



RF Engines Limited



Document Title **RFEL Fixed point radix 2 FFT model for Matlab**

Document Ref H03004

Tel +44 (0)1983 550330

Fax +44 (0)1983 550340

E-mail info@rfel.com

Web www.rfel.com

Filename H03004-Fixed Point FFT Radix 2 Matlab Model.doc

Approved

Issue	1.00				
Date	25 May 03				
Author					
Technical					
Commercial					
QA					

Copyright RF Engines Limited 2003

All Rights reserved

This document is issued by RF Engines Limited (hereinafter called RFEL) in confidence, and is not to be reproduced in whole or in part without the prior written permission of RFEL. The information contained herein is the property of RFEL and is to be used only for the purpose for which it is submitted. None of this document's contents can be disclosed to a third party without the prior written permission of RFEL.

Document History

Revision No	Date Issued	Issued by	Summary of changes
1.00	25 May 2003	PW	Initial

Contents

1	What is it?	3
2	Who are we?	4
3	Installation.	5
4	Using the model.	5
4.1	Initialisation.....	5
4.2	Runtime	6
4.3	Clear.....	7
4.4	Help.....	7
4.5	Version	7
5	Examples	7
5.1	Fixed and Floating Point Comparison	7
5.2	Bit Width Comparison.....	8
5.3	Signals with a Frequency Offset.....	9

1 What is it?

This document describes the RF Engines Ltd (RFEL) Fixed Point Radix 2 FFT Model for Matlab. The version of the software considered here is the fixed point version – which uses fixed point arithmetic throughout and has customisable bit-widths.

The model is designed to accurately represent the functionality of RFEL Vectis FFT cores. It has been tested against RFEL's Vectis FFT implementation and shown to be bit-true provided the modelled bit-widths are correctly set to avoid overflow.

The model is designed to be fully parameterisable, to encompass the widest range of possible requirements. Many of the modelled configurations are provided by RFEL as optimised off-the-shelf FPGA cores (see Who are we?); cores may also be tailor-made to meet higher specification requirements.

The technical description of the model is given in Table 1.

Parameter	Specification
FFT Stages ¹	Integers from 1 to 25. The upper limit also depends on available memory. <i>Note: If you have installed only an evaluation copy of this software the available number of FFT stages will be restricted. A full installation of the model provides FFTs with fully parameterisable sizes.</i>
Bin ordering ²	Bins may be output in bit-reversed order (i.e. no output bit-reverser), or normally ordered, with the startBin (first bin output) as a parameter. Setting the startBin to -1 provides bit-reversed output, otherwise the following values are given: startBin = 0: output order = [0, 1, 2, ..., N-1] startBin = 1: output order = [1, 2, ..., N-1, 0] startBin = N/2: output order = [N/2, ..., N-1, 0, 1, ..., N/2-1] Where N is the number of FFT bins. The most common settings are -1, 0 or N/2.
FFT Precision	Fully customisable at input and output and at all intermediate stages between 1 and 62 bits. See note on Data Precision.
Twiddle Precision	Customisable between 1 and 62 bits. See note on Data Precision.
Processing type	Block Processing
Block Input Size	Equal to the FFT Length
Block Output Size	Equal to the FFT Length
Pipeline Delay	None
FFT Type	Radix-2 Decimate-in-frequency.

¹ The maximum number of FFT points supported by the Vectis FFT range is 2¹⁷. This is probably the practical limit to fit on a single FPGA. Larger FFTs may be provided to order.

² The Vectis FFT range currently provides as options: bit-reversed, normally ordered and N/2 shifted, corresponding respectively to startBin = -1, startBin = 0 and startBin = N/2. Other output orders may be provided to order.

