



rf engines

v(1.00) 15-Dec-2003

Product Specification and User Guide
Programmable Window

PART NUMBER
W_1024_XV2_16_18_17_P

© rf engines limited

all rights reserved

<http://www.rfel.com>

Tel: +44(0)1983 550330



TABLE OF CONTENTS

Overview 3
Core version 3
Deliverables 3
General description 3
Window input format 4
Internal precision and scaling 4
Window output formats 4
Core input synchronisation 4
Core output synchronisation 5
Pipeline Latency 5
Programming Coefficients 6
Programmed Coefficients Format 6
Core interface description 7
Place and Route report showing silicon usage 8
Timing 9
Simulated Power Dissipation Report 9
Verification 10
Tools 10
Delivered file directory structure 10
Using the core 11
Using the VHDL test bench 11
Running the test bench for RTL simulation 12
Running the test bench for VITAL simulation 13

OVERVIEW

This document describes the RF Engines Ltd (RFEL) Programmable Window core. The Window requires coefficients to be programmed before use and is primarily intended for use for signal pre-conditioning in high sample rate DSP systems. The core processes continuous complex data, with no gaps. The core is provided in EDIF netlist form as a component.

CORE VERSION

1.0

DELIVERABLES

Supplied Item	Description
Design	EDIF netlist
Constraints File	UCF (User Constraints File)
Instantiation Template	VHDL
Verification	VHDL test bench including ModelSim script and test data files. Compiled RTL VHDL Model. Placement reports.

Table 1: Items provided with each core

GENERAL DESCRIPTION

The core provides a programmable window function of 1024 coefficients. The architecture is based on an area of dual-port memory, provided with a user interface to program the desired coefficient values. During the programming of the coefficients the output will be unpredictable. The coefficients can then be applied to weight complex input data with the programmed characteristics. The final weighted output of the core is symmetrically rounded, complex (I/Q) data samples. Figure 1 below shows the basic architecture of the core.

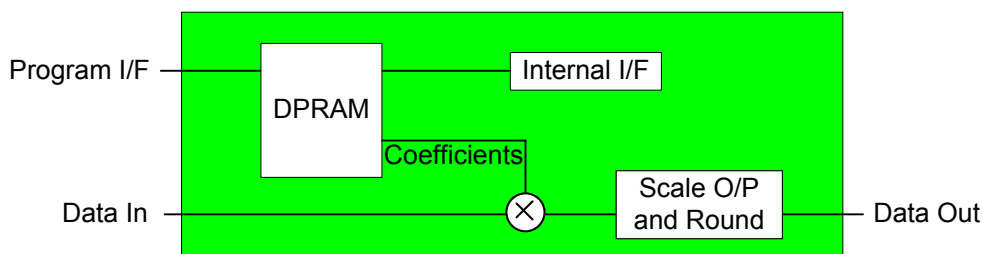


Figure 1: Basic Window

WINDOW INPUT FORMAT

The core is designed to process continuous complex data. A single input '*data_in*' operates at the core clock rate of f_s with alternate in-phase and quadrature-phase samples as shown in Figure 2. The format of the data input is signed 2's-complement.

The signal '*sync_p*' is used to mark the in-phase part of the input sample that corresponds to time zero. This input is required for every 1024 block of interleaved complex samples, 2048 core clock periods. In addition to the '*sync_p*' signal a further signal, '*i_flag_p*' is required to be asserted logic '1' for each in-phase sample, as shown in Figure 2.

Note. The input data rate, f_s , into the core is normally twice the complex sample rate due to the interleaved format; the core must be clocked at this rate of f_s . The output format of the RFEL range of DHBF cores offer direct interface to the programmable window core.

INTERNAL PRECISION AND SCALING

The Programmable Window parameters are given in Table 2.

