Calculating the 4096 real samples FFT using LEA.

The function fft_fixed_q15_4096 calculates an FFT with the length of 4096 real samples using several standard LEA functions. It is not possible to use the standard LEA FFT function for the FFT with the length of 4096 complex samples because there is not enough memory in the LEA.
For a proper operation, input samples should not be placed in one-by-one sequence. They should be placed in the special "bit-reverse for arrays" order. The easiest way to do this is to put the input sample to the right place in the input samples array during reading samples from the adc, sensor, etc.

The function fft_fixed_q15_4096 calculates the FFT of a real signal using the method of calculating a real-FFT throught a complex-FFT.
During a calculation process the function performs the following actions :

1) It rearranges the 4096 real input signal to the 2048 complex signal.
   The 1st real sample - is the real part of the 1st complex sample
   The 2nd real sample - is the imaginary part of the 1st complex sample
   The 3rd real sample - is the real part of the 2nd complex sample
   The 4th real sample - is the imaginary part of the 2nd complex sample
   etc.
2) It calculates 4 complex FFTs with the length of 512 complex samples using the standard LEA function
3) It calculates 2 stages of the "butterfly" operation :
   The 1st stage - it is a conversion of 4 complex arrays with the length of 512 complex samples to 2 complex arrays of the length of 1024 complex samples
   The 2nd stage - it is a conversion of 2 complex arrays with the length of 1024 complex samples to 1 complex array of the length of 2048 complex samples
4) It calculates a "split" operation over the 2048 complex samples array to obtain the output array of complex FFT magnitudes. After the "split" operation the output array consists of 2048 complex FFT magnitudes.

```
#include <msp430.h>

#include "target.h"
#include "config.h"
#include "HAL.h"
#include "string.h"

#include "Boards/E11/E11_InitPins/E11_InitPins.h"
#include "Boards/E11/E11_FR5994/E11_FR5994_TimerA.h"

#include "DSPLib.h"
#include "math.h"
#include "complex.h"
#include "QmathLib.h"

#include "TwiddleTable_4096_Q15.h"
#include "TwiddleTable_2048_Q15.h"
#include "TwiddleTable_1024_Q15.h"
#include "SplitTable_4096_Q15.h"

#include "fft_fixed_q15_4096_lea.h"

static void memcpydma(   u16 DMAchannel,
                         u16* Dst,
                         u16* Src,
                         u16 LengthWords);
static void memcpydma32( u16 DMAchannel,
                         u16* Dst,
                         u32* Src32,
                         u16 LengthWords);
static void FFTglueLea(  int16_t* FFTeven,
                         int16_t* W,
                         int16_t* FFTodd,
                         uint16_t LengthOut);
static void FFTglueBaseOp128Lea(   _q15* EVEN_CMPLX,
                                   _q15* W_CMPLX,
                                   _q15* ODD_CMPLX);
static void FFTsplitLeaMspDspLib(_q15* X_CMPLX, _q15* W_CMPLX);
static void CmplxArrayReverse(int16_t* CmplxArray, uint16_t CmplxLength);

//Split-off a real and an imaginary part
#define RE(x)           (((x)<<1)+0)    //Imaginary
#define IM(x)           (((x)<<1)+1)    //Real

//DMA channel
#define DMA_CHANNEL_MEM  0
#define DMA_TIMEOUT      200000

//LEA defines
#define   LENGTH_REAL_SAMPLES           4096
#define   LENGTH_CMPLX_SAMPLES          (LENGTH_REAL_SAMPLES/2)
#define   LEA_FFT_LEN_CMPLX_SAMPLES     512
```

```
#define FFT_FIXED_Q15_4096_N_PART_I0    4
#define PART_NUM          (LENGTH_REAL_SAMPLES/(2*LEA_FFT_LEN_CMPLX_SAMPLES))

//FFT
#define LEA_CMPLX_FFT_ADDR          0x3000

//GLUE
#define LEA_VECT_LEN_CMPLX_SAMPLES 128
#define LEA_EVEN_LENGTH_BYTES       (CMPLX_LEN_BYTES * LEA_VECT_LEN_CMPLX_SAMPLES)