Data Precision ³	Up to 62 bits of fixed point precision is supported throughout. Intermediate sums and products must not exceed 62 bits precision.
Scaling	Scaling is provided at the output of the FFT. This enables the designer to eliminate MSBs which will never be toggled by a real signal and hence minimise the bit growth through the model.
Overflow Detection	Overflow detection is optionally provided to test whether the signal is in range at various points through the model. This is invaluable in verification that the scalings have been correctly specified.
Platforms	Matlab for Windows. Tested on Windows 2000, Matlab version 6.5.

Table 1: Model Specification

The FFT model is implemented as a fully optimised MEX add-in for Matlab and can therefore be called from Matlab in the same way as any other function. Use of the model involves three stages, initialisation, processing and destruction.

- At initialisation, the user specifies the model parameters (FFT size, Decimation Factor, Bin Ordering and all appropriate bit widths) and the function returns a handle. This handle is then used in all future processing.
- During runtime, the user supplies input data blocks to the model and retrieves the output of the FFT.
- At destruction, the user calls clear on the model, which then deallocates all associated memory.

Use of the model is described in detail in Section 4.

Use of the model is described in detail in Section 4.

2 Who are we?

RF Engines Limited specialises in advanced digital signal processing hardware designs for high quality signal filtering and conditioning. Utilising patented architectures and high-speed Field Programmable Gate array (FPGA) devices, the company supplies off-the-shelf or specialist system solutions for advanced electronic products in the defence, instrumentation and communications markets worldwide.

Based on the Isle of Wight in the UK, the company provides:

standard designs sold as Intellectual Property Cores,

complete turn-key designs developed against a particular specification, and

consultancy services to establish suitable architectures for specific tasks.

For more information go to: <http://www.rfel.com/>

For a list of available products, go to: <http://www.rfel.com/products/Products.asp>

³ The bit widths used through the core will limit the size of the FFT that will fit on a particular FPGA device. Signals with up to up to 17 bits may be multiplied by a single 18-bit multiplier and up to 34 bits may be multiplied using two 18-bit multipliers. Therefore it is in the interest of the designer to keep the bit widths as low as possible to meet a given performance.

Or contact us to discuss your needs at: info@rfel.com

3 Installation.

To install the FFT Software Model, run the installation file and follow the onscreen instructions. The model should be installed to the required directory and then this directory should be placed on the Matlab Path.

4 Using the model.

As noted above, use of the model involves the three steps of initialisation, processing and destruction. This enables blocks of data to be continuously “streamed” into the model, rather than passing in a single input and getting a single output.

The model is shown graphically in Figure 1; all parameters are double precision numbers (i.e. the Matlab standard data type) unless otherwise noted. The length of the input and output vectors is $K = 2^{\text{ORDER}}$.

Some example code is given in Section 5.

During runtime, the model takes inputs and supplies outputs as Matlab’s int64 type. This permits fixed point bit widths of up to 62 bits to be supported.