Parameter	Specification
Input Precision	16 Bits (complex)
Window coefficient bit-width	18 bits
Window coefficient values	Programmable
No of values	1024
Output Rounding	Symmetrical
Output Precision	17 Bits (complex)

Table 2: Programmable Window Specification

WINDOW OUTPUT FORMATS

The core outputs continuous weighted complex data. The format of the data output is signed 2's-complement.

The Programmable Window core has a single output '*data_out*'. This operates at the core clock rate of f_s with alternate in-phase and quadrature-phase samples as shown in Figure 2.

CORE INPUT SYNCHRONISATION

Figure 2 assumes that programming of the coefficients has been previously performed, and shows the data output timing and control signals, relative to the input.

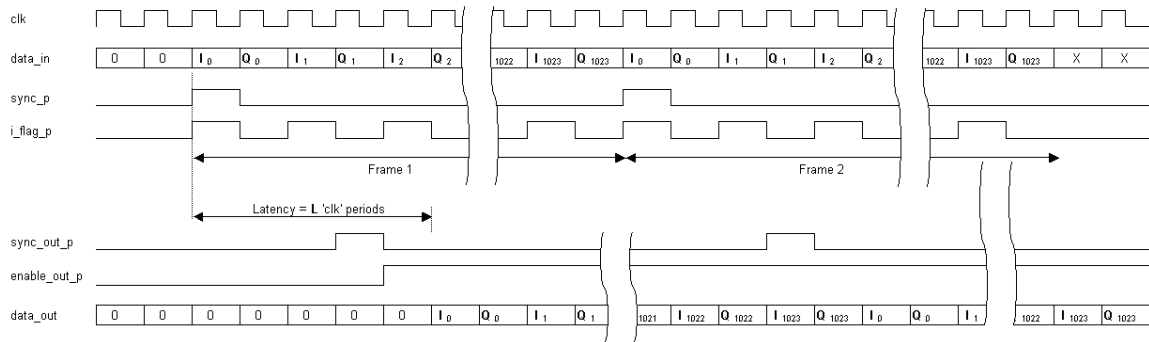


Figure 2: Window I/O Timing Diagram Example

CORE OUTPUT SYNCHRONISATION

Weighted complex samples are output after the core latency period. The first sample is specifically marked with the 'sync_out_p' signal. This precedes the first interleaved in-phase complex sample of 'data_out' by two core clock periods. The active high 'enable_out_p' signal is also asserted for duration equal to the block size; one clock period prior to the first interleaved in-phase complex sample as shown in the timing diagram example of Figure 2.

This timing and output format allows direct interface with the specific RFEL Vectris HiSpeed range of FFT cores.

PIPELINE LATENCY

Pipeline latency, defined as the time from when the first complex sample is clocked into the Window to the time when the first weighted complex output sample is clocked out from the Window, is shown in the timing diagram example of Figure 2.

The pipeline latency 'L' = 5 core clocks.

The Window core is a continuous pipeline, so after the initial delay due to the pipeline filling up as described previously, valid data is continuously available at the output.

PROGRAMMING COEFFICIENTS

The core is designed as an area of dual-port memory, provided with a basic user interface to program the window coefficient values. The data input, 'coeffs_data' is assigned a coefficient value and on the rising edge of the programming clock input, 'wr_clk' the content of the dual-port memory at the location specified by the external programming 'address' input is replaced with the 'coeffs_data' value. The 'wr' signal acts as an enable to qualify the data and address values and must be logic '1' for programming to occur.

The window dual-port memory is implemented using Virtex-II Block RAM and the programming interface must satisfy the relevant device switching characteristics.

PROGRAMMED COEFFICIENTS FORMAT

The format of the coefficient data is signed 2's-complement.

The coefficient values are required to be stored in the dual-port memory in a linear order from address 0 to N-1, where N is the block size of 1024. This is illustrated in Figure 3.