#define LEA_ODD_LENGTH_BYTES        (CMPLX_LEN_BYTES * LEA_VECT_LEN_CMPLX_SAMPLES)
#define LEA_W_LENGTH_BYTES          (CMPLX_LEN_BYTES * LEA_VECT_LEN_CMPLX_SAMPLES)
#define LEA_FFTLO_LENGTH_BYTES      (CMPLX_LEN_BYTES * LEA_VECT_LEN_CMPLX_SAMPLES)
#define LEA_FFTHI_LENGTH_BYTES      (CMPLX_LEN_BYTES * LEA_VECT_LEN_CMPLX_SAMPLES)
#define LEA_ACC_LENGTH_BYTES        (CMPLX_LEN_BYTES * LEA_VECT_LEN_CMPLX_SAMPLES)

#define LEA_BASE_ADDR               0x2C00
#define LEA_EVEN_ADDR               (LEA_BASE_ADDR)
#define LEA_W_ADDR                  (LEA_EVEN_ADDR      +LEA_EVEN_LENGTH_BYTES)
#define LEA_ODD_ADDR                (LEA_W_ADDR         +LEA_W_LENGTH_BYTES)
#define LEA_FFTLO_ADDR              (LEA_ODD_ADDR       +LEA_ODD_LENGTH_BYTES)
#define LEA_FFTHI_ADDR              (LEA_FFTLO_ADDR     +LEA_FFTLO_LENGTH_BYTES)
#define LEA_ACC_ADDR                (LEA_FFTHI_ADDR     +LEA_FFTHI_LENGTH_BYTES)

//SPLIT
#define LEA_X_LO_LENGTH_BYTES       (CMPLX_LEN_BYTES * LEA_VECT_LEN_CMPLX_SAMPLES)
#define LEA_X_HI_LENGTH_BYTES       (CMPLX_LEN_BYTES * LEA_VECT_LEN_CMPLX_SAMPLES)
#define LEA_A_LENGTH_BYTES          (CMPLX_LEN_BYTES * LEA_VECT_LEN_CMPLX_SAMPLES)
#define LEA_B_LENGTH_BYTES          (CMPLX_LEN_BYTES * LEA_VECT_LEN_CMPLX_SAMPLES)
#define LEA_G_LO_LENGTH_BYTES       (CMPLX_LEN_BYTES * LEA_VECT_LEN_CMPLX_SAMPLES)
#define LEA_G_HI_LENGTH_BYTES       (CMPLX_LEN_BYTES * LEA_VECT_LEN_CMPLX_SAMPLES)

#define LEA_X_LO_ADDR               (LEA_BASE_ADDR)
#define LEA_X_HI_ADDR               (LEA_X_LO_ADDR+LEA_X_LO_LENGTH_BYTES)
#define LEA_A_ADDR                  (LEA_X_HI_ADDR+LEA_X_HI_LENGTH_BYTES)
#define LEA_B_ADDR                  (LEA_A_ADDR+LEA_A_LENGTH_BYTES)
#define LEA_G_ADDR                  (LEA_B_ADDR+LEA_B_LENGTH_BYTES)

#define LEA_G_LO_ADDR               (LEA_B_ADDR+LEA_B_LENGTH_BYTES)
#define LEA_G_HI_ADDR               (LEA_G_LO_ADDR+LEA_G_LO_LENGTH_BYTES)
#define LEA_WG_ADDR                 (LEA_G_HI_ADDR+LEA_G_HI_LENGTH_BYTES)
#define X_LENGTH_CMPLX_SAMPLES      2048

//MATH
#define PI                          3.1415926536
#define Q15_PLUS_UNITY              32767
#define Q15_MINUS_UNITY             -32768

//MISC
#define CMPLX_LEN_BYTES  4
#define CMPLX_LEN_WORDS  2
```

```
//Calculate FFT q15 using LEA
msp_status fft_fixed_q15_4096(int16_t *src)  {
     uint16_t i = 0;

     msp_status status;
     msp_cmplx_fft_q15_params cmplx_fft_q15_params;

#ifdef FFT_FIXED_Q15_4096_BENCHMARK
     uint32_t cycleCount = 0;          //For debugging
     uint16_t cycleCountLo = 0;        //For debugging
     uint16_t cycleCountHi = 0;        //For debugging
#endif


     //----------Calculate partial FFT--------
#ifdef FFT_FIXED_Q15_4096_BENCHMARK
     printf("--------FFT--------\n\r");
     msp_benchmarkStart(MSP_BENCHMARK_BASE, 16);
#endif

     for(i=0; i<PART_NUM; i++)  {
          //Copy the partial array to the LEARAM
          memcpydma(     DMA_CHANNEL_MEM,
                         (u16*) LEA_CMPLX_FFT_ADDR,
                         (u16*) src[i*CMPLX_LEN_WORDS*LEA_FFT_LEN_CMPLX_SAMPLES],
                         CMPLX_LEN_WORDS*LEA_FFT_LEN_CMPLX_SAMPLES);

          //Calculate the FFT of the partial array
          cmplx_fft_q15_params.length = LEA_FFT_LEN_CMPLX_SAMPLES;
          cmplx_fft_q15_params.bitReverse = true;
          cmplx_fft_q15_params.twiddleTable = 0;
          status = MAP_msp_cmplx_fft_fixed_q15(&cmplx_fft_q15_params,
                                               (int16_t*) LEA_CMPLX_FFT_ADDR);
          msp_checkStatus(status);

          //Write the result back to the FRAM
          //void memcpydma(   u16 DMAchannel,
          //                  u16* Dst,
          //                  u16* Src,
          //                  u16 LengthWords)
          memcpydma(     DMA_CHANNEL_MEM,
                         (u16*)&src[i*CMPLX_LEN_WORDS*LEA_FFT_LEN_CMPLX_SAMPLES],
                         (u16*) LEA_CMPLX_FFT_ADDR,