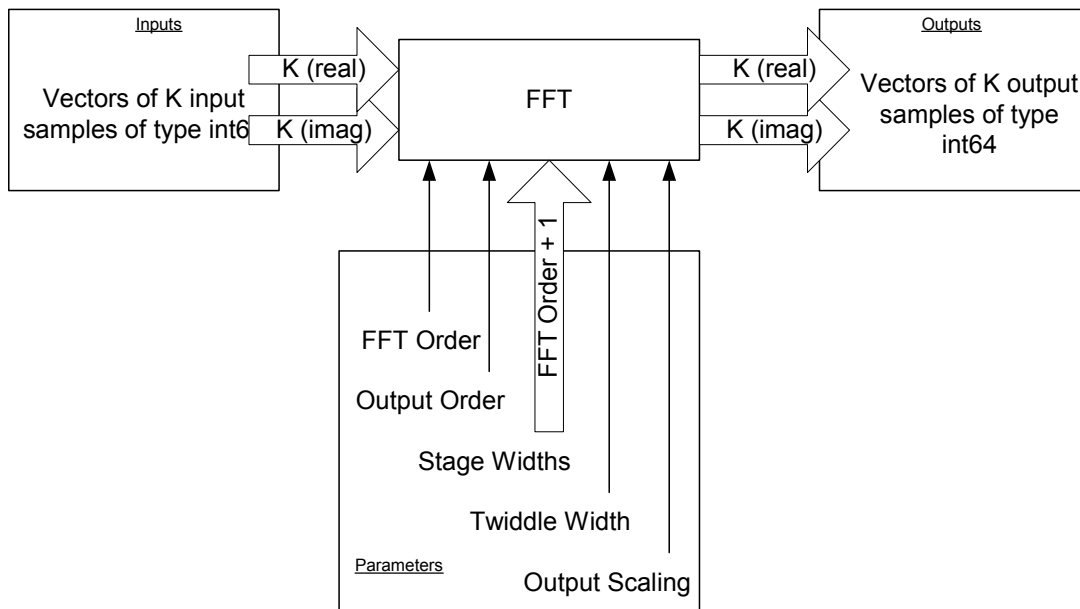


Figure 1: Fixed Point FFT Inputs, Outputs and Parameters

The model is designed to accurately represent the functionality of RFEL FPGA cores. It has been tested against RFEL's standard implementation and shown to be bit-true provided the modelled bit-widths are correctly set to avoid overflow.

The model is designed to be fully parameterisable, to encompass the widest range of possible requirements. Many of the modelled configurations are provided by RFEL as optimised off-the-shelf FPGA cores. Cores may also be tailor-made to meet higher specification requirements.

4.1 Initialisation

The constructor carries all the required parameters.

At each stage the outputs are resized according to the bit-widths specified in STAGEWIDTHS. The vector STAGEWIDTHS is order + 1 in length; the zeroth

element is the input bit width, the first element is the bit width at the output of the first stage, and so on.

The twiddles are generated for the appropriate DFT order and quantised according to the twiddle width. The constructor does some basic error-checking on the inputs, instantiates a new object with the appropriate parameters. A unique handle that may be used to identify the object is passed back to Matlab. If construction of the object fails, an error message is provided for the user.

```
[HANDLE] = xpFft2Mat(STAGES, STARTBIN, STAGEWIDTHS, TWIDDLEWIDTH, FFTSCALING)
```

STAGES (type = DOUBLE): the number of FFT stages

STARTBIN (type = DOUBLE): This specifies the start bin for the output of the FFT and is in the range [-1, 0, 1, FFT Points - 1]. If STARTBIN is positive then consecutive output samples appear sequentially from the given start bin. Typical values are 0 and (FFT Points / 2) - which match the output of the standard Matlab FFT and the Matlab FFTSHIFT function, respectively. STARTBIN = -1 specifies a bit-reversed output.

STAGEWIDTHS (type = DOUBLE): Vector of bit-widths for use through the FFT. Must have (STAGES+1) elements. The first element specifies the bit-width at the input of the FFT; the second element specified the bit-width at the output of the first stage, the (n+1)th element specifies the bit-width at the output of the nth stage.

TWIDDLEWIDTH (type = DOUBLE): Scalar specifying the bit-width of the FFT twiddle factors (the same bit-width is used throughout the model).

FFTSCALING (type = DOUBLE): Scalar specifying the number of MSBs to discard at the output of the FFT. This appears as a factor of $2^{\text{FFTSCALING}}$ at the output of the FFT.

HANDLE (type = UINT32): Handle to a xpFft2Mat object that is then used in all further processing.

4.2 Runtime

The runtime function is used to stream data through the model.

If the inputs are vectors then the FFT is taken of these vectors. The result is then bit-reversed according to the startBin parameter set at initialisation and returned to the workspace. If the inputs are matrices then the FFT is taken down each column of the matrices.

```
[OUTI, OUTQ, OVERFLOW] = xpFft2Mat(HANDLE, INI, INQ)
```

HANDLE (type = UINT32): Handle of valid xpFft2Mat object obtained from INIT function

INI (type = INT64): In-phase input data to the FFT. This is either a vector with the number of elements equal to the number of FFT points or a matrix with the number of rows equal to the number of FFT points.

