

HWAFFT

- The HWAFFT is tightly-coupled with the DSP core which means that it is physically located outside of the DSP core but has access to the core's full memory read bandwidth (busses B, C, and D), access to the core's internal registers and accumulators, and access to its address generation units.
- The HWAFFT cannot access the data write busses or memory mapped registers (MMRs). Because the HWAFFT is seen as part of the execution unit of the CPU, it must also comply to the core's pipeline exceptions, and in particular those caused by stalls and conditional execution.
- The HWAFFT supports two stage modes:
 - Double-Stage Mode – two FFT stages performed in each pass
 - Single-Stage Mode – one FFT stage performed in each pass

HWAFFT Software Interface (Data Types)

- The input and output vectors of the HWAFFT contain complex numbers. Each real and imaginary part is represented by a two's complement, 16-bit fixed-point number.
- The most significant bit holds the number's sign value, and the remaining 15 are fraction bits (S16Q15 format). The range of each number is $[-1, 1 - (1/2)^{15}]$.
- Real and imaginary parts appear in an interleaved order within each vector:

Int16 CMPLX_Vec16[2*N] = ...(N = FFT Length)

Real[0]	Imag[0]	Real[1]	Imag[1]	Real[2]	Imag[2]
Bit15,.....,Bit0	Bit15,.....,Bit0	Bit15,.....,Bit0	Bit15,.....,Bit0	Bit15,.....,Bit0	Bit15,.....,Bit0

Int32 CMPLX_Vec32[N] = ...(N = FFT Length)

Real[0]	Imag[0]	Real[1]	Imag[1]	Real[2]	Imag[2]
Bit31,....., Bit16, Bit15,....., Bit0	Bit31,....., Bit16, Bit15,....., Bit0	Bit31,....., Bit16, Bit15,....., Bit0	Bit31,....., Bit16, Bit15,....., Bit0	Bit31,....., Bit16, Bit15,....., Bit0	Bit31,....., Bit16, Bit15,....., Bit0

HWAAFFT Software Interface (*Functions*)

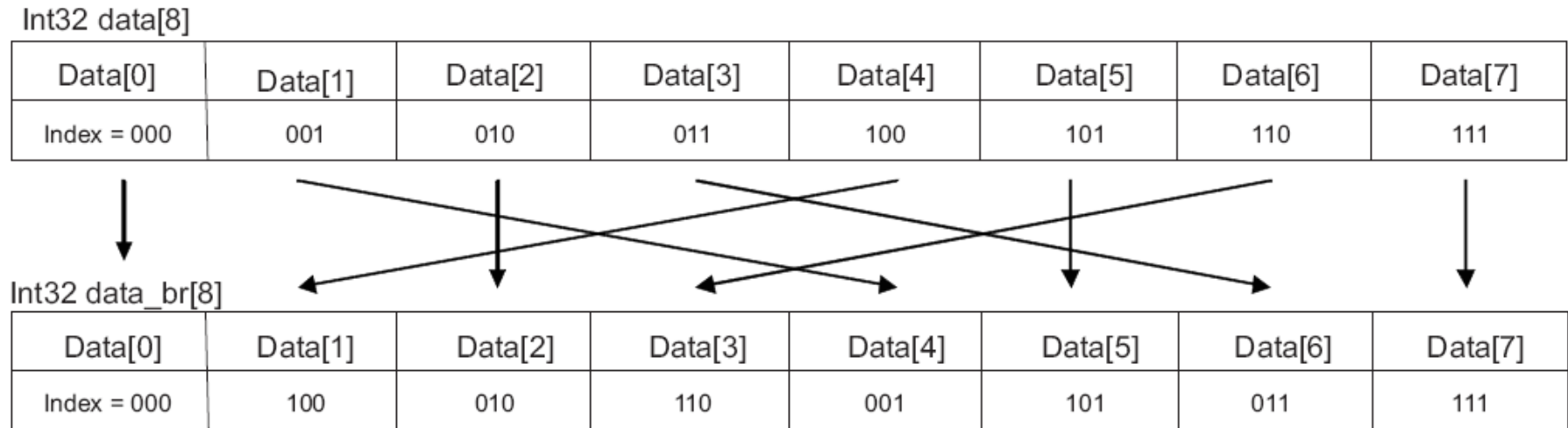
- C-Callable HWAAFFT Functions are provided for computing FFT/IFFT transforms on the HWAAFFT coprocessor.
- These functions contain optimized sequences of coprocessor instructions for computing scaled or unscaled 8-, 16-, 32-, 64-, 128-, 256-, 512-, and 1024-point FFT/IFFTs.
- The HWAAFFT functions are named as follows:

```
Uint16 hwafft_Npts( Performs N-point complex FFT/IFFT, where N = {8, 16, 32, 64, 128, 256, 512, 1024}
Int32 *data,        Input/output – complex vector
Int32 *scratch,     Intermediate/output – complex vector
Uint16 fft_flag,    Flag determines whether FFT or IFFT performed, (0 = FFT, 1 = IFFT)
Uint16 scale_flag   Flag determines whether butterfly output divided by 2 (0 = Scale, 1 = No Scale)
); Return value     Flag determines whether output in data or scratch vector (0 = data, 1 = scratch)
```