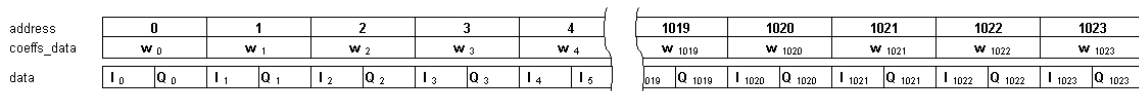


Figure 3: Coefficient storage order

After the coefficients have been programmed, during normal operation the internal read address counts linearly at half the input data rate, this ensures that the correct weighting is applied to each complex in-phase and quadrature-phase input sample. This is also illustrated using Figure 3.

CORE INTERFACE DESCRIPTION

All core inputs should be synchronised to the '*clk*' signal. All core outputs are synchronised by the '*clk*' signal. Programming inputs should be synchronised to the '*wr_clk*' signal.

Signal	Direction	Type	Width	Function
clk	IN	Std logic	1 bit	The core clock rate is equal to f_s . Where f_s is the input rate.
rst_p	IN	Std logic	1 bit	An active-high pulse, of duration greater than 4 core clock periods.
sync_p	IN	Std logic	1 bit	Active-high pulse marking the first sample of a new input block. Coincident with the first sample of complex input data – must also be coincident with <i>i_flag_p</i> .
i_flag_p	IN	Std logic	1 bit	Active-high signal asserted on alternate clock periods. Indicates each in-phase data sample into the core.
data_in	IN	Std logic vector	(15 downto 0)	2's complement interleaved complex data.
sync_out_p	OUT	Std logic	1 bit	Active-high pulse marking the first sample of a new output block. Precedes the first sample of complex output data by two clock periods.
enable_out_p	OUT	Std logic	1 bit	Active-high signal asserted for duration equal to the output block length. Asserted one clock period before the first sample of complex input data.
data_out	OUT	Std logic vector	(16 downto 0)	2's complement interleaved complex data samples.
Programming I/F				
wr_clk	IN	Std logic	1 bit	The programming clock, used for programming the window coefficient values.
wr	IN	Std logic	1 bit	Active-high write enable signal used to validate the programming signals.
address	IN	Std logic vector	(9 downto 0)	Address bus used for programming the window coefficient values.
coeff_data	IN	Std logic vector	(17 downto 0)	Coefficient value data bus. Used during programming of the window coefficient values.