`INQ` (`type = INT64`): Quad-phase input data to the FFT. This is either a vector with the number of elements equal to the number of FFT points or a matrix with the number of rows equal to the number of FFT points.

`OUTI` (`type = INT64`): In-phase output data. This is either a vector with the number of elements equal to the number of FFT points or a matrix with the number of rows equal to the number of FFT points depending on the format of the input data.

`OUTQ` (`type = INT64`): Quad-phase output data. This is either a vector with the number of elements equal to the number of FFT points or a matrix with the number of rows equal to the number of FFT points depending on the format of the input data.

`OVERFLOW` (`type = DOUBLE`): This is an optional output that flags overflow at various stages of the FFT. There is a single element for each frame of data processed. Each element should be interpreted as an unsigned integer, where the LSB denotes overflow at the input, the second LSB denotes overflow at the output of the first FFT stage, and so on. The model will run faster without this output argument so it should only be used when required.

4.3 Clear

All `xpFft2Mat` objects are destroyed when the dll is cleared from memory (i.e. call "clear `xpFft2Mat`") or specific instances may be destroyed using this function.

All memory is deallocated and the handle is then useless and will generate an error if used.

```
xpFft2Mat(HANDLE, 'clear')
```

`HANDLE` (`type = UINT32`): Handle of valid `xpFft2Mat` object obtained from `INIT` function

4.4 Help

Detailed online help may be displayed in the help browser by typing:

```
xpFft2Mat('help')
```

4.5 Version

A string giving version information for the function may be retrieved by calling:

```
[DESCRIPTION] = xpFft2Mat('version')
```

`DESCRIPTION` (`type = CHAR`): String describing the current function version

5 Examples

5.1 Fixed and Floating Point Comparison

This example shows the effects of fixed point processing relative to floating point processing. The model is compared with the standard Matlab FFT, which uses floating point processing throughout. In order to make the comparison a fair one, the data input to the floating point FFT is quantised.

The example is for an FFT of order 10, but is completely parameterisable, so that if your model does not support FFTs of order 10, then simply changing this number will permit the example to run.

The input sample rate is 100MHz, and a sinusoid is inserted at 25MHz. The bit width at the input of the FFT is set to 12 bits, and is set to grow by one bit per FFT stage.

The input signal is scaled so that the maximum value on either the I or Q channel is one half the full range of the input, resulting in a gain being applied of:

$$\text{inputGain} = 0.5 / \max(\text{signal}) * 2^{\text{inputWidth} - 1}$$

The gain due to the FFT may be calculated from:

$$\text{fftGain} = 2^{\text{outputWidth} - \text{inputWidth} - \text{fftOrder}}$$

So that the fixed point outputs must be divided by both these values to provide the correct scaling.

The example script is:

[xpFft2Mat_example1.m](#) (link only works in the Matlab help browser)

The output of the example is shown in Figure 2. The responses for the two cases are very similar. There are clear spurs at 50MHz and 75MHz; in a practical system these may be eliminated by adding dither noise to the signal before it enters the FFT.

