Appendix A

Global Definitions

```
/**************************************/
/*                                    */
/*  Appendix A -- Global Definitions  */
/*                                    */
/*  globDefs.h                        */
/*                                    */
/*  global definitions                */
/*                                    */
/**************************************/

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

#define EOL 10
#define STOP_CHAR 36
#define SPACE 32
#define TRUE 1
#define FALSE 0
#define PI 3.14159265
#define TWO_PI 6.2831853
#define TEN (double) 10.0
#define MAX_COLUMNS 20
#define MAX_ROWS 20

/*  structure definition for single precision complex */
/* struct complex
    {
    float Re;
    float Im;
    }; */
```

```
/* structure definition for double precision complex */
struct complex
     {
     double Re;
     double Im;
     };

typedef int logical;
typedef double real;
```

Appendix

B

Prototypes for C Functions

```
/*************************************************/
/*                                               */
/*  Appendix B -- Prototypes for C Functions     */
/*                                               */
/*************************************************/

int LaguerreMethod(   int order,
                      struct complex coef[],
                      struct complex *zz,
                      real epsilon,
                      real epsilon2,
                      int maxIterations);

void unwrapPhase( int ix, real *phase);

void butterworthFreqResponse(  int order,
                               real frequency,
                               real *magnitude,
                               real *phase);

void butterworthImpulseResponse(   int order,
                                   real deltaT,
                                   int npts,
                                   real yval[]);

void chebyshevFreqResponse(    int order,
                               float ripple,
                               char normalizationType,
                               float frequency,
                               float *magnitude,
                               float *phase);
```

```
void chebyshevImpulseResponse(    int order,
                                  float ripple,
                                  char normalizationType,
                                  float deltaT,
                                  int npts,
                                  float yval[]);


void cauerOrderEstim(real omegaPass,
                     real omegaStop,
                     real maxPassLoss,
                     real maxStopLoss,
                     int *order,
                     real *actualMinStopLoss);


void cauerCoeffs(real omegaPass,
                 real omegaStop,
                 real maxPassLoss,
                 int order,
                 real aa[],
                 real bb[],
                 real cc[],
                 int *numSecs,
                 real *hZero,
                 real *pZero);

void cauerFreqResponse(   int order,
                          real aa[],
                          real bb[],
                          real cc[],
                          real hZero,
                          real pZero,
                          real frequency,
                          real *magnitude,
                          real *phase);


void cauerRescale(    int order,
                      real aa[],
                      real bb[],
                      real cc[],
                      real *hZero,
                      real *pZero,
                      real alpha);

void besselCoefficients(  int order,
                          char typeOfNormalization,
                          real coef[]);
```

```
void besselFreqResponse(  int order,
                          real coef[],
                          real frequency,
                          real *magnitude,
                          real *phase);

void besselGroupDelay(    int order,
                          real coef[],
                          real frequency,
                          real delta,
                          real *groupDelay);

void dft(    struct complex x[],
             struct complex xx[],
             int nn);

void dft2(   struct complex x[],
             struct complex xx[],
             int nn);

void fft(    struct complex x[],
             struct complex xx[],
             int nn);

void cgdFirResponse( int firType,
                     int numbTaps,
                     real hh[],
                     logical dbScale,
                     int numberOfPoints,
                     real hD[]);

void normalizeResponse(   logical dbScale,
                          int numberOfPoints,
                          real hh[]);

void idealLowpass(   int numbTaps,
                     real omegaU,
                     real coefficient[]);

void idealHighpass(  int numbTaps,
                     real omegaL,
                     real coefficient[]);

void idealBandpass(  int numbTaps,
                     real omegaL,
                     real omegaU,
                     real coefficient[]);
```

```
void idealBandstop(  int numbTaps,
                     real omegaL,
                     real omegaU,
                     real coefficient[]);

real contRectangularResponse( real freq,
                              real tau,
