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 14.0.0 and above
- XMOS DSP library module - Version 1.0.0 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 that is not specific to this application note is listed in the references appendix.
- For descriptions of XMOS related terms found in this document please see the XMOS Glossary[1].

Related Documents

- xCORE-200: The XMOS XS2 Architecture (ISA)[2]

[1] http://www.xmos.com/published/glossary
[2] https://www.xmos.com/published/xs2-isa-specification

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 - lib_dsp_adaptive.h
- Filtering - lib_dsp_filters.h
- Basic Math - lib_dsp_math.h
- Matrix Math - lib_dsp_matrix.h
- Statistics - lib_dsp_statistics.h
- Vectors - lib_dsp_vector.h

The library supports variable Q formats from Q16 to Q31, using the macros in the following header file : lib_dsp_qformat.h.

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 - app_vector

To build any of the applications, open src/main.xc by double clicking. 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. All of the test functions are written using Q24 format.

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.

2.2 Create and launch a run configuration

To view xSCOPE data in real-time you first need to create a run 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*. In the resulting dialog, double click on *xCORE Application*, then perform the following operations:

- On the *Main* tab select the desired 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 "lib_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

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, 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 <lib_dsp.h>

// Define constants

#define Q_M               1
#define Q_N               31

#define FIR_FILTER_LENGTH     160

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


// Declare global variables and arrays
int 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),
};


int fir_state[FIR_FILTER_LENGTH];
int lms_coeffs[FIR_FILTER_LENGTH];
int nlms_coeffs[FIR_FILTER_LENGTH];

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

              // Apply LMS Filter
  for( c = 5; c <= 5; c *= 2 )
  {
    printf( "LMS %u\n", c );
    for( int i = 0; i < FIR_FILTER_LENGTH; ++i ) lms_coeffs[i] = fir_coeffs[i];
    for( int i = 0; i < FIR_FILTER_LENGTH; ++i ) fir_state[i] = 0;
    for( int i = 0; i < c+30; ++i )
    {
      x = lib_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( int i = 0; i < FIR_FILTER_LENGTH; ++i ) nlms_coeffs[i] = fir_coeffs[i];
    for( int i = 0; i < FIR_FILTER_LENGTH; ++i ) fir_state[i] = 0;
    for( int i = 0; i < c+30; ++i )
    {
      x = lib_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, XMOS Ltd, All rights reserved
// XMOS DSP Library - Filtering Functions Test Program
// Uses Q24 format

// Include files
#include <stdio.h>
#include <xs1.h>
#include <lib_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( int x ) {if(x >=0) printf("+%f ",F31(x)); else printf("%f ",F31(x));}

// Declare global variables and arrays
int  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)};

int           Dst[INTERP_FILTER_LENGTH];

int 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)};

int 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),
};

int 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];
int filterState[INTERP_FILTER_LENGTH];

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

int main(void)
{
  int i, j, c, r, x, y;


                 // 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++)
  {
    Dst[i] =
      lib_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
  }

  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++)
  {
    Dst[i] =
      lib_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
  }

  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++)
  {
    Dst[i] =
      lib_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
  }

  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 )
    {
      lib_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( int 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 = lib_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, XMOS Ltd, All rights reserved
// XMOS DSP Library - Math Functions Test Program
// Uses Q24 format

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

// Define constants

#define Q_M               8
#define Q_N               24

int main(void)
{
//  printf ("Add With Saturate (2 + 4) : %d\n", F24(xmos_dsp_math_adds (2, 4)));;

  printf ("Multiplication (2 x 4) : %f\n", F24(lib_dsp_math_multiply (Q24(2.), Q24(4.), Q_N)));;

  printf ("Reciprocal (2) : %f\n", F24(lib_dsp_math_reciprocal (Q24(2.), Q_N)));;

  printf ("Inverse square root (2) : %f\n", F24(lib_dsp_math_invsqrroot (Q24(2.), Q_N)));;

  printf ("Square Root (2) : %f\n", F24(lib_dsp_math_squareroot (Q24(2.), Q_N)));;

