AccelWare IP cores provide a direct path to hardware implementation for complex MATLAB® toolbox and built-in functions. AccelWare cores deliver synthesizable, pre-verified DSP functions that enable true, top-down MATLAB architectural synthesis of FPGAs and ASICs. AccelWare IP includes Building Block, Advanced Math, Signal Processing and Communications toolkits.

FFT / IFFT Transform

The AccelWare FFT/IFFT Transform is designed for real-time, continuous communication systems. The FFT/IFFT operates from a single external clock.

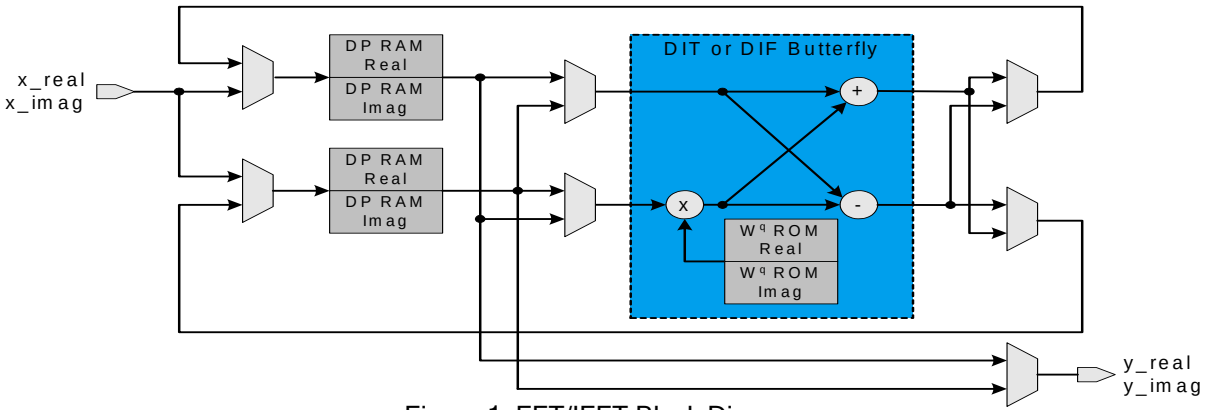

Figure 1: FFT/IFFT Block Diagram

Architecture The AccelWare FFT/IFFT is a minimum-resource implementation of the FFT/IFFT algorithm. A single fly and associated memory are resource-shared to implement each of the necessary stages to perform the entire FFT/IFFT transform. See Figure 1 for a block diagram.

Radix The radix is the logarithmic base of the FFT/IFFT. Two micro-architectures are implemented: radix-2 and radix-4. The radix-4 implementation provides for faster transform times by reducing the number of stages necessary to implement the FFT/IFFT. See Figure 2.