                              logical dbScale);

real discRectangularResponse( real freq,
                              int M,
                              logical normAmp);

real contTriangularResponse(  real freq,
                              real tau,
                              logical dbScale);

real discTriangularResponse(  real freq,
                              int M,
                              logical normAmp);

void triangularWindow( int N, real window[]);

void makeLagWindow(  int N,
                     real window[],
                     int center,
                     real outWindow[]);

void makeDataWindow( int N,
                     real window[],
                     real outWindow[]);

void hannWindow( int nn, real window[]);
void hammingWindow( int nn, real window[]);

int fsDesign(    int nn,
                 int firType,
                 real aa[],
                 real h[]);

findSbPeak(  int bandConfig[],
             int numPts,
             real hh[]);

real goldenSearch(    int firType,
                      int numbTaps,
                      real hD[],
                      real gsTol,
                    int numFreqPts,
                    int bandConfig[],
                    real *fmin);

void setTrans(  int bandConfig[],
                real x,
                real hD[]);

real goldenSearch2( real rhoMin,
                    real rhoMax,
                    int firType,
                    int numbTaps,
                    real hD[],
                    real gsTol,
                    int numFreqPts,
                    real origins[],
                    real slopes[],
                    int bandConfig[],
                    real *fmin);



void setTransition( real origins[],
                    real slopes[],
                    int bandConfig[],
                    real x,
                    real Hd[]);

void optimize2( real yBase,
                int firType,
                int numbTaps,
                real hD[],
                real gsTol,
                int numFreqPts,
                int bandConfig[],
                real tweakFactor,
                real rectComps[]);

void dumpRectComps( real origins[],
                    real slopes[],
                    int numTransSamps,
                    real x);

real gridFreq( real gridParam[], int gI);

real desLpfResp( real freqP, real freq);

real weightLp( real kk, real freqP, real freq);
```

```
void remezError( real gridParam[],
                 int gridMax,
                 int r,
                 real kk,
                 real freqP,
                 int iFF[],
                 real ee[]);


real computeRemezA(  real gridParam[],
                     int gridMax,
                     int r,
                     real kk,
                     real freqP,
                     int iFF[],
                     int initFlag,
                     real contFreq);

void remezSearch(real ee[],
                 real absDelta,
                 int gP,
                 int iFF[],
                 int gridMax,
                 int r,
                 real gridParam[]);

int remezStop(  int iFF[],  int r);

int remezStop2(  real ee[], int iFF[], int r);

void remezFinish(real extFreq[],
                 int nn,
                 int r,
                 real freqP,
                 real kk,
                 real aa[],
                 real h[]);

void remez(  int nn,
             int r,
             int gridDensity,
             real kk,
             real freqP,
             real freqS,
             real extFreq[],
             real h[]);
```

```
iirResponse( struct complex a[],
             int bigN,
             struct complex b[],
             int bigM,
             int numberOfPoints,
             logical dbScale,
             real magnitude[],
             real phase[]);

void impulseInvar(    struct complex pole[],
                      int numPoles,
                      struct complex zero[],
                      int numZeros,
                      real hZero,
                      real bigT,
                      struct complex a[],
                      struct complex b[]);

void stepInvar(  struct complex pole[],
                 int numPoles,
                 struct complex zero[],
                 int numZeros,
                 real hZero,
                 real bigT,
                 struct complex a[],
                 struct complex b[]);

void bilinear(   struct complex pole[],
                 int numPoles,
                 struct complex zero[],
                 int numZeros,
                 real hZero,
                 real bigT,
                 struct complex a[],
                 struct complex b[]);


struct complex cmplx(  real A, real B);
struct complex cAdd( struct complex A, struct complex B);
struct complex cSub( struct complex A, struct complex B);
real cMag(struct complex A);
real cAbs(struct complex A);
double cdAbs(struct complex A);
real arg(struct complex A);
struct complex cSqrt( struct complex A);
struct complex cMult(  struct complex A, struct complex B);
struct complex sMult( real a, struct complex B);
struct complex cDiv( struct complex numer, struct complex denom);
```

```
real sincSqrd( real x);
real sinc( real x);
real acosh( real x);
void pause( logical enabled);
int bitRev(  int L, int N);
int log2(  int N );
real ipow(   real x, int k);
```