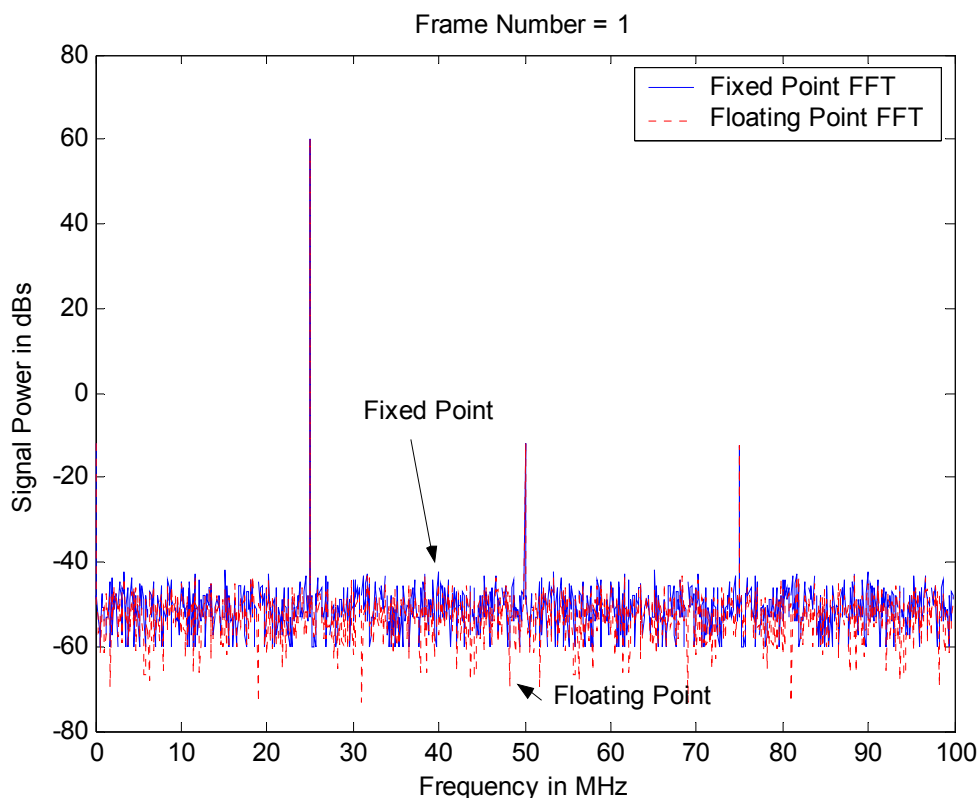


Figure 2: Comparison of Fixed and Floating Point FFTs, normalised by number of FFT Points.

5.2 Bit Width Comparison

This example is similar to example 1, but in this case, two different bit-growths will be investigated. Two models are instantiated, with the first FFT providing one bit per stage of growth and the second providing half a bit per stage growth (i.e. one bit every other stage). The sample rate is again at 100MHz and the input signal is at

37.5MHz. The output of the two FFTs is plotted in Figure 3 along with the response of a floating point implementation. There is a clear difference in the dynamic range of the two fixed point implementations. The spurs are once again clearly visible.

The example script is:

[xpFft2Mat_example2.m](#) (link only works in the Matlab help browser)

Again, the example is for an FFT of order 10, but is completely parameterisable, so that if your model does not support FFTs of order 10, then simply changing this number will permit the example to run.

