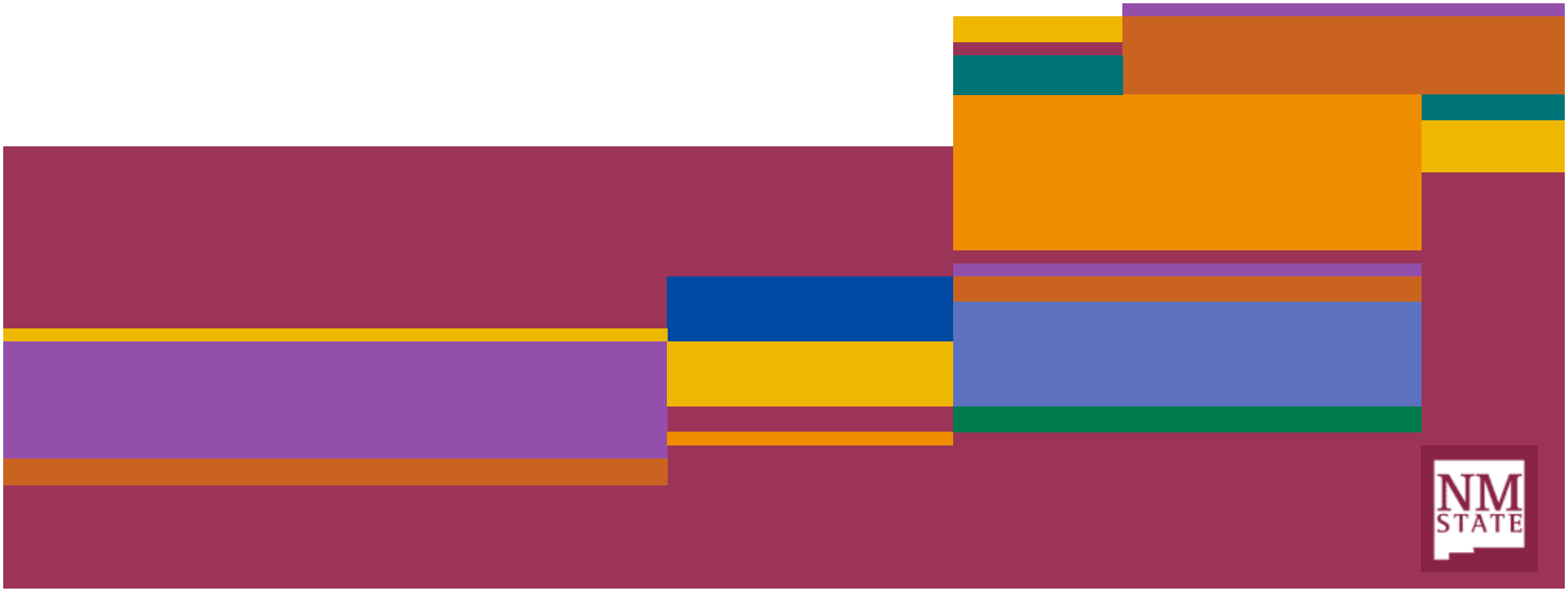
Signal Processing using TMS320C64x Digital Signal Processing Library (DSPLib) Part 2



Categories in DSPLIB

- The routines contained in DSPLib are organized into the following categories:

1. Adaptive filtering
2. Autocorrelation
3. Filtering and convolution
4. Math
5. Matrix/vector operations
6. FFT
7. Miscellaneous

Arguments and Conventions Used

Arguments	Description
x,y	Argument reflecting input data vector.
r	Argument reflecting output data vector
nx,ny,nr	Arguments reflecting the size of vectors x,y, and r, respectively. For functions in the case nx = ny = nr, only nx has been used across
h	Argument reflecting filter coefficient vector (filter routines only)
nh	Argument reflecting the size of vector h
w	Argument reflecting FFT coefficient vector (FFT routines only)

FIR Filter

- void **DSP_fir_gen** (const short * restrict x, const short * restrict h, short * restrict r, int nh, int nr)

  x[nr+nh−1] - Input array of size nr + nh − 1.

  h[nh] - Coefficient array of size nh (coefficients stored in *reverse* order)

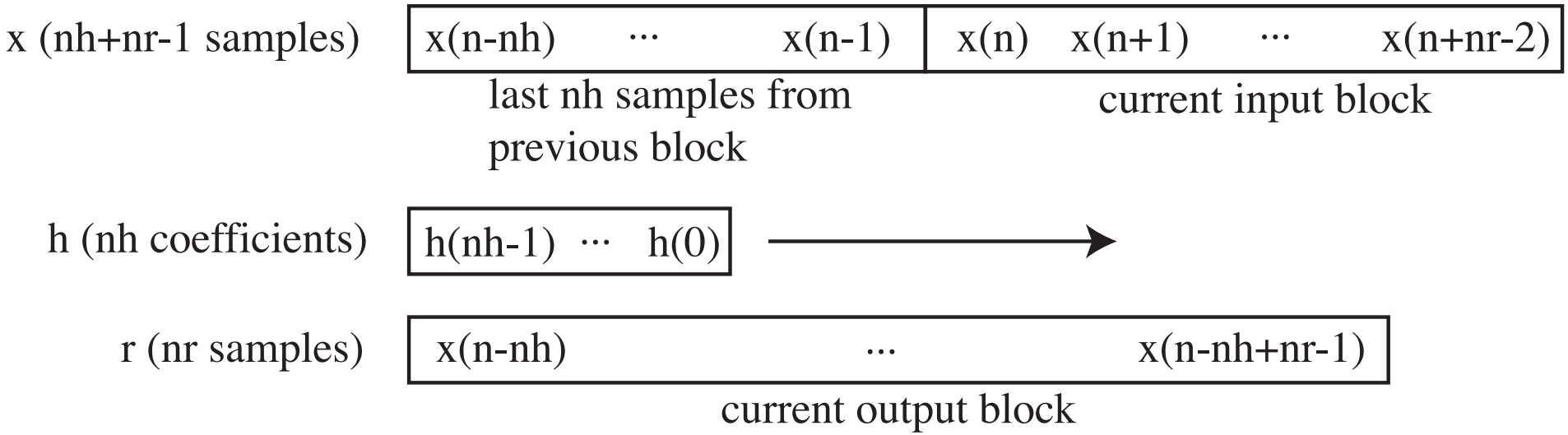
  r[nr] - Output array of size nr (must be word aligned)