Table 3: Window Interface Specification

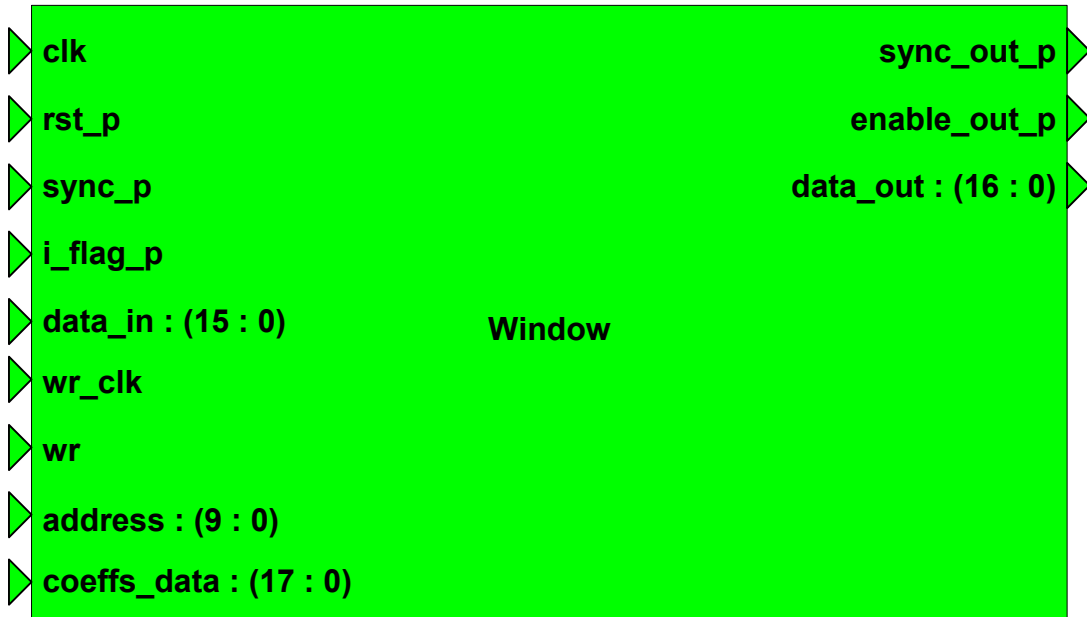


Figure 4: Window Symbol

PLACE AND ROUTE REPORT SHOWING SILICON USAGE

Release 6.1.02i Map G.25a
Xilinx Mapping Report File for Design 'window_core'

Design Information

```
-----
Command Line   : map -u -p xc2v3000-5fg676 window_core.ngd -o
window_core_map.ncd window_core.pcf
Target Device  : x2v3000
Target Package : fg676
Target Speed   : -5
Stepping Level : 1
Mapper Version : virtex2 -- $Revision: 1.16 $
Mapped Date    : Tue Dec 16 14:32:55 2003
```

Design Summary

```
-----
Number of errors:      0
Number of warnings:   72
Logic Utilization:
  Number of Slice Flip Flops:      110 out of 28,672    1%
  Number of 4 input LUTs:          26 out of 28,672    1%
Logic Distribution:
  Number of occupied Slices:        79 out of 14,336    1%
  Number of Slices containing only related logic: 79 out of 79 100%
  Number of Slices containing unrelated logic:    0 out of 79  0%
  *See NOTES below for an explanation of the effects of unrelated logic
Total Number 4 input LUTs:          45 out of 28,672    1%
  Number used as logic:              26
  Number used as a route-thru:       18
  Number used as Shift registers:     1
```



Number of Block RAMs:	1 out of	96	1%
Number of MULT18X18s:	1 out of	96	1%

Total equivalent gate count for design: 70,795
 Peak Memory Usage: 113 MB

NOTES:

Related logic is defined as being logic that shares connectivity - e.g. two LUTs are "related" if they share common inputs. When assembling slices, Map gives priority to combine logic that is related. Doing so results in the best timing performance.

Unrelated logic shares no connectivity. Map will only begin packing unrelated logic into a slice once 99% of the slices are occupied through related logic packing.

Note that once logic distribution reaches the 99% level through related logic packing, this does not mean the device is completely utilized. Unrelated logic packing will then begin, continuing until all usable LUTs and FFs are occupied. Depending on your timing budget, increased levels of unrelated logic packing may adversely affect the overall timing performance of your design.

TIMING

Timing summary:

Timing errors: 0 Score: 0

Constraints cover 468 paths, 0 nets, and 204 connections

Design statistics:

Minimum period: 6.601ns (Maximum frequency: 151.492MHz)

Analysis completed Tue Dec 16 14:33:58 2003

SIMULATED POWER DISSIPATION REPORT

 Release 6.1.02i - XPower SoftwareVersion:G.25a
 Copyright (c) 1995-2003 Xilinx, Inc. All rights reserved.
 Design: window_wrapper
 Preferences: working\pandr>window_wrapper.pcf
 VCD File: working\vital_sim>window_wrapper.vcd
 Part: 2v3000fg676-5
 Data version: ADVANCED,v1.0,07-31-02

Power summary:	I (mA)	P (mW)

Total estimated power consumption:		646

Vccint 1.50V:	207	310



Vccaux 3.30V:	100	330
Vcco33 3.30V:	2	7

Clocks:	0	0
Inputs:	3	4
Logic:	4	5
Outputs:		
Vcco33	0	0
Signals:	0	0

Quiescent Vccint 1.50V:	200	300
Quiescent Vccaux 3.30V:	100	330
Quiescent Vcco33 3.30V:	2	7

Note: The power simulation shown above was run using a clock period of 9.5 ns (105MHz) and 1 μ s of simulation; approximately 1052 random input samples.

