[41] = 0.883250
Dst[42] = 0.903575
Dst[43] = 0.923899
Dst[44] = 0.944222
Dst[45] = 0.964545
Dst[46] = 0.984867
Dst[47] = 1.005188
Dst[48] = 1.025510
Dst[49] = 1.045831

Interpolation
INTERP taps=16 L=2
+0.003916 +0.007832
+0.005832 +0.009364
+0.002734 +0.013280
+0.010566 +0.015196
+0.012098 +0.012098
+0.016014 +0.019930
+0.017930 +0.021462
+0.014832 +0.025378
INTERP taps=24 L=3
+0.003916 +0.007832 +0.001916
+0.005448 +0.004734 +0.005832
+0.013280 +0.006650 +0.007364
+0.010182 +0.010566 +0.015196
+0.012098 +0.012098 +0.012098
```

```
+0.016014 +0.019930 +0.014014
+0.017546 +0.016832 +0.017930
+0.025378 +0.018748 +0.019462
INTERP taps=32 L=4
+0.003916 +0.007832 +0.001916 +0.001532
+0.000818 +0.011748 +0.009748 +0.003448
+0.002350 +0.008650 +0.013664 +0.011280
+0.004266 +0.010182 +0.010566 +0.015196
+0.012098 +0.012098 +0.012098 +0.012098
+0.016014 +0.019930 +0.014014 +0.013630
+0.012916 +0.023846 +0.021846 +0.015546
+0.014447 +0.020748 +0.025762 +0.023378
INTERP taps=40 L=5
+0.003916 +0.007832 +0.001916 +0.001532 -0.003098
+0.007832 +0.015664 +0.003832 +0.003064 -0.006196
+0.011748 +0.023496 +0.005748 +0.004595 -0.009295
+0.015664 +0.031329 +0.007664 +0.006127 -0.012393
+0.019580 +0.039161 +0.009580 +0.007659 -0.015491
+0.023496 +0.046993 +0.011496 +0.009191 -0.018589
+0.027412 +0.054825 +0.013412 +0.010723 -0.021688
+0.031329 +0.062657 +0.015329 +0.012254 -0.024786
INTERP taps=48 L=6
+0.003916 +0.007832 +0.001916 +0.001532 -0.003098 +0.003916
+0.011748 +0.009748 +0.003448 -0.001566 +0.000818 +0.011748
+0.013664 +0.011280 +0.000350 +0.002350 +0.008650 +0.013664
+0.015196 +0.008182 +0.004266 +0.010182 +0.010566 +0.015196
+0.012098 +0.012098 +0.012098 +0.012098 +0.012098 +0.012098
+0.016014 +0.019930 +0.014014 +0.013630 +0.009000 +0.016014
+0.023846 +0.021846 +0.015546 +0.010531 +0.012916 +0.023846
+0.025762 +0.023378 +0.012447 +0.014447 +0.020748 +0.025762
INTERP taps=56 L=7
+0.003916 +0.007832 +0.001916 +0.001532 -0.003098 +0.003916 +0.007832
+0.005832 +0.009364 -0.001182 +0.005448 +0.004734 +0.005832 +0.009364
+0.002734 +0.013280 +0.006650 +0.007364 +0.006266 +0.002734 +0.013280
+0.010566 +0.015196 +0.008182 +0.004266 +0.010182 +0.010566 +0.015196
+0.012098 +0.012098 +0.012098 +0.012098 +0.012098 +0.012098 +0.012098
+0.016014 +0.019930 +0.014014 +0.013630 +0.009000 +0.016014 +0.019930
+0.017930 +0.021462 +0.010916 +0.017546 +0.016832 +0.017930 +0.021462
+0.014832 +0.025378 +0.018748 +0.019462 +0.018364 +0.014832 +0.025378
INTERP taps=64 L=8
+0.003916 +0.007832 +0.001916 +0.001532 -0.003098 +0.003916 +0.007832 +0.001916
+0.005448 +0.004734 +0.005832 +0.009364 -0.001182 +0.005448 +0.004734 +0.005832
+0.013280 +0.006650 +0.007364 +0.006266 +0.002734 +0.013280 +0.006650 +0.007364
+0.010182 +0.010566 +0.015196 +0.008182 +0.004266 +0.010182 +0.010566 +0.015196
+0.012098 +0.012098 +0.012098 +0.012098 +0.012098 +0.012098 +0.012098 +0.012098
+0.016014 +0.019930 +0.014014 +0.013630 +0.009000 +0.016014 +0.019930 +0.014014
+0.017546 +0.016832 +0.017930 +0.021462 +0.010916 +0.017546 +0.016832 +0.017930
+0.025378 +0.018748 +0.019462 +0.018364 +0.014832 +0.025378 +0.018748 +0.019462