- The data and scratch vectors must reside in separate blocks of RAM (DARAM or SARAM) to maximize memory bandwidth.

HWAFFT Software Interface (Functions)

- Before computing the FFT/IFFT on the HWAFFT, the input buffer must be bit-reversed to facilitate a Radix-2 DIT computation. This function contains optimized assembly that executes on the CPU to rearrange the Int32 elements of the input vector by placing each element in the destination vector at the index that corresponds to the bit-reversal of its current index.



```
void hwafft_br(  
    Int32 *data,  
    Int32 *data_br,  
    Uint16 data_len,  
);
```

Performs out-of-place bit-reversal on 32-bit data vector
Input – 32-bit data vector
Output – bit-reversed data vector
Length of complex data vector

Function Descriptions and ROM Locations

Function Name	Description	VC5505 (PG1.4) ROM Address	C5505/C5515 (PG2.0) ROM Address
hwafft_br	Int32 (32-bit) vector bit-reversal, Strict alignment requirement	0x00ff7342	0x00ff6cbe
hwafft_8pts	8-point FFT/IFFT, 1 double-stage, 1 single-stage	0x00ff7356	0x00ff6cd2
hwafft_16pts	16-point FFT/IFFT, 2 double-stages, 0 single-stages	0x00ff7445	0x00ff6dc1
hwafft_32pts	32-point FFT/IFFT, 2 double-stages, 1 single-stage	0x00ff759b	0x00ff6f17
hwafft_64pts	64-point FFT/IFFT, 3 double-stages, 0 single-stages	0x00ff78a4	0x00ff7220
hwafft_128pts	128-point FFT/IFFT, 3 double-stages, 1 single-stage	0x00ff7a39	0x00ff73b5
hwafft_256pts	256-point FFT/IFFT, 4 double-stages, 0 single-stages	0x00ff7c4a	0x00ff75c6
hwafft_512pts	512-point FFT/IFFT, 4 double-stages, 1 single-stage	0x00ff7e48	0x00ff77c4
hwafft_1024pts	1024-point FFT/IFFT, 5 double-stages, 0 single-stages	0x00ff80c2	0x00ff7a3e

Project Configuration for Calling Functions from ROM

- In order to utilize these HWAAFFT routines in ROM, add the following lines to the bottom of the project's linker CMD file and remove the hwafft.asm file from the project (or exclude it from the build).

```
/* C5515 HWAAFFT Routines ROM Addresses: */
```

```
_hwafft_br      = 0x00ff6cd6;
```

```
_hwafft_8pts    = 0x00ff6cea;
```

```
_hwafft_16pts   = 0x00ff6dd9;
```

```
_hwafft_32pts   = 0x00ff6f2f;
```

```
_hwafft_64pts   = 0x00ff7238;
```

```
_hwafft_128pts  = 0x00ff73cd;
```

```
_hwafft_256pts  = 0x00ff75de;
```

```
_hwafft_512pts  = 0x00ff77dc;
```

```
_hwafft_1024pts = 0x00ff7a56;
```