Appendix C

Functions for Complex Arithmetic

```
/*****************************************************/
/*                                                   */
/*  Appendix C -- Functions for Complex Arithmetic   */
/*                                                   */
/*****************************************************/
#include "globDefs.h"
#include "protos.h"


/**************************************/
/*                                    */
/*     cmplx()                        */
/*                                    */
/*  merges two real into one complex  */
/*                                    */
/**************************************/
struct complex cmplx(  real A, real B)
{
struct complex result;

result.Re = A;
result.Im = B;
return( result);
}

/***************************************/
/*                                     */
/*  cAdd()                             */
/*                                     */
/***************************************/
struct complex cAdd(
                      struct complex A,
                      struct complex B)
{
```

```
result.Re = A.Re + B.Re;
result.Im = A.Im + B.Im;
return( result);
}

/*************************************/
/*                                   */
/*  cSub()                           */
/*                                   */
/*************************************/
struct complex cSub(
                    struct complex A,
                    struct complex B)
{
struct complex result;

result.Re = A.Re - B.Re;
result.Im = A.Im - B.Im;
return( result);
}

/*************************************/
/*                                   */
/*  cMag()                           */
/*                                   */
/*************************************/
real cMag(struct complex A)
{
real result;
result = sqrt(A.Re*A.Re + A.Im*A.Im);
return( result);
}

/*************************************/
/*                                   */
/*  cAbs()                           */
/*                                   */
/*************************************/
real cAbs(struct complex A)
{
real result;
result = sqrt(A.Re*A.Re + A.Im*A.Im);
return( result);
}
```

```
/***************************************/
/*                                     */
/*  cdAbs()                            */
/*                                     */
/***************************************/
double cdAbs(struct complex A)
{
double result;
result = sqrt(A.Re*A.Re + A.Im*A.Im);
return( result);
}

/***************************************/
/*                                     */
/*  arg()                              */
/*                                     */
/***************************************/
real arg(struct complex A)
{
real result;

if( (A.Re == 0.0)  && (A.Im == 0.0) )
    {
    result = 0.0;
    }
else
    {
    result = atan2( A.Im, A.Re );
    }
return( result);
}

/****************************************/
/*                                      */
/*  cSqrt()                             */
/*                                      */
/****************************************/
struct complex cSqrt( struct complex A)
{
struct complex result;
double r, theta;

r = sqrt(cdAbs(A));
theta = arg(A)/2.0;
result.Re = r * cos(theta);
result.Im = r * sin(theta);
return( result);
}
```

```
/*****************************************/
/*                                       */
/*  cMult()                              */
/*                                       */
/*****************************************/
struct complex cMult(
                          struct complex A,
                          struct complex B)
{
struct complex result;

result.Re = A.Re*B.Re - A.Im*B.Im;
result.Im = A.Re*B.Im + A.Im*B.Re;
return( result);
}
/*****************************************/
/*                                       */
/*  sMult()                              */
/*                                       */
/*****************************************/
struct complex sMult(
                          real a,
                          struct complex B)
{
struct complex result;

result.Re = a*B.Re;
result.Im = a*B.Im;
return( result);
}

/*****************************************/
/*                                       */
/*  cDiv()                               */
/*                                       */
/*****************************************/
struct complex cDiv(
                      struct complex numer,
                      struct complex denom)
{
real bottom,real_top,imag_top;
struct complex result;

bottom = denom.Re*denom.Re + denom.Im*denom.Im;
real_top = numer.Re*denom.Re + numer.Im*denom.Im;
imag_top = numer.Im*denom.Re - numer.Re*denom.Im;
result.Re = real_top/bottom;
result.Im = imag_top/bottom;
return( result);
}
```