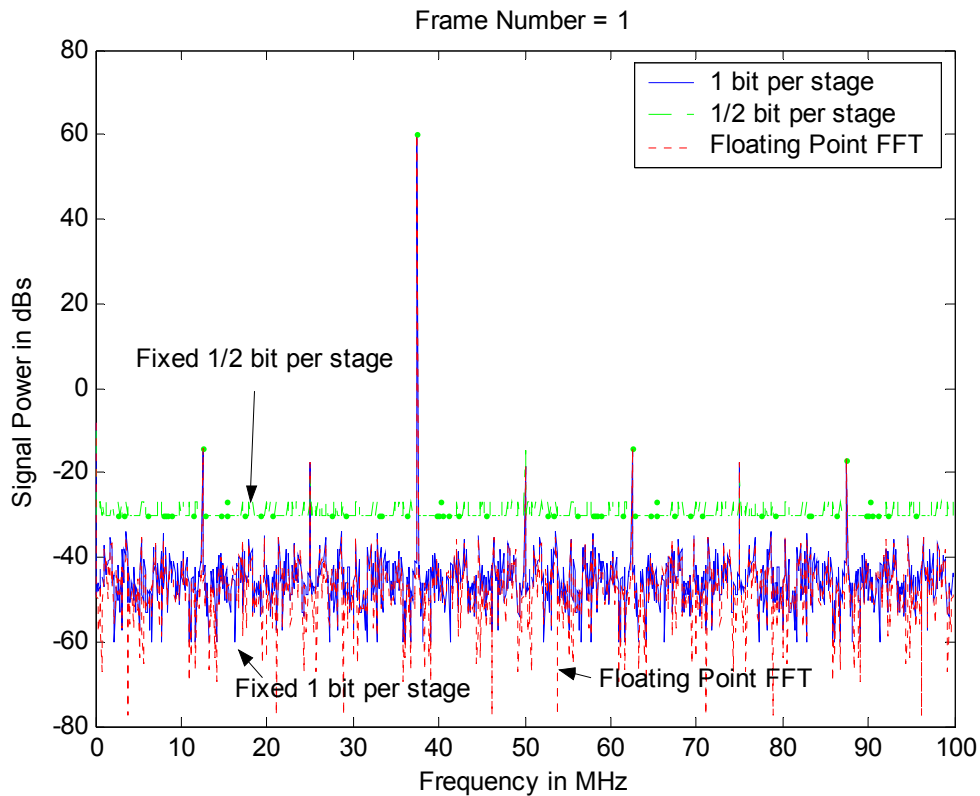


Figure 3: Comparison of fixed point FFTs with different bit growths

5.3 Signals with a Frequency Offset

Finally, an example is provided which has a signal that is not located at the centre of an FFT bin. The parameters are the same as for the previous example, but the sinusoid is now located at 33.33MHz. Signals that are not located at the centre of an FFT bin are not orthogonal to all the other bins, resulting in contributions to a large number of bins, as shown in Figure 4. The responses of the three implementations are almost identical. The result may be masking of low power signals.

For systems where this is a problem, a polyphase DFT may be used to eliminate this spectral spreading. The results of the application of an exemplary Polyphase DFT to the described system are shown in Figure 5 – clearly it provides much better channel selectivity. Polyphase DFT cores, and a supporting model are also available from RFEL.

The script to run this example is:

[xpFft2Mat_example3.m](#) (link only works in the Matlab help browser)