Decimation
DECIM taps=32 M=02
+0.003916 +0.013664 +0.012098 +0.023846 +0.027294 +0.028112 +0.037860 +0.036294
+0.048042 +0.051490 +0.052308 +0.062056 +0.060489 +0.072238 +0.075685 +0.076503
DECIM taps=32 M=03
+0.003916 +0.015196 +0.023846 +0.024196 +0.037860 +0.040210 +0.051490 +0.060140
+0.060489 +0.074154
DECIM taps=32 M=04
+0.003916 +0.012098 +0.027294 +0.037860 +0.048042 +0.052308 +0.060489 +0.075685
```

```
DECIM taps=32 M=05
+0.003916 +0.016014 +0.028112 +0.040210 +0.052308 +0.064405
DECIM taps=32 M=06
+0.003916 +0.023846 +0.037860 +0.051490 +0.060489
DECIM taps=32 M=07
+0.003916 +0.025762 +0.036294 +0.060140
DECIM taps=32 M=08
+0.003916 +0.027294 +0.048042 +0.060489
```

C.3 Math Functions

```
Test example for Math functions
===============================
Test Multiplication and Division
--------------------------------
Note: All calculations are done in Q8.24 format. That gives 7 digits of precision after the decimal point
Note: Maximum double representation of Q8.24 format: 127.99999994

Multiplication (11.31370850 x 11.31370850): 127.99999964

Multiplication (11.4 x 11.4). Will overflow!: -126.04000056

Saturated Multiplication (11.4 x 11.4): 127.99999994

Multiplication of small numbers (0.0005 x 0.0005): 0.00000024

Signed Division 1.12345600 / -128.00000000): -0.00877702

Signed Division -1.12345600 / -128.00000000): 0.00877702

Signed Division -1.12345600 / 127.99999990): -0.00877702

Signed Division 1.12345600 / 127.99999990): 0.00877702

Division 1.12345600 / 127.99999990): 0.00877696

Error report from test_multipliation_and_division:
Cases Error == -1: 1, percentage: 20.00
Cases Error == 0: 4, percentage: 80.00
Maximum Positive Error: 0
Maximum Negative Error: -1
Number of values checked: 5

Test Exponential and Logarithmic Functions
------------------------------------------
- Testing dsp_math_exp
Cases Error == -9: 1, percentage:  0.10
Cases Error == -6: 1, percentage:  0.10
Cases Error == -5: 4, percentage:  0.39
Cases Error == -4: 2, percentage:  0.19
Cases Error == -3: 1, percentage:  0.10
Cases Error == -2: 2, percentage:  0.19
Cases Error == -1: 10, percentage:  0.97
Cases Error == 0: 994, percentage: 96.88
Cases Error == 1: 9, percentage:  0.88
Cases Error == 2: 1, percentage:  0.10
Maximum Positive Error: 2
Maximum Negative Error: -26
Number of values checked: 1026
987 per thousand in one bit error

- Testing dsp_math_log
Cases Error == -2: 74, percentage:  7.21
Cases Error == -1: 397, percentage: 38.69
Cases Error == 0: 509, percentage: 49.61
Cases Error == 1: 45, percentage:  4.39
Cases Error == 2: 1, percentage:  0.10
Maximum Positive Error: 2
Maximum Negative Error: -2
Number of values checked: 1026
926 per thousand in one bit error

Test Squareroot
---------------
- Testing dsp_math_sqrt
Cases Error == -1: 9, percentage: 27.27
Cases Error == 0: 24, percentage: 72.73
Maximum Positive Error: 0
Maximum Negative Error: -1
Number of values checked: 33
1000 per thousand in one bit error

Test Trigonometric Functions
----------------------------
```

```
- Testing dsp_math_sin
Cases Error == -1: 92, percentage:  8.97
Cases Error == 0: 840, percentage: 81.87
Cases Error == 1: 94, percentage:  9.16
Maximum Positive Error: 1
Maximum Negative Error: -1
Number of values checked: 1026
1000 per thousand in one bit error