                         CMPLX_LEN_WORDS*LEA_FFT_LEN_CMPLX_SAMPLES);
     }

#ifdef FFT_FIXED_Q15_4096_BENCHMARK
     cycle Count = msp_benchmarkStop(MSP_BENCHMARK_BASE);
     cycleCountHi = ((uint32_t)cycleCount / (uint32_t)10000);
     cycleCountLo = ((uint32_t)cycleCount % (uint32_t)10000);
     printf("cycleCount = %04u%04u\n\r", cycleCountHi, cycleCountLo);
#endif
```

```
        //----------Glue 1st stage--------
#ifdef FFT_FIXED_Q15_4096_BENCHMARK
    printf("--------Glue 1st stage (2 glues)--------\n\r");
    msp_benchmarkStart(MSP_BENCHMARK_BASE, 16);
#endif

    //Input  - 4 arrays of 512 complex samples
    //Output - 2 arrays of 1024 complex samples
    //Summary 2 glues

    //1st glue
    FFTglueLea(    &src[0*CMPLX_LEN_WORDS*LEA_FFT_LEN_CMPLX_SAMPLES],//FFTeven,
                   (int16_t*) W_1024_Q15,                            //W,
                   &src[1*CMPLX_LEN_WORDS*LEA_FFT_LEN_CMPLX_SAMPLES],//FFTodd,
                   1024);                                            //LengthOut

    //2nd glue
    FFTglueLea(    &src[2*CMPLX_LEN_WORDS*LEA_FFT_LEN_CMPLX_SAMPLES],//FFTeven,
                   (int16_t*) W_1024_Q15,                            //W,
                   &src[3*CMPLX_LEN_WORDS*LEA_FFT_LEN_CMPLX_SAMPLES],//FFTodd,
                   1024);                                            //LengthOut

#ifdef FFT_FIXED_Q15_4096_BENCHMARK
    cycleCount = msp_benchmarkStop(MSP_BENCHMARK_BASE);
    cycleCountHi = ((uint32_t)cycleCount / (uint32_t)10000);
    cycleCountLo = ((uint32_t)cycleCount % (uint32_t)10000);
    printf("cycleCount = %04u%04u\n\r", cycleCountHi, cycleCountLo);
#endif

    //----------Glue 2nd stage--------
#ifdef FFT_FIXED_Q15_4096_BENCHMARK
    printf("--------Glue 2nd stage (1 glue)--------\n\r");
    msp_benchmarkStart(MSP_BENCHMARK_BASE, 16);
#endif

    //Input  - 2 arrays of 2048 complex samples
    //Output - 1 array of  4096 complex samples
    //Summary 2 glues

    //1st glue
    FFTglueLea(
    &src[0*CMPLX_LEN_WORDS*2*LEA_FFT_LEN_CMPLX_SAMPLES],//FFTeven,
                   (int16_t*) W_2048_Q15,               //W,
                   &src[1*CMPLX_LEN_WORDS*2*LEA_FFT_LEN_CMPLX_SAMPLES],//FFTodd,
                   2048);                               //LengthOut

#ifdef FFT_FIXED_Q15_4096_BENCHMARK
    cycleCount = msp_benchmarkStop(MSP_BENCHMARK_BASE);
    cycleCountHi = ((uint32_t)cycleCount / (uint32_t)10000);
    cycleCountLo = ((uint32_t)cycleCount % (uint32_t)10000);
```

```
    printf("cycleCount = %04u%04u\n\r", cycleCountHi, cycleCountLo);
#endif

    //----------split--------
#ifdef FFT_FIXED_Q15_4096_BENCHMARK
    printf("--------Split--------\n\r");
    msp_benchmarkStart(MSP_BENCHMARK_BASE, 16);
#endif

    FFTsplitLeaMspDspLib(src,                          //_q15* X_CMPLX,
                         (_q15*) W_4096_Q15);          //_q15* W_CMPLX);

#ifdef FFT_FIXED_Q15_4096_BENCHMARK
    cycleCount = msp_benchmarkStop(MSP_BENCHMARK_BASE);
    cycleCountHi = ((uint32_t)cycleCount / (uint32_t)10000);
    cycleCountLo = ((uint32_t)cycleCount % (uint32_t)10000);
    printf("cycleCount = %04u%04u\n\r", cycleCountHi, cycleCountLo);
#endif

    return MSP_SUCCESS;
}



//DMA events trigger initialization
void DMA_Init_LEA( uint16_t DmaChannel )  {
    DMA->uControl = DMA_Read_Modify_Write_Disable_bit;

    DMA_Set_Channel_Trigger( DmaChannel, DMA_Trigger_DMAREQ  );
}



//Copy data MEM->MEM using DMA
void memcpydma(     u16 DMAchannel,
                    u16* Dst,
                    u16* Src,
                    u16 LengthWords)  {
    u16 i = 0;

    //DMA channel initialization
    DMA->Channel[DMAchannel].Transfer_Size = LengthWords;
    DMA->Channel[DMAchannel].Source_Address16 = LoAddress( Src );
    DMA->Channel[DMAchannel].Destination_Address16 = LoAddress( Dst );
    DMA->Channel[DMAchannel].uControl =
