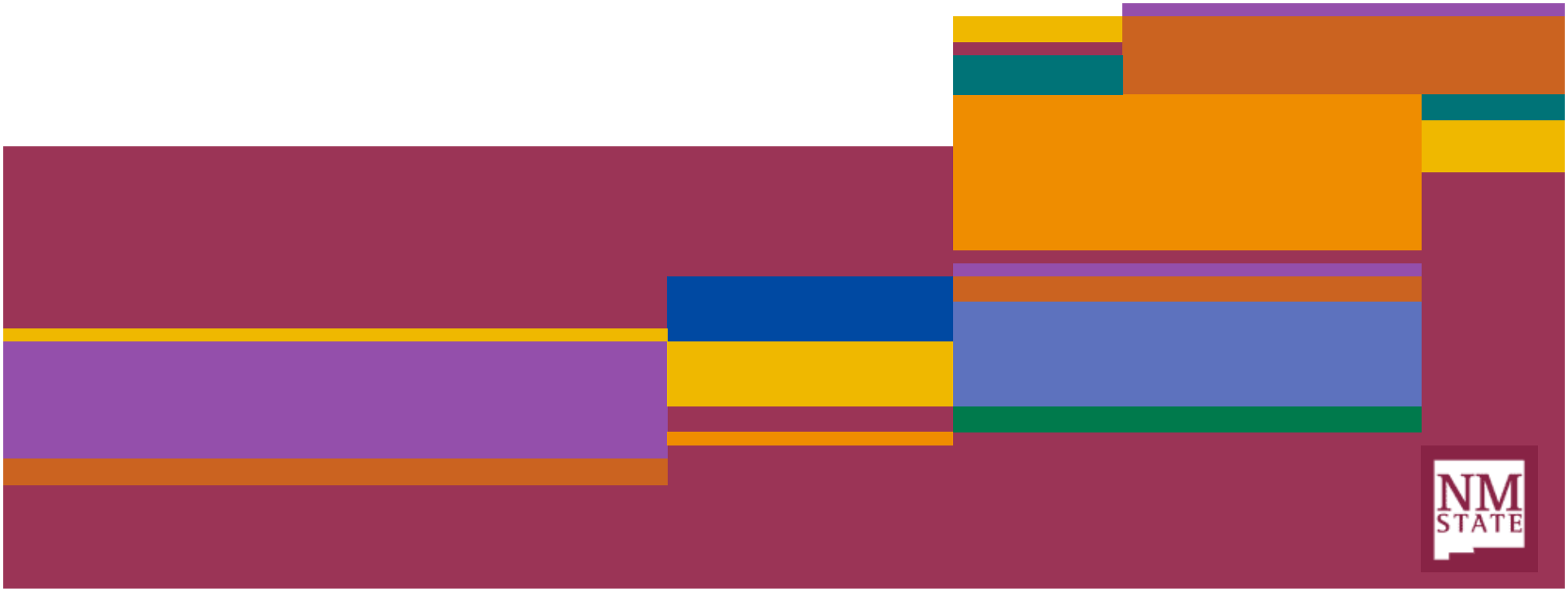
TMS320C6416: Real-Time Processing using the Fast Fourier Transform



Discrete Fourier Transform (DFT)

- DFT

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{kn}, \quad 0 \le k \le N-1$$

where "twiddle factors" are defined as

$$W_N \equiv \exp(-j2\pi/N)$$

and

$$k\text{th DFT frequency} \equiv 2\pi k/N$$

- FFT is a "fast" algorithm for DFT
  - *N* is a power of 2
  - Spectral values are returned in bit-reversed (BR) order

Introduction to C6416 FFT

- DSPLib FFTs are assembly optimized for C62xx and C64xx
- Complex values are stored with Re and Im interleaved

Requirements

- FFT routines require **twiddle factor** tables which can be generated by running tw_fft16x16 in

  C:\CCStudio\c6400\dsplib\support\fft

- Example: Generate array of 2N 16-bit twiddle factors

  ```
  tw_fft16x16 [-s scale] N > tw_16x16xN.h
  ```

  – Scaling of twiddle factors is necessary to prevent overflow of output. Scaling by 2 gives an effective divide by 4 at each FFT stage guaranteeing no overflow

- FFT routines require a **bit-reversal** (BR) table which can be generated by running bitrev_index in

  C:\CCStudio\c6400\dsplib\support\fft

  – Contains indices 0-63 in bit-reversed order

Complex 16x16-bit FFT with Rounding

- void **DSP_fft16x16r**(int nx, short * restrict x, const short * restrict w, const unsigned char * restrict brev, short * restrict y, int radix, int offset, int nmax)

nx -- Length of FFT in complex samples. Must be power of 2 or 4, and ≤16384

x[2*nx] -- Pointer to complex 16-bit, normal-order input array (DW aligned)

w[2*nx] -- Pointer to complex FFT coefficients (twiddle factors) (DW aligned)

brev[64] -- Pointer to bit reverse table containing 64 entries

y[2*nx] -- Pointer to complex 16-bit, normal-order output array (DW aligned)

radix -- Smallest FFT butterfly used in computation used for decomposing FFT into sub-FFTs (4? in our example code)

offset -- Index in complex samples of sub-FFT from start of main FFT (0 in our example)

nmax -- Size of main FFT in complex samples (same as nx in our example)

Complex 16x16-bit FFT with Rounding

Description: This routine implements a cache-optimized complex forward mixed radix FFT with **scaling**, **rounding** and **digit reversal**. Input data x[ ], output data y[ ] and coefficients w[ ] are 16-bit. The output is returned in the separate array y[ ] in **normal order**. Each complex value is stored as **interleaved** 16-bit real and imaginary parts. The code uses a special ordering of FFT coefficients (also called twiddle factors).

