



rf engines

v(1.00) 14-Jan-2004

Product Specification and User Guide
Power Block Accumulator

PART NUMBER
BACC_1024_XV2_28_256_64_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
Power Block Accumulator Input Format 4
Internal precision and scaling 4
Power Block Accumulator output formats 5
Core input synchronisation 5
Core output synchronisation 5
Pipeline Latency 5
Programming Number of Frames to Accumulate 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) Power Block Accumulator core. The core requires the number of blocks of data for accumulation to be programmed before use and is primarily intended for use after an FFT in high sample rate DSP systems. The core processes complex data, with or without gaps between each block. A block is defined as 1024 samples of continuous complex data, which is normally supplied by a 1024-point Vectis 'HiSpeed' FFT. The core is provided in EDIF netlist form as a component.

CORE VERSION1.0

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

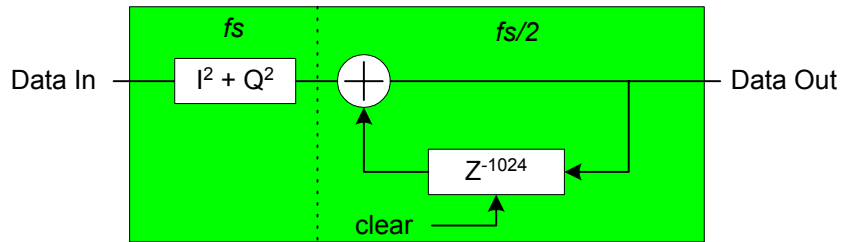
GENERAL DESCRIPTION

The core provides accumulation of power for blocks of 1024 continuous complex samples over a programmable range. Accumulation is programmable over the range of 1 to 256 blocks, via an external interface. After programming, the output will be incorrect until the programmed number of blocks of data have been input to the core.

Power is calculated as I^2+Q^2 and uses the full precision of the input data. Programming an accumulation value of 0 will output power un-averaged

Programming an accumulation value, 'A', between 1 and 255 will output block-accumulated power. The accumulated power that has been averaged over [A – 1] blocks.

The accumulated output of the core is the full precision of the power. Figure 1 shows the basic architecture of the core.


Figure 1: Basic Power Block Accumulator

POWER BLOCK ACCUMULATOR 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_in*' is used to mark the in-phase part of the input sample that corresponds to time zero. This input is required for every new block of interleaved complex samples. In addition to the '*sync_in*' signal a further signal, '*enable_in*' is required to be asserted logic '1' for the duration of each block of input samples, as shown in Figure 2.

Note. The input data rate, f_s , into the core is 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 Vectis 'HiSpeed' FFT cores offer direct interface to the Power Block Accumulator core.

If each block of input data does not arrive as continuous data then gaps between each block must be multiples of 2 core clock periods, f_s . This should be the case due to interleaved format of the complex input data.

INTERNAL PRECISION AND SCALING

The Power Block Accumulator parameters are given in Table 2.

Parameter	Specification
Input Precision	28 Bits (complex)
Power Precision	57 bits (internal)
Block Length	1024
Accumulated Power Precision	64 bits maximum *
Output Precision	64 Bits

Table 2: Power Block Accumulator Specification

*If accumulation is set to 256 blocks, and both in-phase and quadrature-phase samples are "1000...0000" for all 256 samples, then an overflow condition will occur.

This situation cannot occur when the core is fed from a Vectis 'HiSpeed' FFT, and is extremely unlikely to occur in normal use.

POWER BLOCK ACCUMULATOR OUTPUT FORMATS

The Power Block Accumulator core has a single output *'data_out'*. This operates at half the core clock rate of $f_s/2$ with accompanying *'clk_en_out'* signal as shown in Figure 2. The format of the data output is 64-bit signed 2's complement.

CORE INPUT SYNCHRONISATION

Figure 2 assumes that programming of the accumulation constant has been previously performed, and shows the data output timing and control signals, relative to the input for two blocks.

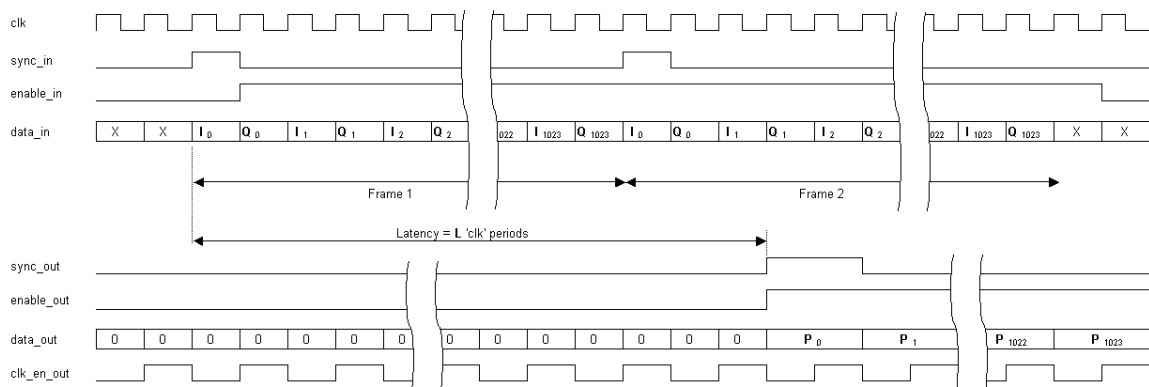


Figure 2: Core I/O Timing Diagram Example

CORE OUTPUT SYNCHRONISATION

Power samples are output after the core latency period. The first sample is specifically marked with the *'sync_out'* signal. This is coincident with the first sample of *'data_out'*. The active high *'enable_out'* signal is also asserted for duration equal to the block size coincident with the first sample as shown in the timing diagram example of Figure 2.