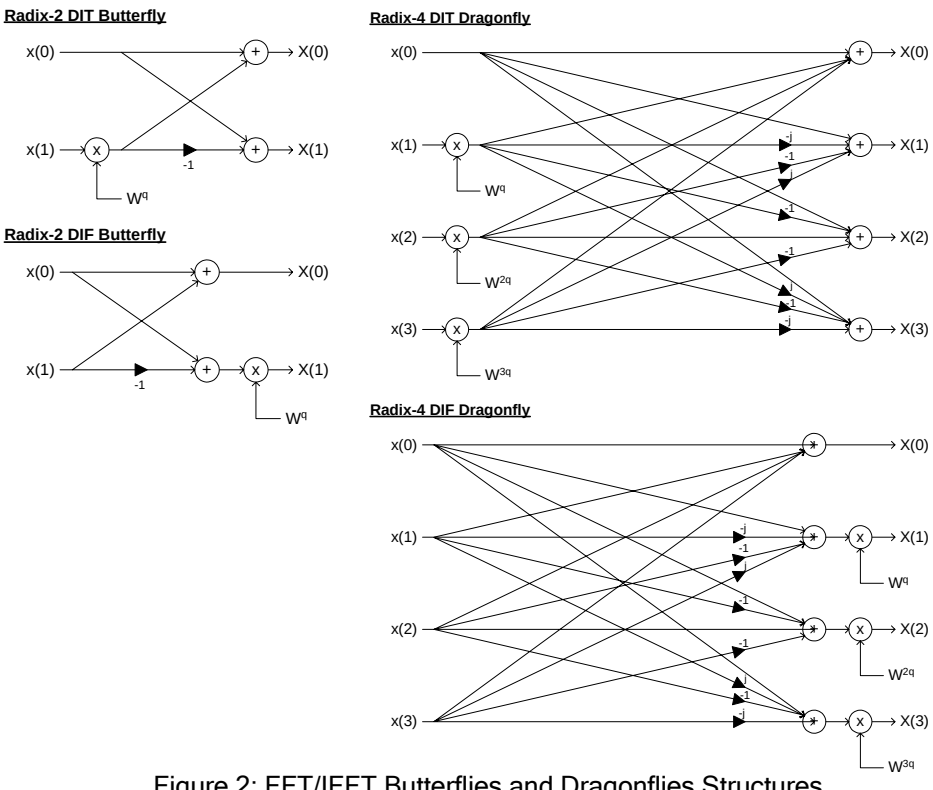

Figure 2: FFT/IFFT Butterflies and Dragonflies Structures

Data I/O Format The I/O format can either be scalar-based (1 sample/clk) or array-based (the entire FFTLength array in a single clk). The array-based I/O format makes comparison with the MATLAB FFT/IFFT very straight-forward. See Figure 3 for timing.

Pipelined Fly For high speed applications the user has the option to select a pipelined fly (for both radix-2 and radix-4 micro-architectures), which inserts pipeline registers into the fly structure.

Scaling For an IFFT to be an inverse of an FFT, the product of the FFT and IFFT scaling must be 1/FFTLength. Optional 1/FFTLength scaling can be implemented by selecting this via the scaling implementation parameter.

Complex Multiplier Two complex multiplier micro-architectures are available: the traditional 4 multiplier / 2 adder structure, and a 3 multiplier / 5 adder structure, providing the user with control over the resources utilized in the target fabric to implement the FFT/IFFT.

Natural Order I/O The decimation algorithm will naturally have FFT/IFFT inputs or outputs in digit/bit reverse ordering; DIF has natural order input and digit/bit reverse output, DIT has digit/bit reverse input and natural order output. The AccelWare FFT/IFFT has an option to force both input and output to be natural order regardless of the decimation algorithm selected.

FFT/IFFT Transform Time The transform time through the AccelWare FFT/IFFT depends upon which radix micro-architecture is selected and whether the pipelined fly is selected. The transform time can be calculated via:

Pipelined Fly = No:

$$TransformTime = \log_{radix}(fftlength) * (fftlength / radix) + 3$$

Pipelined Fly = Yes:

$$TransformTime = \log_{radix}(fftlength) * (fftlength / radix) + 3 + delay$$

delay	Radix 2	Radix4
3mult/5add	7	9
4mult/2add	6	8

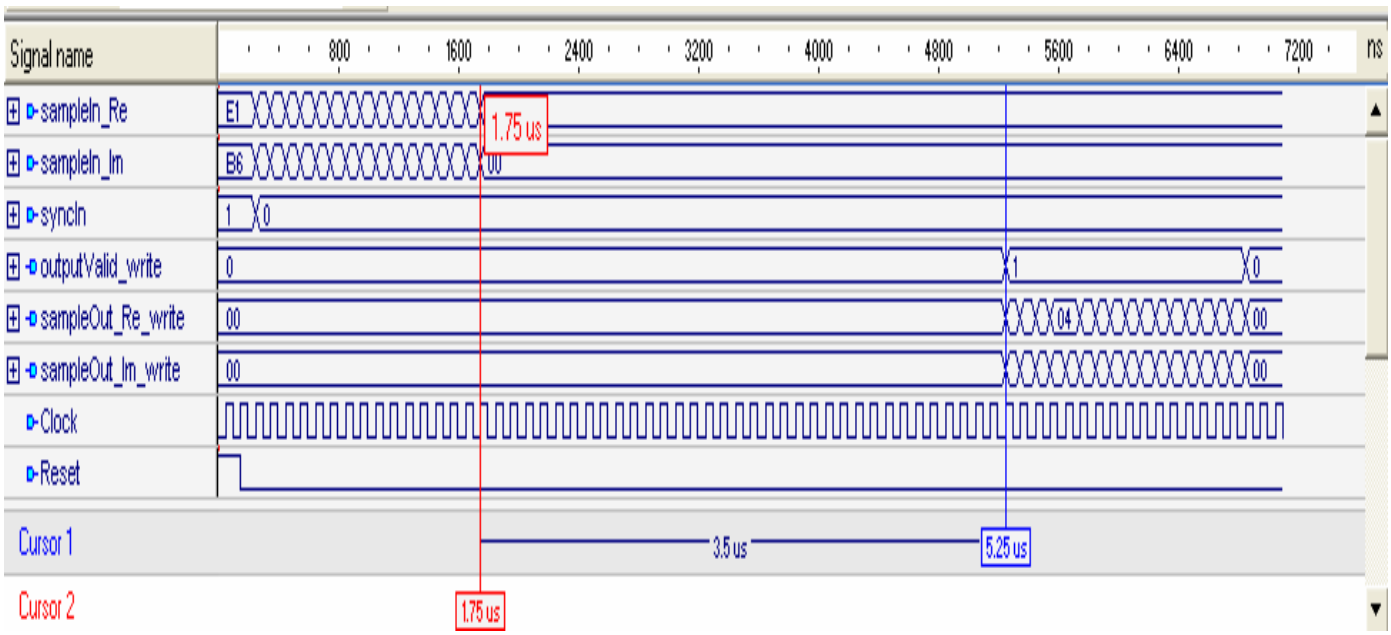

Figure 3: Timing example

Input			
Signal Name	**Signal Description**	**Type**	**Range**
x_real	Real portion of the x input. N distinct samples to which the FFT is applied.	quantized/ fixed-precision	$[-2^{InputDataWidth-1}, 2^{InputDataWidth-1}-1]/2^{InputDataFractWidth}$
x_imag	Imaginary portion of the x input. N distinct samples to which the FFT is applied.	quantized/ fixed-precision	$[-2^{InputDataWidth-1}, 2^{InputDataWidth-1}-1]/2^{InputDataFractWidth}$
synchIn	Pulse signaling start of valid input data.	Binary	[0, 1]

Output			
Signal Name	**Signal Description**	**Type**	**Range**
y_real	Real portion of the y output. N distinct samples as result of FFT operation.	quantized/ fixed-precision	$[-2^{OutputDataWidth-1}, 2^{OutputDataWidth-1}-1]/2^{OutputDataFractWidth}$
y_imag	Imaginary portion of the y output. N distinct samples as result of FFT operation.	quantized/ fixed-precision	$[-2^{OutputDataWidth-1}, 2^{OutputDataWidth-1}-1]/2^{OutputDataFractWidth}$
outputValid	Signals when valid data is on the output ports. outputValid=1 when FFT data is being output.	Binary	[0, 1]

Functional Parameters			
Parameter Name	**Parameter Description**	**Type**	**Range**
FFTLength	Number of differential points in the FFT; same as 'N' for the Matlab fft model. Must satisfy condition: N = $Radix^M$, M Î Z^+.	Positive Integer	[8, 65536]