- Testing dsp_math_cos
Cases Error == -1: 129, percentage: 12.57
Cases Error == 0: 612, percentage: 59.65
Cases Error == 1: 285, percentage: 27.78
Maximum Positive Error: 1
Maximum Negative Error: -1
Number of values checked: 1026
1000 per thousand in one bit error

- Testing dsp_math_atan
ERROR: error 26222520 is greater than max_error 1
result is 0x0, Expected is 0xfe6fe048

Cases Error == 0: 64, percentage: 98.46
Errors smaller than min_error 1: 1
Maximum Positive Error: 26222520
Maximum Negative Error: 0
Number of values checked: 65
984 per thousand in one bit error

- Testing dsp_math_sinh
Cases Error == -15: 5, percentage:  0.49
Cases Error == -14: 2, percentage:  0.20
Cases Error == -13: 4, percentage:  0.39
Cases Error == -12: 5, percentage:  0.49
Cases Error == -11: 5, percentage:  0.49
Cases Error == -10: 3, percentage:  0.29
Cases Error == -9: 6, percentage:  0.59
Cases Error == -8: 3, percentage:  0.29
Cases Error == -7: 13, percentage:  1.27
Cases Error == -6: 6, percentage:  0.59
Cases Error == -5: 7, percentage:  0.68
Cases Error == -4: 20, percentage:  1.95
Cases Error == -3: 18, percentage:  1.76
Cases Error == -2: 33, percentage:  3.22
Cases Error == -1: 147, percentage: 14.34
Cases Error == 0: 451, percentage: 44.00
Cases Error == 1: 147, percentage: 14.34
Cases Error == 2: 33, percentage:  3.22
Cases Error == 3: 18, percentage:  1.76
Cases Error == 4: 20, percentage:  1.95
Cases Error == 5: 7, percentage:  0.68
Cases Error == 6: 6, percentage:  0.59
Cases Error == 7: 13, percentage:  1.27
Cases Error == 8: 3, percentage:  0.29
Cases Error == 9: 6, percentage:  0.59
Cases Error == 10: 3, percentage:  0.29
Cases Error == 11: 5, percentage:  0.49
Cases Error == 12: 5, percentage:  0.49
Cases Error == 13: 4, percentage:  0.39
Cases Error == 14: 2, percentage:  0.20
Cases Error == 15: 5, percentage:  0.49
Maximum Positive Error: 27
Maximum Negative Error: -27
Number of values checked: 1025
726 per thousand in one bit error

- Testing dsp_math_cosh
Cases Error == -15: 6, percentage:  0.59
Cases Error == -14: 8, percentage:  0.78
Cases Error == -13: 8, percentage:  0.78
Cases Error == -12: 10, percentage:  0.98
Cases Error == -11: 8, percentage:  0.78
Cases Error == -10: 8, percentage:  0.78
Cases Error == -9: 10, percentage:  0.98
Cases Error == -8: 8, percentage:  0.78
```

```
Cases Error == -7: 24, percentage:  2.34
Cases Error == -6: 18, percentage:  1.76
Cases Error == -5: 20, percentage:  1.95
Cases Error == -4: 30, percentage:  2.93
Cases Error == -3: 36, percentage:  3.51
Cases Error == -2: 64, percentage:  6.24
Cases Error == -1: 172, percentage: 16.78
Cases Error == 0: 503, percentage: 49.07
Cases Error == 1: 54, percentage:  5.27
Cases Error == 2: 10, percentage:  0.98
Cases Error == 3: 4, percentage:  0.39
Cases Error == 4: 2, percentage:  0.20
Cases Error == 5: 2, percentage:  0.20
Maximum Positive Error: 5
Maximum Negative Error: -27
Number of values checked: 1025
711 per thousand in one bit error
```

C.4 Matrix Functions

```
Matrix multiplication: R = X * Y
Input column vector X[2] (30 degrees from x axis):
0.86602539
0.50000000
Result of multiplying column vector X[2] with rotation matrix Y[2][2] (90 degrees rotation):
-0.49999994
0.86602545

Matrix negation: R = -X
-0.110000, -0.120000, -0.130000
-0.210000, -0.220000, -0.230000
-0.310000, -0.320000, -0.330000