1024-Point FFT Example

```
#define FFT_FLAG      ( 0 )    /* HWAFFT to perform FFT */
#define IFFT_FLAG     ( 1 )    /* HWAFFT to perform IFFT */
#define SCALE_FLAG    ( 0 )    /* HWAFFT to scale butterfly output */
#define NOSCALE_FLAG  ( 1 )    /* HWAFFT not to scale butterfly output */
#define OUT_SEL_DATA  ( 0 )    /* Indicates HWAFFT output located in input data vector */
#define OUT_SEL_SCRATCH ( 1 )  /* Indicates HWAFFT output located in scratch vector */
Int32 *data;
Int32 *data_br;
Uint16 fft_flag;
Uint16 scale_flag;
Int32 *scratch;
Uint16 out_sel;
Int32 *result;
```

Compute 1024-point FFT with Scaling enabled: a $\frac{1}{2}$ scale factor after every stage:

```
fft_flag = FFT_FLAG;
scale_flag = SCALE_FLAG;

data = <1024-point Complex input>;

/* Bit-Reverse 1024-point data, Store into data_br, data_br aligned to
   12-least significant binary zeros*/
hwafft_br(data, data_br, DATA_LEN_1024); /* bit-reverse input data,
                                           Destination buffer aligned */
data = data_br;

/* Compute 1024-point FFT, scaling enabled. */
out_sel = hwafft_1024pts(data, scratch, fft_flag, scale_flag);

if (out_sel == OUT_SEL_DATA) {
    result = data;
}else {
    result = scratch;
}
```

1024-Point IFFT Example

```
#define FFT_FLAG      ( 0 )  /* HWAFFT to perform FFT */
#define IFFT_FLAG     ( 1 )  /* HWAFFT to perform IFFT */
#define SCALE_FLAG    ( 0 )  /* HWAFFT to scale butterfly output */
#define NOSCALE_FLAG  ( 1 )  /* HWAFFT not to scale butterfly output */
#define OUT_SEL_DATA   ( 0 )  /* Indicates HWAFFT output located in input data vector */
#define OUT_SEL_SCRATCH ( 1 ) /* Indicates HWAFFT output located in scratch vector */
Int32 *data;
Int32 *data_br;
Uint16 fft_flag;
Uint16 scale_flag;
Int32 *scratch;
Uint16 out_sel;
Int32 *result;
```