        DMA_Block_Transfer_Bits                     |
        DMA_Destination_Address_Incremented_Bits    |
        DMA_Source_Address_Incremented_Bits         |
        DMA_Channel_Destination_Word_Transfer_bit   |
        DMA_Channel_Source_Word_Transfer_bit        |
        DMA_Channel_Level_Sensitive_bit             |
        DMA_Channel_Enable_bit                      |
```

```
            //DMA_Channel_Interrupt_Enable_bit          |
            DMA_Channel_Start_bit                       ;


      //Wait for DMA to finish
      for(i=0; i<DMA_TIMEOUT; i++)  {
            if(DMA->Channel[DMAchannel].Control.Interrupt_Flag==1)  {
                  break;
            }
      }
}



//Copy data FRAM2->MEM using DMA
static void memcpydma32( u16 DMAchannel,
                         u16* Dst,
                         u32* Src32,
                         u16 LengthWords)  {
      u16 i = 0;

      //DMA channel initialization
      DMA->Channel[DMAchannel].Transfer_Size = LengthWords;
      DMA->Channel[DMAchannel].Source_Address32 = Src32;
      DMA->Channel[DMAchannel].Destination_Address16 = LoAddress( Dst );
      DMA->Channel[DMAchannel].uControl =
            DMA_Block_Transfer_Bits                     |
            DMA_Destination_Address_Incremented_Bits    |
            DMA_Source_Address_Incremented_Bits         |
            DMA_Channel_Destination_Word_Transfer_bit   |
            DMA_Channel_Source_Word_Transfer_bit        |
            DMA_Channel_Level_Sensitive_bit             |
            DMA_Channel_Enable_bit                      |

            //DMA_Channel_Interrupt_Enable_bit          |

            DMA_Channel_Start_bit                       ;


      //Wait for DMA to finish
      for(i=0; i<DMA_TIMEOUT; i++)  {
            if(DMA->Channel[DMAchannel].Control.Interrupt_Flag==1)  {
                  break;
            }
      }
}
```

```
//Merge even and odd partial arrays (inplace)
//FFTeven   - even part of complex samples FFT
//FFTodd    - odd part of complex samples FFT
//W         - rotate multipliers
//LengthOut - glue length (complex samples)
//Result :
//FFTeven <- FFTeven+W.*FFTodd;
//FFTodd  <- FFTeven-W.*FFTodd;
static void FFTglueLea(  int16_t* FFTeven,
                         int16_t* W,
                         int16_t* FFTodd,
                         uint16_t LengthOut)  {
     uint16_t i = 0;
     uint16_t BaseOpNum = LengthOut/(2*LEA_VECT_LEN_CMPLX_SAMPLES);

     for(i=0; i<BaseOpNum; i++)  {
          FFTglueBaseOp128Lea(&FFTeven[ i*                        //EVEN_CMPLX,
                                        CMPLX_LEN_WORDS *
                                        LEA_VECT_LEN_CMPLX_SAMPLES],
                                        &W[i*                     //W_CMPLX,
                                           CMPLX_LEN_WORDS *
                                           LEA_VECT_LEN_CMPLX_SAMPLES],
                                        &FFTodd[i*                //ODD_CMPLX);
                                                CMPLX_LEN_WORDS *
                                                LEA_VECT_LEN_CMPLX_SAMPLES]);
     }
}


//Base operation of the glue using LEA
//Glue of 128 complex samples
//EVEN+W*ODD
//EVEN-W*ODD
static void FFTglueBaseOp128Lea(   _q15* EVEN_CMPLX,
                                   _q15* W_CMPLX,
                                   _q15* ODD_CMPLX)  {
     msp_cmplx_mpy_q15_params      cmplx_mpy_q15_params;
     msp_cmplx_add_q15_params      cmplx_add_q15_params;
     msp_cmplx_sub_q15_params      cmplx_sub_q15_params;
     msp_cmplx_shift_q15_params    cmplx_shift_q15_params;
     msp_status status;

     uint16_t SamplesNum = LEA_VECT_LEN_CMPLX_SAMPLES;

     //Pointers to memory arrays in LEA
     _q15* p_even_lea_15      = (_q15*) LEA_EVEN_ADDR;
     _q15* p_w_lea_15         = (_q15*) LEA_W_ADDR;