Appendix D

Miscellaneous Support Functions

```
/**********************************************/
/*                                            */
/*  Appendix D                                */
/*                                            */
/*  Miscellaneous Support Functions           */
/*                                            */
/**********************************************/
#include <stdlib.h>
#include <math.h>
#include <ctype.h>
#include "globDefs.h"
#include "protos.h"

/*******************************/
/*                             */
/*   sincSqrd()                */
/*                             */
/*******************************/


real sincSqrd( real x)
{
real result;
if( x==0.0)
    {
    result = 1.0;
    }
else
    {
    result = sin(x)/x;
    result = result * result;
    }
return(result);
}
/***********************************/
/*                                 */
/*   sinc()                        */
/*                                 */
/***********************************/

real sinc( real x)
{
real result;
if( x==0.0)
    {
    result = 1.0;
    }
else
    {
    result = sin(x)/x;
    }
return(result);
}


/***********************************/
/*                                 */
/*   acosh()                        */
/*                                 */
/***********************************/

real acosh( real x)
{
real result;
result = log(x+sqrt(x*x-1.0));
return(result);
}
/***********************************/
/*                                 */
/*   pause()                       */
/*                                 */
/***********************************/

void pause( logical enabled)
{
char inputString[20];
if(enabled) {
    printf("enter anything to continue\n");
    gets(inputString);
    }
return;
}


/***********************************/
/*                                 */
/*   bitRev()                      */
/*                                 */
/***********************************/

int bitRev(  int L, int N)
{
int work, work2, i, bit;

work2 = 0;
work = N;
for(i=0; i<L; i++) {
    bit = work%2;
    work2 = 2 * work2 + bit;
    work /=2;
    }
return(work2);
}

/***********************************/
/*                                 */
/*   log2()                        */
/*                                 */
/***********************************/

int log2(  int N )
{
int work, result;

result = 0;
work = N;
for(;;) {
    if(work == 0) break;
    work /=2;
    result ++;
    }
return(result-1);
}
```

```
/********************************/
/*                              */
/*   ipow()                     */
/*                              */
/********************************/

real ipow(   real x,
             int k)
{
real result;
int n;
if(k==0)
    {result = 1.0;}
else
    {result = x;
    for( n=2; n<=k; n++)
        { result = result * x;}
    }
return(result);
}
```

Bibliography

Abramowitz, M., and I. A. Stegun: *Handbook of Mathematical Functions*, National Bureau of Standards, Appl. Math Series 55, 1966.

Antoniou, A.: *Digital Filters: Analysis and Design*, McGraw-Hill, New York, 1979.

Antoniou, A.: "Accelerated Procedure for the Design of Equiripple Non-recursive Digital Filters,“ *Proceedings IEE, PART G*, vol. 129, pp. 1–10.

Bartlett, M. S.: "Periodogram Analysis and Continuous Spectra," *Biometrika*, vol. 37, pp. 1–16, 1950.

Blackman, R. B., and J. W. Tukey: *The Measurement of Power Spectra*, Dover, New York, 1958.

Boyer, C. B.: *A History of Mathematics*, Wiley, New York, 1968.

Brigham, E. O.: *The Fast Fourier Transform*, Prentice-Hall, Englewood Cliffs, N.J., 1974.

Burrus, C. S., and T. W. Parks: *DFT/FFT and Convolution Algorithms*, Wiley-Interscience, New York, 1984.

Cadzow, J. A.: *Discrete-Time Systems*, Prentice-Hall, Englewood Cliffs, N.J., 1973.

Chen, C-T.: *Linear System Theory and Design*, Holt, Rinehart and Winston, New York, 1984.

Cheyney, E. W.: *Introduction to Approximation Theory*, McGraw-Hill, New York, 1966.

Dolph, C. L.: "A Current Distribution for Broadside Arrays Which Optimizes the Relationship Between Beam Width and Side-Lobe Level," *Proc. IRE*, vol. 35, pp. 335–348, June 1946.

Dym, H., and H. P. McKean: *Fourier Series and Integrals*, Academic, New York, 1972.

Hamming, R. W.: *Numerical Methods for Engineers and Scientists*, McGraw-Hill, New York, 1962.

Hamming, R. W.: *Digital Filters*, 2d ed., Prentice-Hall, Englewood Cliffs, N.J., 1983.

Harris, F. J.: "On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform, "*Proc. IEEE*, vol. 66, pp. 51–83, January 1978.

Haykin, S.: *Communication Systems*, 2d ed., Wiley, New York, 1983.

Helms, H. D.: "Nonrecursive Digital Filters: Design Methods for Achieving Specifications on Frequency Response," *IEEE Trans. Audio and Electroacoust.*, vol. AU-16, pp. 336–342, September 1968.