Compute 1024-point IFFT with Scaling disabled:

```
fft_flag = IFFT_FLAG;
scale_flag = NOSCALE_FLAG;

data = <1024-point Complex input>;

/* Bit-Reverse 1024-point data, Store into data_br, data_br aligned to
   12-least significant binary zeros */
hwafft_br(data, data_br, DATA_LEN_1024);
data = data_br;

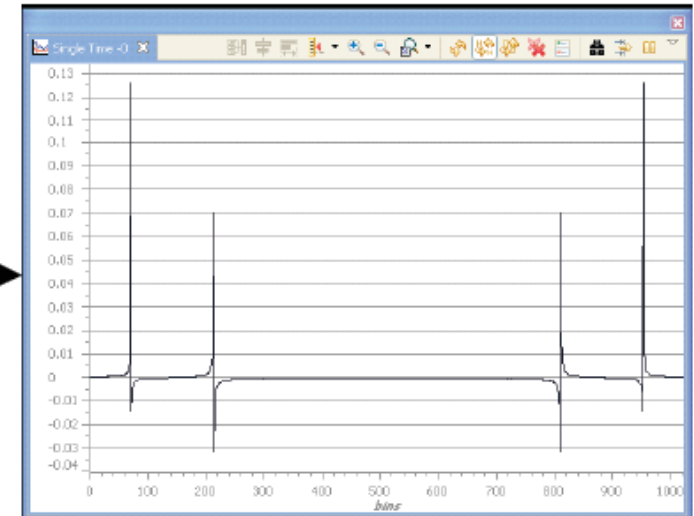
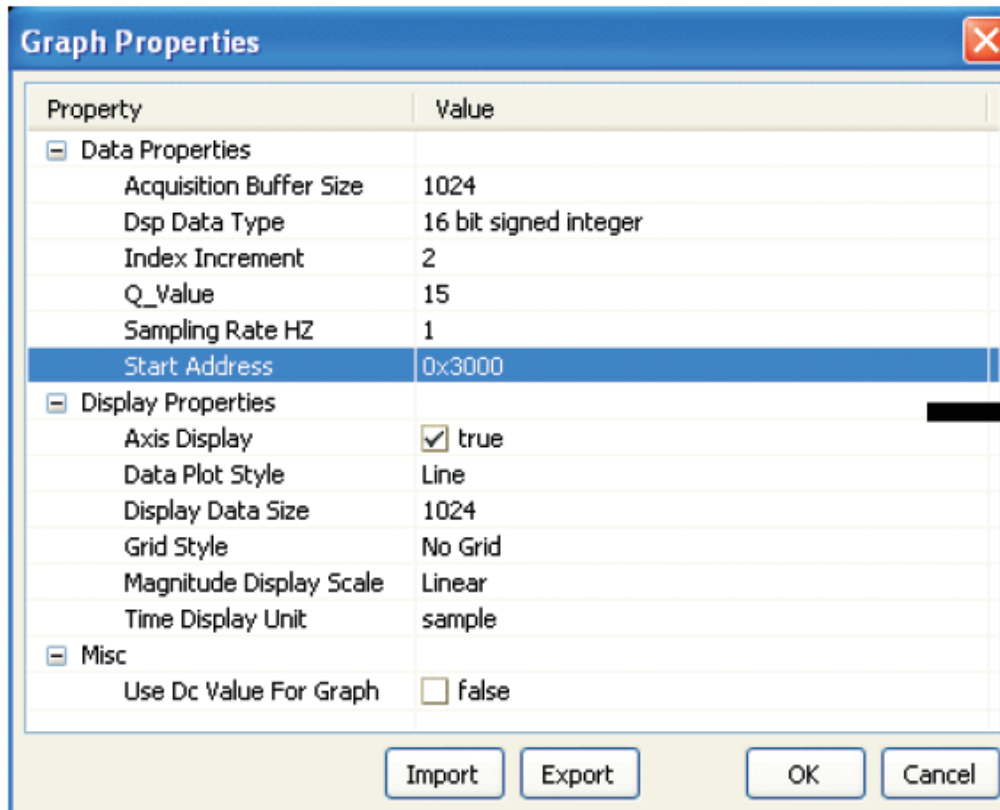
/* Compute 1024-point IFFT, scaling disabled */
out_sel = hwafft_1024pts(data, scratch, fft_flag, scale_flag);

if (out_sel == OUT_SEL_DATA) {
    result = data;
} else {
    result = scratch;
}
```