     _q15* p_odd_lea_15       = (_q15*) LEA_ODD_ADDR;
     _q15* p_fftlo_lea_15     = (_q15*) LEA_FFTLO_ADDR;
     _q15* p_ffthi_lea_15     = (_q15*) LEA_FFTHI_ADDR;
     _q15* p_acc_lea_15       = (_q15*) LEA_ACC_ADDR;
```

```
//-----------Copy data from FRAM, FRAM2 to LEARAM----------
//EVEN
memcpydma(      DMA_CHANNEL_MEM,     //u16 DMAchannel,
                (u16*)p_even_lea_15,//u16* Dst,
                (u16*)EVEN_CMPLX,   //u16* Src,
                //An amount of words to copy
                CMPLX_LEN_WORDS*LEA_VECT_LEN_CMPLX_SAMPLES);//u16 Length);

//W
memcpydma32(DMA_CHANNEL_MEM,        //u16 DMAchannel,
                (u16*)p_w_lea_15,   //u16* Dst,
                (u32*)W_CMPLX,      //u32* Src32,
                //An amount of words to copy
                CMPLX_LEN_WORDS*
                LEA_VECT_LEN_CMPLX_SAMPLES);  //u16 LengthWords);

//ODD
memcpydma(      DMA_CHANNEL_MEM,     //u16 DMAchannel,
                (u16*)p_odd_lea_15, //u16* Dst,
                (u16*)ODD_CMPLX,    //u16* Src,
                //An amount of words to copy
                CMPLX_LEN_WORDS*LEA_VECT_LEN_CMPLX_SAMPLES);//u16 Length);

//-----------Calculate the 1st part of output samples----------
//FFTLO=EVEN+W*ODD
//ACC=W.*ODD
cmplx_mpy_q15_params.length = SamplesNum;
status = msp_cmplx_mpy_q15    (&cmplx_mpy_q15_params,  //params,
                               p_w_lea_15,             //srcA,
                               p_odd_lea_15,           //srcB,
                               p_acc_lea_15);          //dst
msp_checkStatus(status);

//FFTLO=EVEN+W*ODD
cmplx_add_q15_params.length = SamplesNum;
msp_cmplx_add_q15 ( &cmplx_add_q15_params,   //params,
                    p_even_lea_15,           //srcA,
                    p_acc_lea_15,            //srcB,
                    p_fftlo_lea_15);         //dst
msp_checkStatus(status);

//Divide FFTLO by 2
cmplx_shift_q15_params.length = SamplesNum;
cmplx_shift_q15_params.shift = -1;
cmplx_shift_q15_params.conjugate = 0;
msp_cmplx_shift_q15 (    &cmplx_shift_q15_params, //params,
                         p_fftlo_lea_15,          //src,
                         p_fftlo_lea_15);         //dst

//-----------Calculate the 2nd part of output samples----------
//FFTHI=EVEN-W*ODD
cmplx_sub_q15_params.length = SamplesNum;
```

```
    msp_cmplx_sub_q15 ( &cmplx_sub_q15_params,   //params,
                        p_even_lea_15,           //srcA,
                        p_acc_lea_15,            //srcB,
                        p_ffthi_lea_15);         //dst
    msp_checkStatus(status);

    //Divide FFTLO by 2
    cmplx_shift_q15_params.length = SamplesNum;
    cmplx_shift_q15_params.shift = -1;
    cmplx_shift_q15_params.conjugate = 0;
    msp_cmplx_shift_q15 (&cmplx_shift_q15_params, //params,
                         p_ffthi_lea_15,          //src,
                         p_ffthi_lea_15);         //dst

    //-------Copy the result vectors back to the FRAM---------
    //EVEN_CMPLX<-fftlo
    memcpydma(     DMA_CHANNEL_MEM,          //u16 DMAchannel,
                   (u16*)EVEN_CMPLX,         //u16* Dst,
                   (u16*)p_fftlo_lea_15,     //u16* Src,
                   CMPLX_LEN_WORDS*LEA_VECT_LEN_CMPLX_SAMPLES);//u16 Length);

    //ODD_CMPLX<-ffthi
    memcpydma(     DMA_CHANNEL_MEM,          //u16 DMAchannel,
                   (u16*)ODD_CMPLX,          //u16* Dst,
                   (u16*)p_ffthi_lea_15,     //u16* Src,
                   CMPLX_LEN_WORDS*LEA_VECT_LEN_CMPLX_SAMPLES);//u16 Length);
}



//split operation using LEA

//G(k)  =0.5*[X(k)+X*(N-K)]-0.5*j*e^(-j*2*pi*k/(2*N))[X(k)-X*(N-k)];
//G(N-k)=0.5*[X(k)+X*(N-K)]*-0.5*j*e^(-j*2*pi*k/(2*N))[X(k)-X*(N-k)]*;
//Computational algorithm:
//1)A(k)=0.5*[X(k)+X*(N-K)];
//2)B(k)=0.5*j*e^(-j*2*pi*k/(2*N))[X(k)-X*(N-k)];
//3)G(k)=A(k)-jB(k);          <- inplace
//4)G(N-k)=A*(k)-jB*(k);      <- inplace