  return (0);
}
```

B.4 Matrix Functions

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

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

// Define constants

#define Q_M               8
#define Q_N               24

#define MATRIX_NUM_ROWS   3
#define MATRIX_NUM_COLS   3

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

int main(void)
{
                                                  // Matrix negation: R = -X
  lib_dsp_matrix_negate (Src1,                    // 'input_matrix_X':  Pointer/reference to source data
                         Dst,                     // 'result_matrix_R': Pointer to the resulting 2-dimensional
                            ↪  data array
                         MATRIX_NUM_ROWS,         // 'row_count':       Number of rows in input and output
                            ↪ matrices
                         MATRIX_NUM_COLS);        // 'column_count':    Number of columns in input and output
                            ↪ matrices

  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]));

                                                  // Matrix / scalar addition: R = X + a
  lib_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
                       MATRIX_NUM_ROWS,           // 'row_count':       Number of rows in input and output
                          ↪ matrices
                       MATRIX_NUM_COLS);          // '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]));

                                                  // Matrix / scalar multiplication: R = X * a
  lib_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
                       MATRIX_NUM_ROWS,           // 'row_count':       Number of rows in input and output
                          ↪ matrices
                       MATRIX_NUM_COLS,           // '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]));

                                              // Matrix / matrix addition: R = X + Y
  lib_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
                       MATRIX_NUM_ROWS,       // 'row_count':       Number of rows in input and output
                          ↪ matrices
                       MATRIX_NUM_COLS);      // '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]));

                                              // Matrix / matrix subtraction: R = X - Y
  lib_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
                       MATRIX_NUM_ROWS,       // 'row_count':       Number of rows in input and output
                          ↪ matrices
                       MATRIX_NUM_COLS);      // '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]));

                                              // Matrix / matrix multiplication: R = X * Y
  lib_dsp_matrix_mulm (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
                       MATRIX_NUM_ROWS,       // 'row_count':       Number of rows in input and output
                          ↪ matrices
                       MATRIX_NUM_COLS,       // '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 / matrix multiplication: 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]));

                                              // Matrix transposition
  lib_dsp_matrix_transpose (Src1,             // 'input_matrix_X':  Pointer to source data array
                            Dst,              // 'result_matrix_R': Pointer to the resulting 2-dimensional
                               ↪  data array
                            MATRIX_NUM_ROWS,  // 'row_count':       Number of rows in input and output
                               ↪ matrices
                            MATRIX_NUM_COLS,  // '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]));

  return (0);
}
```

B.5 Statistics Functions

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

// Include files
#include <stdio.h>
#include <xs1.h>
#include <lib_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
int  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)};
int  Src2[] = { Q24(.51), Q24(.52), Q24(.53), Q24(.54), Q24(.55)};
int           Dst[SAMPLE_LENGTH];

int main(void)
{
  int result;

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

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

  result =
    lib_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 =
    lib_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 =
    lib_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, XMOS Ltd, All rights reserved
// XMOS DSP Library - Vector Functions Test Program
// Uses Q24 format

// Include files
#include <stdio.h>
#include <xs1.h>
#include <lib_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
int  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)};
int  Src2[] = { Q24(.51), Q24(.52), Q24(.53), Q24(.54), Q24(.55), Q24(.56), Q24(.57), Q24(.58), Q24(.59), Q24
   ↪ (.60)};
int  Src3[] = { Q24(.61), Q24(.62), Q24(.63), Q24(.64), Q24(.65), Q24(.66), Q24(.67), Q24(.68), Q24(.69), Q24
   ↪ (.70)};
int           Dst[SAMPLE_LENGTH];

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

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

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

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

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

  lib_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]));
  }

  lib_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]));
  }
```

```
lib_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]));
}

lib_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]));
}

lib_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]));
}

lib_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]));
}

lib_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]));
}

lib_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]));
}

lib_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]));
  }

  lib_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]));
  }

  lib_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]));
  }

  lib_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]));
  }

  return (0);
}
```

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] = 0.074797
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] = 0.271590
Dst[13] = 0.296325
Dst[14] = 0.320256
Dst[15] = 0.343501
Dst[16] = 0.366176
Dst[17] = 0.388382
Dst[18] = 0.410208
Dst[19] = 0.431725
Dst[20] = 0.452995
Dst[21] = 0.474067
Dst[22] = 0.494981
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] = 0.761254
Dst[36] = 0.781595
Dst[37] = 0.801932
Dst[38] = 0.822265
Dst[39] = 0.842595
Dst[40] = 0.862923
Dst[41] = 0.883250
Dst[42] = 0.903575
Dst[43] = 0.923899
Dst[44] = 0.944222
Dst[45] = 0.964544
Dst[46] = 0.984866
Dst[47] = 1.005188
Dst[48] = 1.025509
Dst[49] = 1.045830

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

```
Multiplication (2 x 4) : 8.000000
Reciprocal (2) : 0.500000
Inverse square root (2) : 0.707107
Square Root (2) : 1.414214
```

C.4 Matrix Functions

```
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 / matrix multiplication: R = X * Y
0.185600, 47.702807, -75.889648
0.338600, 87.026792, 7.209260
0.491600, 126.350807, 90.316030
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] = -127.389999
Dst[1] = -127.379997
Dst[2] = -127.370003
Dst[3] = -127.360001
Dst[4] = -127.349998
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
```