VERIFICATION

Verification of the core is achieved in several distinct phases in the design cycle.

Initially, a bit-true Matlab model of the core is used to provide reference output data, using a specific set of input stimuli. These models exactly match the behaviour of the hardware architecture and their outputs are bit-true representations of the actual core.

The input stimuli are used to functionally (pre-synthesis) test the VHDL code using the ModelSim simulator. The results of these simulations are automatically compared with the reference data generated by the bit-true Matlab model and pass/fail indications are reported.

After the VHDL has been synthesised and place-and-route of the core is complete, timing simulations, based on the chip vendor's simulation netlist and gate delay files for the core are performed. These also take the form of the functional tests, but being based on the compiled netlist and timing files from the FPGA vendor, they verify the expected timings of the delivered core.

The bit-true Matlab models are not included.

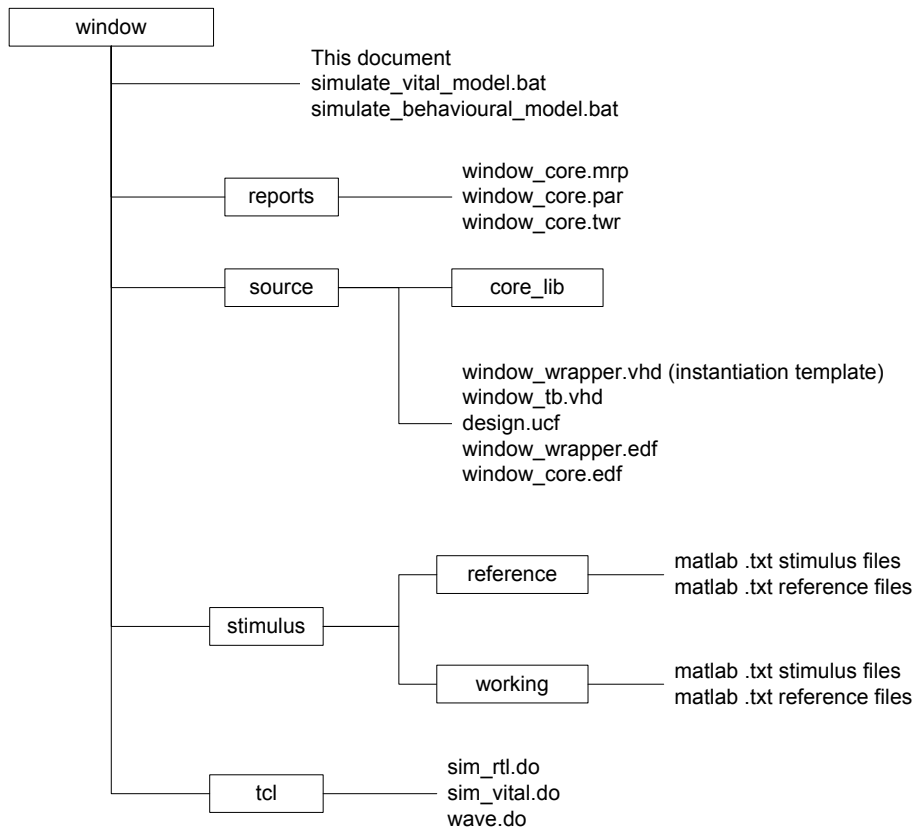
TOOLS

The following tools and versions were used to generate this delivery.

Modelsim PE	Version: 5.4e
Leonardo Spectrum Level 2 or 3	Version: LS2003b_35
Xilinx ISE	Release Version: 6.1.02i
Matlab	Version: 6.5

DELIVERED FILE DIRECTORY STRUCTURE

The delivered file structure is shown in Figure 5.