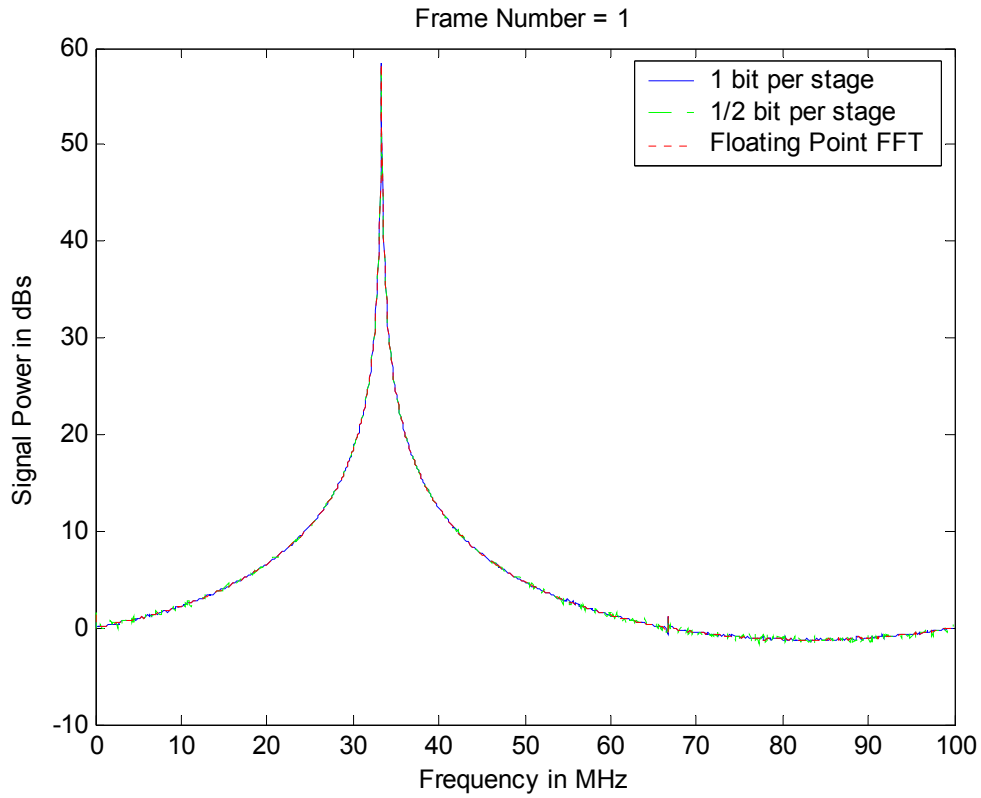


Figure 4: Comparison of FFTs for a signal that is not at the centre of a bin

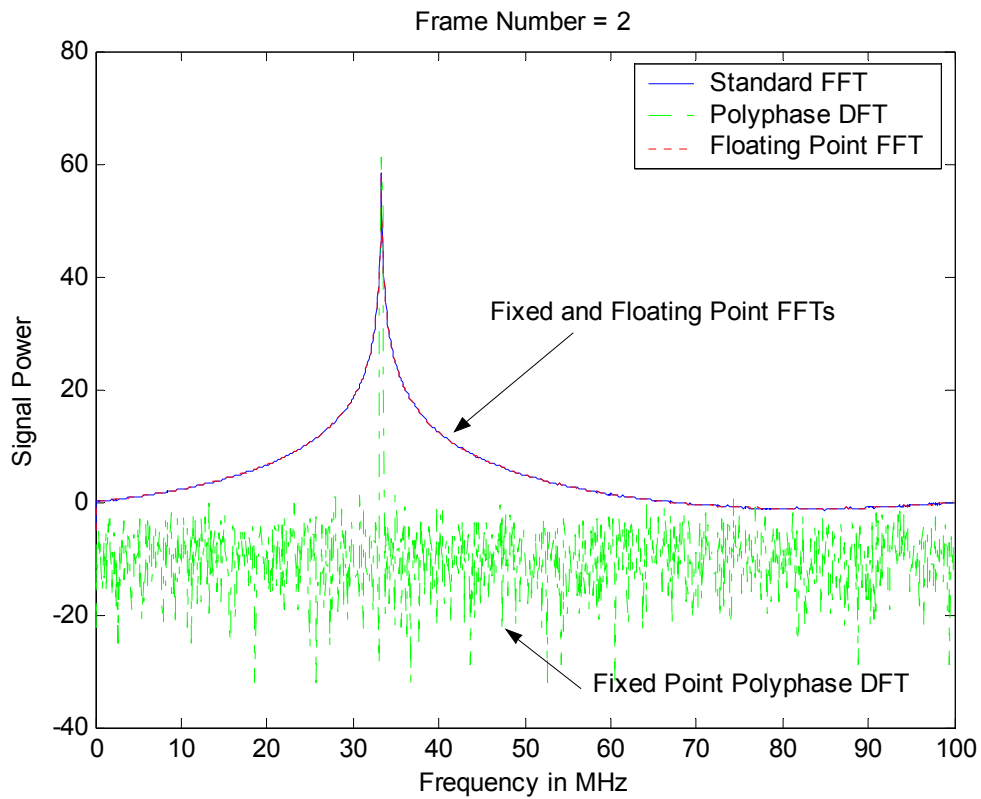


Figure 5: Comparison of FFT with Polyphase DFT