DSPF_sp_cfftr2_dit *Single-precision floating-point radix-2 FFT with complex input*

Function void DSPF_sp_cfftr2_dit (float * x, float * w, short n)

Arguments

x	Pointer to complex data input.
w	Pointer to complex twiddle factor in bit-reverse order.
n	Length of FFT in complex samples, power of 2 such that n ≥ 32 and n<=32K.

Description

This routine performs the decimation-in-time (DIT) radix-2 FFT of the input array x. x has N complex floating-point numbers arranged as successive real and imaginary number pairs. Input array x contains N complex points (N*2 elements). The coefficients for the FFT are passed to the function in array w which contains N/2 complex numbers (N elements) as successive real and imaginary number pairs. The FFT coefficients w are in N/2 bit-reversed order The elements of input array x are in normal order The assembly routine performs 4 output samples (2 real and 2 imaginary) for a pass through inner loop.

```
How to Use

void main(void)
     {
      gen_w_r2(w, N); // Generate coefficient table
      bit_rev(w, N>>1); // Bit-reverse coefficient table
      DSPF_sp_cfftr2_dit(x, w, N);
                      // input in normal order, output in
                      // order bit-reversed
                      // coefficient table in bit-reversed
                      // order
     }
```

Note that (bit-reversed) coefficients for higher order FFT (1024 point) can be used unchanged as coefficients for a lower order FFT (512, 256, 128 ... ,2) The routine can be used to implement inverse FFT by any one of the following methods:

1) Inputs (x) are replaced by their complex-conjugate values. Output values are divided by N.
2) FFT coefficients (w) are replaced by their complex conjugates. Output values are divided by N.
3) Swap real and imaginary values of input.
4) Swap real and imaginary values of output.

Algorithm

This is the C equivalent of the assembly code. Note that the assembly code is hand optimized and restrictions may apply.

```
void DSPF_sp_cfftr2_dit(float* x, float* w, short n)
{
   short n2, ie, ia, i, j, k, m;
   float rtemp, itemp, c, s;
   n2 = n;
   ie = 1;
   for(k=n; k > 1; k >>= 1)
   {
      n2 >>= 1;
      ia = 0;
      for(j=0; j < ie; j++)
      {
         c = w[2*j];
         s = w[2*j+1];
         for(i=0; i < n2; i++)
         {
            m = ia + n2;
            rtemp     = c * x[2*m]   + s * x[2*m+1];
            itemp     = c * x[2*m+1] - s * x[2*m];
            x[2*m]    = x[2*ia]   - rtemp;
            x[2*m+1]  = x[2*ia+1] - itemp;
            x[2*ia]   = x[2*ia]   + rtemp;
            x[2*ia+1] = x[2*ia+1] + itemp;
            ia++;
         }
         ia += n2;
      }
      ie <<= 1;
   }
}
```

The following C code is used to generate the coefficient table (non-bit reversed).

```
#include <math.h>

/* generate real and imaginary twiddle
   table of size n/2 complex numbers */

gen_w_r2(float* w, int n)
```

```
{
   int i;
   float pi = 4.0*atan(1.0);
   float e = pi*2.0/n;
    for(i=0; i < ( n>>1 ); i++)
    {
       w[2*i]   = cos(i*e);
       w[2*i+1] = sin(i*e);
    }
}
```

The following C code is used to bit reverse the coefficients.

```
bit_rev(float* x, int n)
{
  int i, j, k;
  float rtemp, itemp;
  j = 0;
  for(i=1; i < (n-1); i++)
  {
     k = n >> 1;
     while(k <= j)
     {
        j -= k;
        k >>= 1;
     }
     j += k;
     if(i < j)
     {
        rtemp    = x[j*2];
        x[j*2]   = x[i*2];
        x[i*2]   = rtemp;
        itemp    = x[j*2+1];
        x[j*2+1] = x[i*2+1];
        x[i*2+1] = itemp;
     }
```

```
    }
}
```

Special Requirements

- ❑ n is a integral power of 2 such that n ≥32 and n <=32K.
- ❑ The FFT Coefficients w are in bit-reversed order
- ❑ The elements of input array x are in normal order
- ❑ The imaginary coefficients of w are negated as {cos(d*0), sin(d*0), cos(d*1), sin(d*1) ...} as opposed to the normal sequence of {cos(d*0), −sin(d*0), cos(d*1), −sin(d*1) ...} where d = 2*PI/n.
- ❑ x and w are double-word aligned.

Implementation Notes

- ❑ The two inner loops are combined into one inner loop whose loop count is n/2.
- ❑ The prolog has been completely merged with the epilog. But this gives rise to a problem which has not been overcome. The problem is that the minimum trip count is 32. The safe trip count is at least 16 bound by the size of the epilog. In addition because of merging the prolog and the epilog a data dependency via memory is caused which forces n to be at least 32.
- ❑ The bit-reversed twiddle factor array w can be generated by using the tw_r2fft function provided in the dsplib\support\fft directory or by running tw_r2fft.exe provided in dsplib\bin. The twiddle factor array can also be generated by using gen_w_r2 and bit_rev algorithms as described above.
- ❑ The function bit_rev in dsplib\support\fft can be used to bit reverse the output array to convert it into normal order.
- ❑ ***Endianness:*** This code is little endian.
- ❑ ***Interruptibility:*** This code is interrupt-tolerant but not interruptible.

Benchmarks

Cycles	(2 * n * log(base−2) n) + 42 For n = 64, Cycles = 810
Code size (in bytes)	1248