static void FFTsplitLeaMspDspLib(_q15* X_CMPLX, _q15* W_CMPLX)  {
    msp_cmplx_mpy_q15_params     cmplx_mpy_q15_params;
    msp_cmplx_add_q15_params     cmplx_add_q15_params;
    msp_cmplx_sub_q15_params     cmplx_sub_q15_params;
    msp_cmplx_conj_q15_params    cmplx_conj_q15_params;
    msp_cmplx_shift_q15_params   cmplx_shift_q15_params;
    msp_cmplx_fill_q15_params    cmplx_fill_q15_params;
    msp_status status;

    uint16_t SamplesNum = LEA_VECT_LEN_CMPLX_SAMPLES;
    uint16_t k = 0;
    uint16_t i_hi_start = 0;
    uint16_t i_hi_end = 0;
```

```
uint16_t BaseOpNum = 0;

//Pointers to memory arrays in LEA
_q15* p_x_lo_15                = (_q15*) LEA_X_LO_ADDR;
_q15* p_x_hi_15          = (_q15*) LEA_X_HI_ADDR;
_q15* p_a_15             = (_q15*) LEA_A_ADDR;
_q15* p_b_15             = (_q15*) LEA_B_ADDR;
_q15* p_w_15             = (_q15*) LEA_WG_ADDR;

BaseOpNum = X_LENGTH_CMPLX_SAMPLES/LEA_VECT_LEN_CMPLX_SAMPLES;
BaseOpNum = BaseOpNum/2;
for(k=0; k<BaseOpNum; k++)  {
     //-----------Copy data to LEA----------
     //X_LO
     memcpydma(     DMA_CHANNEL_MEM,              //u16 DMAchannel,
                    (u16*)p_x_lo_15,              //u16* Dst,
                    (u16*)&(X_CMPLX[k*
                                   CMPLX_LEN_WORDS*
                                   LEA_VECT_LEN_CMPLX_SAMPLES]), //u16* Src,
                    //An amount of words to copy
                    CMPLX_LEN_WORDS*
                    LEA_VECT_LEN_CMPLX_SAMPLES);  //u16 Length);

     //W
     memcpydma32(DMA_CHANNEL_MEM,                 //u16 DMAchannel,
                    (u16*)p_w_15,                 //u16* Dst,
                    (u32*)&(W_CMPLX[k*
                                   CMPLX_LEN_WORDS*
                                   LEA_VECT_LEN_CMPLX_SAMPLES]),//u32*Src32,
                    //An amount of words to copy
                    CMPLX_LEN_WORDS*
                    LEA_VECT_LEN_CMPLX_SAMPLES);  //u16 LengthWords);

     //X_HI
     if(k==0)  {
          i_hi_start = X_LENGTH_CMPLX_SAMPLES-
                       (k+1)*LEA_VECT_LEN_CMPLX_SAMPLES+1;
          i_hi_start = CMPLX_LEN_WORDS*i_hi_start;
          i_hi_end = LEA_VECT_LEN_CMPLX_SAMPLES-1;
          memcpydma(DMA_CHANNEL_MEM,                  //u16 DMAchannel,
                       (u16*)p_x_hi_15,               //u16* Dst,
                       (u16*)&(X_CMPLX[i_hi_start]),  //u16* Src,
                       //An amount of words to copy
                       CMPLX_LEN_WORDS*
                       (LEA_VECT_LEN_CMPLX_SAMPLES-1)); //u16 LengthWords);
          p_x_hi_15[RE(i_hi_end)] = X_CMPLX[RE(0)];
          p_x_hi_15[IM(i_hi_end)] = X_CMPLX[IM(0)];
     }
     else  {
          i_hi_start = X_LENGTH_CMPLX_SAMPLES-
                       (k+1)*LEA_VECT_LEN_CMPLX_SAMPLES+1;
          i_hi_start = CMPLX_LEN_WORDS*i_hi_start;
```

```
    memcpydma(DMA_CHANNEL_MEM,                  //u16 DMAchannel,
              (u16*)p_x_hi_15,                  //u16* Dst,
              (u16*)&(X_CMPLX[i_hi_start]),     //u16* Src,
              //An amount of words to copy
              CMPLX_LEN_WORDS*
              LEA_VECT_LEN_CMPLX_SAMPLES);      //u16 LengthWords);
}

//--------------A(k)=0.5*[X(k)+X*(N-k)]-------------
//p_x_hi_15=X(N-k)
CmplxArrayReverse(p_x_hi_15, LEA_VECT_LEN_CMPLX_SAMPLES);

//p_x_hi_15=X*(N-k)
cmplx_conj_q15_params.length = SamplesNum;
status = msp_cmplx_conj_q15 (&cmplx_conj_q15_params,//params,
                                  p_x_hi_15,       //src,
                                  p_x_hi_15);      //dst)
msp_checkStatus(status);

//p_a_15=X(k)+X*(N-k)
cmplx_add_q15_params.length = SamplesNum;
status = msp_cmplx_add_q15 (&cmplx_add_q15_params,    //params,
                                p_x_lo_15,            //srcA,
                                p_x_hi_15,            //srcB,