Helms, H. D.: "Digital Filters with Equiripple or Minimax Responses," *IEEE Trans Audio Electroacoust.*, vol. AU-19, pp. 87–94, March 1971.

Herrmann, O.: "Design of Nonrecursive Digital Filters with Linear Phase," *Electronics Letters*, vol. 6, pp. 328–329, 1970.

Hofstetter, E. M., A. V. Oppenheim, and J. Siegel: "A New Technique for the Design of Non-Recursive Digital Filters," *Proc. Fifth Annual Princeton Conf. on Inform. Sci. and Syst.*, pp. 64–72, 1971.

Kanefsky, M.: *Communication Techniques for Digital and Analog Signals*, Harper & Row, New York, 1985.

Kay, S. M.: *Modern Spectral Estimators: Theory & Application*, Prentice-Hall, Englewood Cliffs, N.J., 1988.

Marple, S. L.: *Digital Spectral Analysis with Applications*, Prentice-Hall, Englewood Cliffs, N.J., 1987.

Nussbaumer, H. J.: *Fast Fourier Transform and Convolution Algorithms*, Springer-Verlag, New York, 1982.

Oppenheim, A. V., and R. W. Schafer: *Digital Signal Processing*, Prentice-Hall, Englewood Cliffs, N.J., 1975.

Oppenheim, A. V., and R. W. Schafer: *Discrete-Time Signal Processing*, Prentice-Hall, Englewood Cliffs, N.J., 1989.

Papoulis, A.: *The Fourier Integral and Its Applications*, McGraw-Hill, New York, 1962.

Parks, T. W., and C. S. Burrus: *Digital Filter Design*, Wiley-Interscience, New York, 1987.

Parks, T. W., and J. H. McClellan: "Chebyshev Approximation for Nonrecursive Digital Filters with Linear Phase," *IEEE Trans. Circuit Theory*, vol. CT-19, pp. 189–194, March 1972.

Parks, T. W., and J. H. McClellan: "A Computer Program for Designing Optimum FIR Linear Phase Digital Filters," *IEEE Trans. Audio Electroacoust.*, vol. AU-21, pp. 506–526, December 1973.

Peled, A., and B. Liu: *Digital Signal Processing*, Wiley, New York, 1976.

Press, W. H., et al.: *Numerical Recipes*, Cambridge University Press, Cambridge, 1986.

Priestley, M. B.: *Spectral Analysis and Time Series*, vol. 1: *Univariate Series*, Academic, London, 1981.

Rabiner, L. R., and B. Gold: *Theory and Application of Digital Signal Processing*, Prentice-Hall, Englewood Cliffs, N.J., 1975.

Roberts, A. A., and C. T. Mullis: *Digital Signal Processing*, Addison-Wesley, Reading, Mass., 1987.

Rorabaugh, B.: *Signal Processing Design Techniques*, TAB Professional and Reference Books, Blue Ridge Summit, Pa., 1986.

Schwartz, L.: *Théorie des distributions*, Herman & Cie, Paris, 1950.

Schwartz, R. J., and B. Friedland: *Linear Systems*, McGraw-Hill, New York, 1965.

Spiegel, M. R.: *Laplace Transforms*, Schaum's Outline Series, McGraw-Hill, New York, 1965.

Stanley, W. D.: *Digital Signal Processing*, Reston, Reston, Va., 1975.

Tufts, D. W., and J. T. Francis: "Designing Digital Low-pass Filters—Comparison of Some Methods and Criteria," *IEEE Trans. Audio Electroacoust.*, vol. AU-18, pp. 487–494, December 1970.

Tufts, D. W., D. W. Rorabacher, and M. E. Mosier: "Designing Simple, Effective Digital Filters," *IEEE Trans. Audio Electroacoust.*, vol. AU-18, pp. 142–158, 1970.

Van Valkenburg, M. E.: *Network Analysis*, Prentice-Hall, Englewood Cliffs, N.J., 1974.

Weaver, H. J.: *Theory of Discrete and Continuous Fourier Analysis*, Wiley, New York, 1989.

Williams, C. S.: *Designing Digital Filters*, Prentice-Hall, Englewood Cliffs, N.J., 1986.

Index