Matrix / scalar addition: R = X + a
2.110000, 2.120000, 2.130000
2.210000, 2.220000, 2.230000
2.310000, 2.320000, 2.330000

Matrix / scalar multiplication: R = X + a
0.220000, 0.240000, 0.260000
0.420000, 0.440000, 0.460000
0.620000, 0.640000, 0.660000

Matrix / matrix addition: R = X + Y
0.520000, 0.540000, 0.560000
0.720000, 0.740000, 0.760000
0.920000, 0.940000, 0.960000

Matrix / matrix subtraction: R = X - Y
-0.300000, -0.300000, -0.300000
-0.300000, -0.300000, -0.300000
-0.300000, -0.300000, -0.300000

Matrix transposition
0.110000, 0.210000, 0.310000
0.120000, 0.220000, 0.320000
0.130000, 0.230000, 0.330000
```

C.5 Statistics Functions

```
Vector Mean = 0.355000
Vector Power (sum of squares) = 7.342500
Vector Root Mean Square = 0.383210
Vector Dot Product = 0.345500
```

C.6 Vector Functions

```
Minimum location = 0
Minimum = 0.110000
Maximum location = 49
Maximum = 0.600000
Vector Negate Result
Dst[0] = -0.110000
Dst[1] = -0.120000
Dst[2] = -0.130000
Dst[3] = -0.140000
Dst[4] = -0.150000
Vector Absolute Result
Dst[0] = 0.110000
Dst[1] = 0.120000
Dst[2] = 0.130000
Dst[3] = 0.140000
Dst[4] = -0.150000
Vector / scalar addition Result
Dst[0] = 2.110000
Dst[1] = 2.120000
Dst[2] = 2.130000
Dst[3] = 2.140000
Dst[4] = 2.150000
Vector / scalar multiplication Result
Dst[0] = 0.220000
Dst[1] = 0.240000
Dst[2] = 0.260000
Dst[3] = 0.280000
Dst[4] = 0.300000
Vector / vector addition Result
Dst[0] = 0.620000
Dst[1] = 0.640000
Dst[2] = 0.660000
Dst[3] = 0.680000
Dst[4] = 0.700000
Vector / vector subtraction Result
Dst[0] = -0.400000
Dst[1] = -0.400000
Dst[2] = -0.400000
Dst[3] = -0.400000
Dst[4] = -0.400000
Vector / vector multiplication Result
Dst[0] = 0.056100
Dst[1] = 0.062400
Dst[2] = 0.068900
Dst[3] = 0.075600
Dst[4] = 0.082500
Vector multiplication and scalar addition Result
Dst[0] = 2.056100
Dst[1] = 2.062400
Dst[2] = 2.068900
Dst[3] = 2.075600
Dst[4] = 2.082500
Vector / Scalar multiplication and vector addition Result
Dst[0] = 0.730000
Dst[1] = 0.760000
Dst[2] = 0.790000
Dst[3] = 0.820000
Dst[4] = 0.850000
Vector / Scalar multiplication and vector subtraction Result
Dst[0] = -0.453900
Dst[1] = -0.457600
Dst[2] = -0.461100
Dst[3] = -0.464400
Dst[4] = -0.467500
Vector / Vector multiplication and vector addition Result
Dst[0] = 0.610370
Dst[1] = 0.620370
Dst[2] = 0.630370
Dst[3] = 0.640370
Dst[4] = 0.650370
Vector / Vector multiplication and vector subtraction Result
Dst[0] = -0.553900
Dst[1] = -0.557600
```

```
Dst[2] = -0.561100
Dst[3] = -0.564400
Dst[4] = -0.567500
Complex vector / Vector multiplication Result
C_real = -22.000000, -28.000000, -34.000000, -40.000000, -42.000000, -20.000000, -22.000000, -40.000000,
  ↪ -9.000000, -14.000000, -17.000000, -22.000000,
C_imag = 20.000000, 36.000000, 56.000000, 80.000000, 76.000000, 20.000000, 20.000000, 60.000000, 17.000000,
  ↪ 48.000000, 65.000000, 104.000000,
```

C.7 Downsample by 3 Functions

Ommitted for brevity. Please see file `os3_test.expect` for golden expected test output.

C.8 Oversample by 3 Functions

Ommitted for brevity. Please see file `os3_test.expect` for golden expected test output.