**Figure 5: Delivered file structure**

USING THE CORE

The following steps should be followed to ensure the EDIF netlist is included in the final design:

1. Include the instance(s) of the core within the target design (using '*window/source/window_wrapper.vhd*' as a reference).
2. Place the core EDIF file '*window/source/window_core.edf*' in the same directory as the final design EDIF, or add the macro search path in the process properties dialog box (right mouse click over implement design > properties) in XilinxISE.
3. Use the parameters contained in the constraints file '*window/source/design.ucf*' as a reference.
4. Continue through normal place and route using the XilinxISE tools and the core will be automatically integrated into the design.

USING THE VHDL TEST BENCH

The VHDL test bench provides control and data stimuli to either the pre-compiled behavioural VHDL model, or a VITAL VHDL model that can be created by the XilinxISE tools. The Matlab bit-true model generated data stimuli and reference result files, are held in the

'*window/stimulus/working*' directory. The test bench produces a latched pass / fail indication by automatically comparing the core output against the reference file values during simulation.

The name of the VHDL test bench is: - *window_tb.vhd*

N.B. The window_tb.vhd and window_wrapper.vhd files are NOT to be edited by the user since all parameters are passed down via the simulation script.

RUNNING THE TEST BENCH FOR RTL SIMULATION

Running the DOS BAT file '*window/simulate_behavioural_model.bat*' will perform a behavioural simulation. The script performs the following:

1. If the directory '*window/working/behavioural_sim*' does not already exist (as will be the case the first time the script is run), then it is created.
2. The pre-compiled behavioural VHDL model '*window/source/core_lib*' is copied into '*window/working/behavioural_sim*'.
3. Modelsim is launched in GUI mode, and the TCL script '*window/tcl/sim_rtl.do*' is run. Modelsim's transcript path is set to '*window/working/behavioural_sim/transcript.txt*' so that it may be viewed after simulation if required.
4. When Modelsim has launched, the TCL script '*window/tcl/sim_rtl.do*' performs the following:
 - a. Modelsim's working directory is set to '*window/working/behavioural_sim*'.
 - b. A working library '*window/working/behavioural_sim/lib*' is created and mapped to as '*work*'.
 - c. The copied version of the pre-compiled behavioural VHDL model library '*core_lib*' is mapped to as '*window_core_library*', and updated (vcom -refresh) to the current ModelSim library format from it's delivered ModelSim 5.4e format.
 - d. The VHDL core wrapper '*window/source/window_wrapper.vhd*' and the VHDL test bench '*window/source/window_tb.vhd*' are compiled, and the test bench is loaded.
 - e. When the test bench is loaded, various generic values are assigned including the clock period of the core, input and output bit-widths and stimulus / reference result file paths (set to '*window/stimulus/working*').
 - f. The TCL script '*window/source/wave.do*' is run to open a wave window with the relevant test bench signals required to check the simulation when completed.
 - g. The simulation is run until all of the test bench processes have reached their '*wait*' states.
5. The waveforms in the waveform window will illustrate the core signals and reference output results as multiplexed I and Q. The latched '*sim_error*' signal should be a '0' to indicate no errors.
6. A result file is also produced in the '*window/stimulus/working*' directory, the samples are interleaved I and Q.

RUNNING THE TEST BENCH FOR VITAL SIMULATION

Before VITAL simulation can take place, the user is required to edit and update the script '*window/tcl/sim_vital.do*' to reflect the location of the compiled Xilinx '*simprim*' library. The following text within the script should be edited appropriately:

```
#####  
#### MAP TO YOUR COMPILED SIMPRIM LIBRARY HERE....  
vmap simprim E:/Xilinx/sim_lib/simprim  
#####
```

When this is done, VITAL simulation will be performed by running the DOS BAT file '*window/simulate_vital_model.bat*'. This script performs the following:

1. If the directory '*window/working/pandr*' does not already exist (as will be the case the first time the script is run), then the script performs the following:
 - a. The directory '*window/working/pandr*' is created. This is where all of the wrapped core place and route files will be written.
 - b. '*window/source/window_wrapper.edf*' is implemented using the XilinxISE tools. This netlist is the core wrapper design, synthesised with I/O pads, I/O registers and a clock buffer.