Implementation Parameters			
Parameter Name	**Parameter Description**	**Type**	**Range**
DataIOFormat	Input and output data can be brought in and out 1 sample at a time (Scalar) or the entire array, that is FFTLength long, can be brought in and out.	String	[Scalar, Array]
DecimationAlgorithm	Decimation algorithm to be used: Frequency (DIF) or Time (DIT).	String	[Frequency, Time]
Radix	The logarithmic base for the FFT/IFFT algorithm.	Integer	[2, 4]
NaturalOrderIO	Decimation algorithm will naturally have either inputs or outputs in digit/bit reverse ordering; DIF has natural order input and digit/bit reverse output, DIT has digit/bit reverse input and natural order output. 'yes' will force input and output to be natural order regardless of decimation type.	String	[no, yes]
Scaling	1/FFTLength scaling can be implemented. For an IFFT to be an inverse of an FFT, the product of the FFT and IFFT scaling must be 1/FFTLength.	String	[no, yes]
ComplexMultiplier	Complex multiplier micro-architectures are available.	String	[3 Multiplier / 5 Adder, 4 Multiplier / 2 Adder]
PipelinedFly	For high speed applications setting this to 'yes' will insert pipelined registers in the 'fly' operator. Setting to 'no' a lower speed implementation will result.	String	[no, yes]
InputDataWidth	Total number of bits used to represent the input x.	Positive Integer	[4, 32]
InputDataFractWidth	Number of bits used to represent fractional part of input wordwidth.	Positive Integer	[1, InputDataWidth-1]
TwiddleWidth	Number of bits used to represent phase factors.	Positive Integer	[4, 32]

Related MATLAB Function

```
y = fft(x, N);
```

AccelWare Function Call Syntax

[outputValid y_real y_imag] = fft_xxx[1] (x_real, x_imag, syncIn); within the context of the sample-based streaming loop construct, e.g.:

Example

```
% Streaming loop
for K=0:7168
        % AW function
        [outputValid(K) SO_re(K) SO_im(K)] = fft_001(SI_re(K),SI_im(K),syncIn(K));
end
```

Internally, processing occurs in three major states:
(1) *input data*, (1024 iterations)[2]
(2) *process data* (5120 iterations)[2]
(3) *output data* (1024 iterations)[2]
which accounts for '2*N+1' iterations on the 'K' streaming counter.

[1] xxx will be a unique number that is assigned to the model.
[2] iterations given are for a 1024-point FFT; pipelining and complex multiplier architecture will add 6-7 more iterations.

Supported MATLAB Syntax

y = fft(x, N);

Unsupported MATLAB Syntax

y=fft(x);
y-fft(x, [ ], dim);
y=fft(x), N, dim);

Differences in Operation between AccelWare FFT/IFFT and MATLAB fft()/ifft()

In MATLAB the input <x> is a complex vector of dimension '1×N' passed to fft() in parallel with the output <y> occurring as a complex vector of dimension '1×N' in parallel as well. Since this is not practical in hardware, in fft_xxx() the input <x> occurs as a serial stream of 'input wordwidth'-bit symbols and the output <y> occurs as serial stream of 'output wordwidth'-bit symbols. The 'syncIn' signal indicates to the fft_xxx() the start of serially streamed input data. The 'outputValid' when valid data is available at the data outputs; 'outputValid' will serially output 'N' 1's. In addition, complex numbers are not currently supported and consequently the complex vectors <x>, <y> are represented as <x_real, x_imag>, <y_real, y_imag > respectively.

MATLAB supports an fft(x) function call which infers the number of points (N) from the input x. This operation is not supported in AccelWare - the number of points must be specified explicitly in the parameters.

MATLAB supports a dimension parameter 'dim' which allows the user to specify which dimension of x to apply the FFT operation to. The AccelWare fft() will not support this feature since it is not possible to implement in hardware.

Ordering Information

The AccelWare FFT/IFFT block is included in the AccelWare Signal Processing Toolkit (AccelChip part number **AWSPT**) and is provided as an option to the AccelChip DSP Synthesis product (AccelChip part number **ACDSP**). For further information on availability, contact your local AccelChip sales representative or send email to sales@accelchip.com.

AccelChip Incorporated
1900 McCarthy Blvd., Suite. 204, Milpitas, CA 95035 phone (408) 943 0700 option 1 fax (408) 943 0661