                                p_a_15);              //dst
msp_checkStatus(status);

//p_a_15=0.5*[X(k)+X*(N-k)]
cmplx_shift_q15_params.length = SamplesNum;
cmplx_shift_q15_params.shift = -1;
cmplx_shift_q15_params.conjugate = 0;
status = msp_cmplx_shift_q15 (&cmplx_shift_q15_params,//params,
                              p_a_15,                 //src,
                              p_a_15);                //dst
msp_checkStatus(status);


//--------------B(k)=0.5*W(k)*[X(k)-X*(N-k)]-------------
//p_b_15=[X(k)-X*(N-k)]
cmplx_sub_q15_params.length = SamplesNum;
status = msp_cmplx_sub_q15 ( &cmplx_sub_q15_params,   //params,
                              p_x_lo_15,              //srcA,
                              p_x_hi_15,              //srcB,
                              p_b_15);                //dst
msp_checkStatus(status);

//p_b_15=W(k)*[X(k)-X*(N-k)]
cmplx_mpy_q15_params.length = SamplesNum;
status = msp_cmplx_mpy_q15    (&cmplx_mpy_q15_params, //params,
                              p_w_15,                 //srcA,
                              p_b_15,                 //srcB,
                              p_b_15);                //dst
```

```
msp_checkStatus(status);

//p_b_15=0.5*W(k)*[X(k)-X*(N-k)]
cmplx_shift_q15_params.length = SamplesNum;
cmplx_shift_q15_params.shift = -1;
cmplx_shift_q15_params.conjugate = 0;
status = msp_cmplx_shift_q15 (&cmplx_shift_q15_params,//params,
                              p_b_15,                 //src,
                              p_b_15);                //dst
msp_checkStatus(status);


//--------------G(k)=A(k)+(0-j)*B(k)-----------
//p_x_lo_15 = (0-j)
cmplx_fill_q15_params.length = SamplesNum;
cmplx_fill_q15_params.realValue = 0;
cmplx_fill_q15_params.imagValue = Q15_MINUS_UNITY;
status = msp_cmplx_fill_q15(  &cmplx_fill_q15_params,  //params,
                              p_x_lo_15);              //dst
msp_checkStatus(status);

//p_x_lo_15=(0-j)*B(k)
cmplx_mpy_q15_params.length = SamplesNum;
status = msp_cmplx_mpy_q15    (&cmplx_mpy_q15_params,  //params,
                               p_x_lo_15,              //srcA,
                               p_b_15,                 //srcB,
                               p_x_lo_15);             //dst
msp_checkStatus(status);

//p_x_lo_15=A(k)+(0-j)*B(k)
cmplx_add_q15_params.length = SamplesNum;
status = msp_cmplx_add_q15 (  &cmplx_add_q15_params,   //params,
                              p_a_15,                  //srcA,
                              p_x_lo_15,               //srcB,
                              p_x_lo_15);              //dst
msp_checkStatus(status);

//p_x_lo_15=0.5*p_x_lo_15
cmplx_shift_q15_params.length = SamplesNum;
cmplx_shift_q15_params.shift = -1;
cmplx_shift_q15_params.conjugate = 0;
status = msp_cmplx_shift_q15 (&cmplx_shift_q15_params,//params,
                              p_x_lo_15,              //src,
                              p_x_lo_15);             //dst
msp_checkStatus(status);

//--------------G(N-k)=A*(k)+(0-j)*B*(k)-----------
//p_a_15 = A*(k)
cmplx_conj_q15_params.length = SamplesNum;
status = msp_cmplx_conj_q15 (&cmplx_conj_q15_params,  //params,
                             p_a_15,                  //src,
                             p_a_15);                 //dst
```

```
msp_checkStatus(status);

//p_b_15 = B*(k)
cmplx_conj_q15_params.length = SamplesNum;
status = msp_cmplx_conj_q15 (&cmplx_conj_q15_params,  //params,
                             p_b_15,                  //src,
                             p_b_15);                 //dst
msp_checkStatus(status);

//p_x_hi_15=(0-j)
cmplx_fill_q15_params.length = SamplesNum;
cmplx_fill_q15_params.realValue = 0;
cmplx_fill_q15_params.imagValue = Q15_MINUS_UNITY;
status = msp_cmplx_fill_q15(&cmplx_fill_q15_params,   //params,
                             p_x_hi_15);              //dst
msp_checkStatus(status);