The core '*window/source/window_core.edf*' is automatically included within the wrapper because '*window/source*' has been specified as a macro search path within the script.
 - c. After implementation, the script checks to see which version of XilinxISE is currently installed. If version 5 is present, then the '*ngd2vhdl*' function is used to create the VHDL VITAL simulation model, otherwise it is assumed that the user has version 6 or later, and the newer '*netgen*' function is used instead.
2. If the directory '*window/working/vital_sim*' does not already exist (as will be the case the first time the script is run), then it is created.
3. ModelSim is launched in GUI mode, and the TCL script '*window/tcl/sim_vital.do*' is run. ModelSim's transcript path is set to '*window/working/vital_sim/transcript.txt*' so that it may be viewed after simulation if required.
4. '*simulate_vital_model.bat*' now waits until the file '*window/working/vital_sim/done*' is written by ModelSim to indicate that simulation is complete..
5. When ModelSim has launched, the TCL script '*window/tcl/sim_vital.do*' performs the following:
 - a. Modelsim's working directory is set to '*window/working/vital_sim*'.
 - b. A working library '*window/working/vital_sim/lib*' is created and mapped to as '*work*'.
 - c. The compiled '*simprim*' library is mapped as specified by the user.
 - d. The '*window/working/pandr/window_wrapper_ba.vhd*' VITAL model and the VHDL test bench '*window/source/window_tb.vhd*' are compiled, and the test bench is loaded.
 - e. The SDF file '*window/working/pandr/window_wrapper_ba.sdf*' is applied to the unit under test '*uut*' when the test bench is loaded for simulation.
 - f. When the test bench is loaded, various generic values are assigned including the clock period of the core, input and output bit-widths and stimulus / reference result file paths, (set to '*window/stimulus/working*').
 - g. The TCL script '*window/source/wave.do*' is run to open a wave window with the relevant test bench signals required to check the simulation when completed.



- h. The simulation is run for 50 μ s to a point where data is being processed, and then ModelSim is instructed to create a VCD file containing signal state information of all signals within the unit under test '*uut*'. The simulation is run for a further 10 μ s to allow >1000 samples of random data to pass through the core and be logged within the VCD file. When the simulation is complete, the file '*window/working/vital_sim/done*' is created to tell '*window/simulate_vital_model.bat*' to proceed with power analysis.
7. When '*window /working/vital_sim/done*' has been written, the XilinxISE '*xpwr*' tool is launched to provide power analysis figures for the simulated VITAL design. The results of this power analysis are written to '*window/working/vital_sim/window_wrapper.pwr*', and are displayed on the Command Prompt window.
8. A results file is also produced in the '*window/stimulus/working*' directory, the samples are interleaved I and Q. A full simulation will not have been performed as this simulation is targeted at power analysis only.

ADC	Analogue to Digital Converter
ASIC	Application Specific Integrated Circuit
CLB	Configurable Logic Block
DFT	Discrete Fourier Transform
DHBF	Distributed Half-Band Filter
EDIF	Electronic Data Interchange Format
FFT	Fast Fourier Transform
FPGA	Field Programmable Gate Array
HDL	Hardware Description Language
I/F	Interface
I/O	Input / Output
LSB	Least Significant Bit
MSB	Most Significant Bit
MS/s	Million Samples Per Second
PFT	Pipelined Frequency Transform
PFFT	Pipelined Fast Fourier Transform
RFEL	RF Engines Limited
RTL	Register Transfer Level
RPM	Relationally Placed Macro
UCF	User Constraints File
VHDL	Very High Speed IC Hardware Description Language
VITAL	VHDL Initiative Toward ASIC Libraries
.bat	Batch File
.do	Modelsim script file
.vhd	A VHDL File
.edf	An EDIF File
.ucf	User Constraints File
.mrp	Map Report File
.par	Place and Route Report File
.sdf	Standard Delay Format
.twr	Place and Route Timing Report File

Table 4: Glossary