Graphing FFT Results in CCS4

- Tool → Graph → Single Time

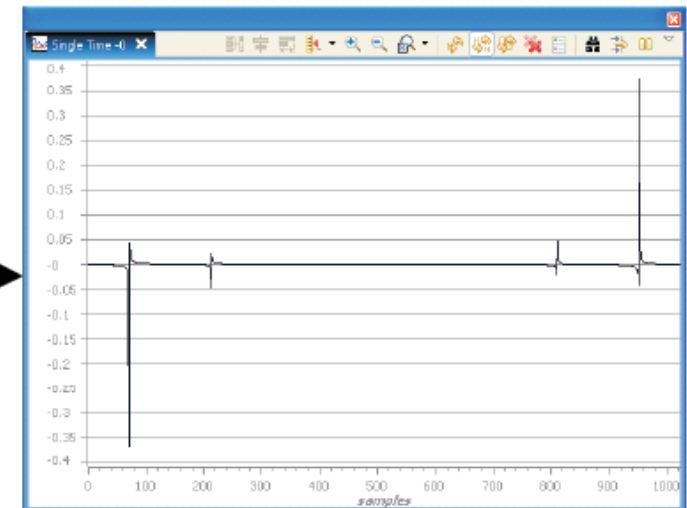
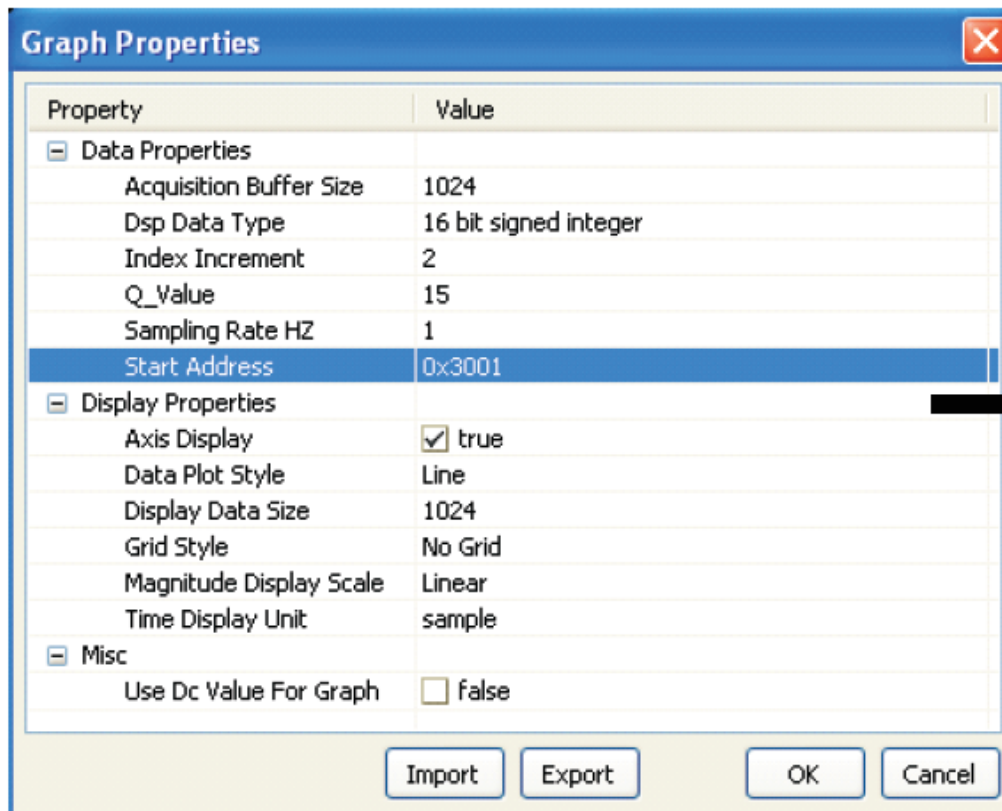
Plot the real part:



Graphing FFT Results in CCS4

- Tool → Graph → Single Time

Plot the imaginary part:



Energy Efficiency of HWAFFT

Complex FFT	FFT with HWA		CPU (Scale)		HWA VS. CPU	
	FFT + BR Cycles	Energy/FFT (nJ/FFT)	FFT + BR Cycles	Energy/FFT (nJ/FFT)	x Times Faster (Scale)	x Times Energy Efficient (Scale)
16 pt	115+55 =170	32.1	344+117=461	157.1	2.7	4.9
64 pt	285+151 =436	98.5	1194+211=1405	531.7	3.2	5.4
256 pt	1133+535=1668	407.2	5404+543=5947	2354.2	3.6	5.8
512 pt	2693+1047=3740	939.7	11829+907=12736	5097.5	3.4	5.4
1024 pt	5244+2071=7315	1836.2	25934+1783=27717	11097.9	3.8	6.0

FFT HWA is 4 ~ 6x more energy efficient and 2.2 ~ 3.8x faster

Note:

1. BR = Bit-Reverse Operation
2. Power measurement Condition: at room temp only, all peripherals are clock gated, measured at Vddc