//p_x_hi_15=(0-j)*B*(k)
cmplx_mpy_q15_params.length = SamplesNum;
status = msp_cmplx_mpy_q15    (&cmplx_mpy_q15_params, //params,
                               p_x_hi_15,             //srcA,
                               p_b_15,                //srcB,
                               p_x_hi_15);            //dst
msp_checkStatus(status);

//p_x_hi_15=A*(k)+(0-j)*B*(k)
cmplx_add_q15_params.length = SamplesNum;
status = msp_cmplx_add_q15 (  &cmplx_add_q15_params,  //params,
                             p_a_15,                  //srcA,
                             p_x_hi_15,               //srcB,
                             p_x_hi_15);              //dst
msp_checkStatus(status);

//p_x_hi_15=G(N-k)
CmplxArrayReverse(p_x_hi_15, LEA_VECT_LEN_CMPLX_SAMPLES);

//p_x_hi_15=0.5*p_x_hi_15
cmplx_shift_q15_params.length = SamplesNum;
cmplx_shift_q15_params.shift = -1;
cmplx_shift_q15_params.conjugate = 0;
status = msp_cmplx_shift_q15 (&cmplx_shift_q15_params,//params,
                             p_x_hi_15,               //src,
                             p_x_hi_15);              //dst
msp_checkStatus(status);

//-----------------Unload data from LEA-------------
//X_LO
memcpydma(     DMA_CHANNEL_MEM,                      //u16 DMAchannel,
               (u16*)&(X_CMPLX[k*                    //u16* Dst,
                           CMPLX_LEN_WORDS*
                           LEA_VECT_LEN_CMPLX_SAMPLES]),
               (u16*)p_x_lo_15,                      //u16* Src,
```

```
                        //An amount of words to copy
                        CMPLX_LEN_WORDS*
                        LEA_VECT_LEN_CMPLX_SAMPLES);//u16 Length);

        //X_HI
        if(k==0)  {
             i_hi_start = X_LENGTH_CMPLX_SAMPLES-
                          (k+1)*LEA_VECT_LEN_CMPLX_SAMPLES+1;
             i_hi_start = CMPLX_LEN_WORDS*i_hi_start;
             i_hi_end = LEA_VECT_LEN_CMPLX_SAMPLES-1;
             memcpydma(DMA_CHANNEL_MEM,                          //u16 DMAchannel,
                       (u16*)&(X_CMPLX[i_hi_start]),             //u16* Dst,
                       (u16*)p_x_hi_15,                          //u16* Src,
                       //An amount of words to copy
                       CMPLX_LEN_WORDS*
                       (LEA_VECT_LEN_CMPLX_SAMPLES-1)); //u16 LengthWords);
             X_CMPLX[RE(i_hi_end)] = p_x_hi_15[RE(0)];
             X_CMPLX[IM(i_hi_end)] = p_x_hi_15[IM(0)];
        }
        else  {
             i_hi_start = X_LENGTH_CMPLX_SAMPLES-
                          (k+1)*LEA_VECT_LEN_CMPLX_SAMPLES+1;
             i_hi_start = CMPLX_LEN_WORDS*i_hi_start;
             memcpydma(DMA_CHANNEL_MEM,                      //u16 DMAchannel,
                       (u16*)&(X_CMPLX[i_hi_start]),         //u16* Dst,
                       (u16*)p_x_hi_15,                      //u16* Src,
                       //An amount of words to copy
                       CMPLX_LEN_WORDS*
                       LEA_VECT_LEN_CMPLX_SAMPLES);     //u16 LengthWords);
        }
    }

    //------------G(N/2)=X*(N/2)-----------
    k=X_LENGTH_CMPLX_SAMPLES/2;
    X_CMPLX[RE(k)] = X_CMPLX[RE(k)];
    X_CMPLX[IM(k)] = -X_CMPLX[IM(k)];

    //------------G(N/2)=0.5*G(N/2)-----------
    X_CMPLX[RE(k)] >>= 1;
    X_CMPLX[IM(k)] >>= 1;
}
```

```
//Reverse of a complex array
static void CmplxArrayReverse(int16_t* CmplxArray, uint16_t CmplxLength)  {
     uint16_t i = 0;
     uint16_t CmplxLengthHalf = CmplxLength/2;
     int16_t TempRe = 0;
     int16_t TempIm = 0;

     for(i=0; i<CmplxLengthHalf; i++)  {
          TempRe=CmplxArray[RE(i)];
          TempIm=CmplxArray[IM(i)];

          CmplxArray[RE(i)]=CmplxArray[RE(CmplxLength-1-i)];
          CmplxArray[IM(i)]=CmplxArray[IM(CmplxLength-1-i)];

          CmplxArray[RE(CmplxLength-1-i)]=TempRe;
          CmplxArray[IM(CmplxLength-1-i)]=TempIm;
     }
}
```