  nh - Number of coefficients (must be ≥ 5)

  nr - Number of samples in output array (must be a multiple of 4)

Description: Computes output of an FIR filter using coefficients stored in h. The input is stored in x. The resulting output array is stored in r. Operates on 16-bit data with a 32-bit accumulate. Filter calculates nr output samples using nh coefficients

DSP_fir_gen in Real-Time Implementation



Example 1:

```
/* Non-Realtime FIR Code */
#include "C:\CCStudio\c6400\dsplib\include\dsp_fir_gen.h“
int main (void)
{
int nh=5,nr=4;
short  x[8] =
  {32767,-32767,32767,-32767,32767,-32767,32767,-32767};
short  h[5] = {1,2,3,4,5};
short r[4];
  DSP_fir_gen(x,h,r,nh,nr);
  return 0;
}
```

Symmetric FIR Filter (Linear Phase)

- void **DSP_fir_sym** (const short * restrict x, const short * restrict h, short * restrict r, int nh, int nr, int s)

  x[nr+2*nh] - Input array (size nr + 2*nh) (double-word aligned)

  h[nh+1] - Coefficient array (size nh + 1) (double-word aligned). Coefficients stored in *normal* order and only half (nh+1 out of 2*nh+1) are required

  r[nr] - Output array of size nr (word aligned)

  nh - Number of coefficients (must be multiple of 8). The number of original symmetric coefficients is 2*nh+1

  nr - Number of samples to calculate (must be multiple of 4)

  S - Number of insignificant digits to truncate, e.g., 15 for short (i.e. Q.15 two's complement fractional) input data and coefficients

IIR 4th order (5 coeffs)

- void **DSP_iir** (short * restrict r1, const short * restrict x, short * restrict r2, const short * restrict h2, const short * restrict h1, int nr)

  r1[nr+4] - Output array (used in actual computation-first four elements must have the previous outputs)

  x[nr+4] - Input array

  r2[nr] - Output array (stored)

  h2[5] – Feedforward coefficients

  h1[5] – Feedfeedback coefficients (h1[0] is not used)

  nr - Number of output samples (must be ≥ 8)

Description: 4th order IIR filtering

Example 2:

```
/* Non-Realtime IIR Code */
#include "C:\CCStudio\c6400\dsplib\include\dsp_iir.h“
int main (void)
{
int m,nh=5,nr=8;
short  x[12];
short  h1[5] = {32767,100,200,400,500}; //feedback coeffs
short  h2[5] = {32767,-320,300,400,500}; //feedforward coeffs
short r1[12],r2[8];
for (m=0;m<12;m++)
{x[m]=64; r1[m]=0;}   //assigning the input and zeroing output
   DSP_iir(r1,x,r2,h2,h1,nr); //Call to IIR
   return 0;
}
```

All-pole IIR Lattice Filter

- void **DSP_iirlat**(const short * restrict x, int nx, const short * restrict k, int nk, int * restrict b, short * restrict r)

  x[nx] - Input array

  nx - Length of input

  k[nk] - Reflection coefficients

  nk - Number of reflection coefficients/lattice stages (must be >=4 and multiple of 2 to avoid bank conflicts)

  b[nk+1] - Delay line elements from previous call. Should be initialized to all zeros prior to the first call

  r[nx] Output vector

Other Filter and Convolution Functions

Function & Description
•void **DSP_fir_cplx** (short *x, short *h, short *r, int nh, int nx) Complex FIR Filter (nh is a multiple of 2) •void **DSP_fir_r4** (short *x, short *h, short *r, int nh, int nr) FIR Filter (nh is a multiple of 4) •void **DSP_fir_r8** (short *x, short *h, short *r, int nh, int nr) FIR Filter (nh is a multiple of 8)

Adaptive filtering: LMS FIR

- long **DSP_firlms2**(short * restrict h, const short * restrict x, short b, int nh)

  h[nh] - Coefficient Array

  x[nh+1] - Input Array

  b - Error from previous FIR

  nh - Number of coefficients (must be multiple of 4)

  return long return value

Description: The Least Mean Square adaptive filter computes an update of coefficients by adding weighted error times inputs to the original coefficients. The input array includes the last nh inputs followed by a new single sample input. The coefficient array includes nh coefficients

Other *Miscellaneous* Functions

Function & Description
void **DSP_blk_eswap32**(void *x, void *r, int nx) Endian-swap a block of 32-bit values
void **DSP_blk_eswap64**(void *x, void *r, int nx) Endian-swap a block of 64-bit values
void **DSP_fltoq15** (float *x,short *r, short nx) Float to short (Q.15 two's complement) conversion
int **DSP_minerror** (short *GSP0_TABLE,short *errCoefs, int *savePtr_ret) Minimum Energy Error Search