Complex Forward FFT with Normally-Ordered Output

- void **DSP_fft** (const short * restrict w, int nx, short * restrict x, short * restrict y)

w[2*nx]	Pointer to vector of Q.15 FFT coefficients of size 2 * nx elements
nx	Number of complex elements in vector x. Must be a power of 4 and $4 \le nx \le 65536$
x[2*nx]	Pointer to input sequence of size 2 * nx elements
y[2*nx]	Pointer to output sequence of size 2 * nx elements

Description: This routine is used to compute an FFT of a complex sequence of size nx, using "decimation-in-frequency." This routine also performs digit reversal as a special last step so output 'y' is returned in normal order instead of bit-reversed order.

Complex 16- x 32-bit FFT with Rounding

- void **DSP_fft16x32**(const short * restrict w, int nx, int * restrict x, int * restrict y)

w[2*nx] Pointer to complex Q.15 FFT coefficients (twiddle factors)

nx Length of FFT in complex samples. Must be power of 2 or 4, and 16 ≤ nx ≤ 32768

x[2*nx] Pointer to complex 32-bit data input.

y[2*nx] Pointer to complex 32-bit data output

Description: This routine computes an extended precision complex forward mixed radix FFT with rounding and digit reversal. Input data x[ ] and output data y[ ] (returned in normal order) are 32-bit, coefficients w[ ] are 16-bit.

Complex 16- x 32-bit IFFT with Rounding

- void **DSP_ifft16x32**(const short * restrict w, int nx, int * restrict x, int * restrict y)

w[2*nx] Pointer to complex Q.15 FFT coefficients

nx Length of FFT in complex samples. Must be power of 2 or 4, and 16 ≤ nx ≤ 32768

x[2*nx] Pointer to complex 32-bit data input

y[2*nx] Pointer to complex 32-bit data output

Other Implementations: One can use fft16x32 to perform IFFT, by conjugating input, performing FFT, and conjugating output. This allows fft16x32 to perform the IFFT as well. However if the double conjugation needs to be avoided then this routine uses the same twiddle factors as the FFT and performs an IFFT.

Other FFT Functions

Function & Description

void DSP_bitrev_cplx (int *x, short *index, int nx)
    Complex Bit-Reverse
void DSP_radix2 (int nx, short *x, short *w)
    Complex Forward FFT (radix 2)
void DSP_r4fft (int nx, short *x, short *w)
    Complex Forward FFT (radix 4)
void DSP_fft(short *w, int nx, short *x, short *y)
    Complex out of place, Forward FFT (radix4) with digit reversal.
These functions are not optimized for C6461 (used for C6200)

Other FFT Functions

Function & Description

- void **DSP_fft16x16t**(short *w, int nx, short *x, short *y)

Mixed radix FFT with truncation, digit reversal, out of place. Input and output: 16 bits, Twiddle factor: 16 bits

- void **DSP_fft32x32**(int *w, int nx, int *x, int *y)

Extended precision, mixed radix FFT, rounding, digit reversal, out of place. Input and output: 32 bits, Twiddle factor: 32 bits

Other FFT Functions

Function & Description
•void **DSP_fft32x32s**(int *w, int nx, int *x, int *y) Extended precision, mixed radix FFT, digit reversal, out of place., with scaling and rounding. Input and output: 32 bits, Twiddle factor: 32 bits • void **DSP_ifft32x32**(int *w, int nx, int *x, int *y) Extended precision, mixed radix IFFT, digit reversal, out of place., with scaling and rounding. Input and output: 32 bits, Twiddle factor: 32 bits

Example: Non-realtime FFT example

```
//This is Non-Real-Time FFT code
#include "costabl.h"
#include "tw_16x16x256.h“
int main (void)
{
short FFTLEN=256;
short y[512];
unsigned char brev[64] = { 0, 32, 16, 48, 8, 40, 24, 56, 4, 36, 20, 52, 12, 44, 28,
    60, 2, 34, 18, 50, 10, 42, 26, 58, 6, 38, 22, 54, 14, 46, 30, 62, 1, 33, 17, 49,
    9, 41, 25, 57, 5, 37, 21, 53, 13, 45, 29, 61, 3, 35, 19, 51, 11, 43, 27, 59, 7,
    39, 23, 55, 15, 47, 31, 63 };
    DSP_fft16x16r( FFTLEN, x, w, brev, y, 4, 0, FFTLEN );
}
```