PIPELINE LATENCY

Pipeline latency, defined as the time from when the first complex sample is clocked into the Power Block Accumulator core to the time when the first accumulated power sample is clocked out from the Power Block Accumulator core, is shown in the timing diagram example of Figure 2.

The pipeline latency $'L' = 10 + (2048 * \text{number of blocks to accumulate})$ core clocks.

This assumes input data has no gaps between each block.



PROGRAMMING NUMBER OF FRAMES TO ACCUMULATE

The core is designed with a basic user interface to program the number of blocks or frames over which to accumulate power.

The input, '*acc_frames*' is presented with the desired number of frames to accumulate and on the rising edge of the programming clock input, '*wr_clk*', the content of the accumulation control register is replaced with the '*acc_frames*' value. The '*wr*' signal acts as an enable to qualify the data and must be logic '1' for programming to occur.

The programming interface must satisfy the relevant device switching characteristics of the target technology.

The format of the accumulation constant is 8-bit unsigned 2's complement in the range 0 to 255.

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_in	IN	Std logic	1 bit	Active-high pulse marking the first sample of a new input block. Coincident with the first in-phase sample of complex input data.
enable_in	IN	Std logic	1 bit	Active-high signal asserted for duration equal to the input block length. Asserted one clock period after the first in-phase sample of complex input data.
data_in	IN	Std logic vector	(27 downto 0)	2's complement interleaved complex data.
sync_out	OUT	Std logic	1 bit	Active-high pulse 2 clock periods in duration marking the first sample of a new output block. Coincident with the first sample of output data.
enable_out	OUT	Std logic	1 bit	Active-high signal asserted coincident with the first sample of output data for duration equal to the output block length.
data_out	OUT	Std logic vector	(63 downto 0)	Signed 2's complement power data samples at $f_s/2$ rate.
clk_en_out	OUT	Std logic vector	1 bit	Active-high signal asserted on alternate clock periods.
Programming I/F				
wr_clk	IN	Std logic	1 bit	The programming clock, used for programming the accumulation constant.
wr	IN	Std logic	1 bit	Active-high write enable signal used to validate the programming signals.
acc_frames	IN	Std logic vector	(7 downto 0)	Accumulation constant value data bus. Used during programming to set the number of blocks of power data to accumulate.

Table 3: Power Block Accumulator Interface Specification

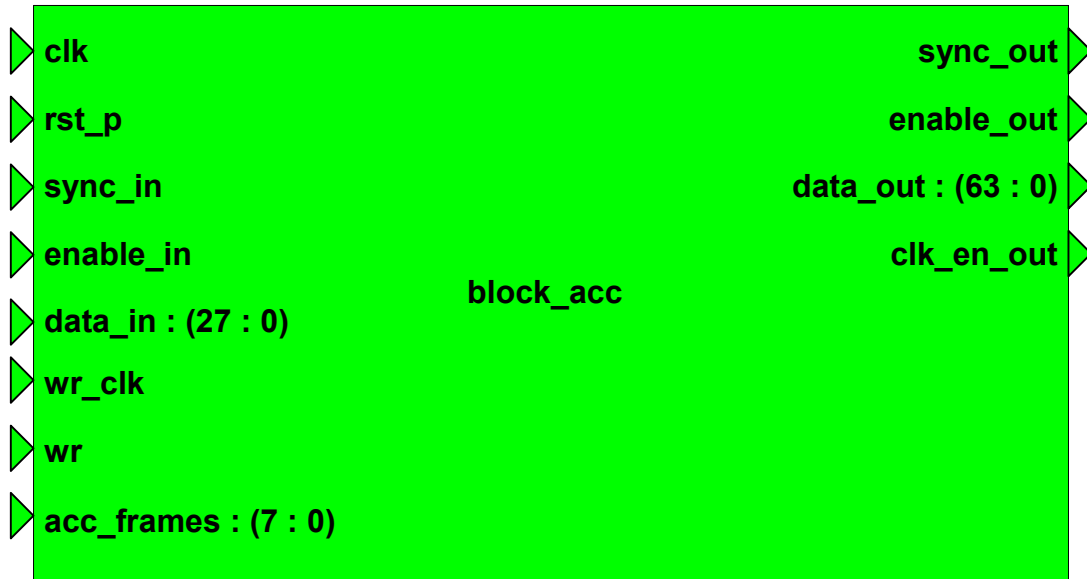


Figure 3: Power Block Accumulator Symbol

PLACE AND ROUTE REPORT SHOWING SILICON USAGE

Release 5.2.03i - Map F.31
 Xilinx Mapping Report File for Design 'sp_blk_acc_core'

Design Information

```
-----
Command Line   : map -u -p xc2v3000-5fg676 sp_blk_acc_core.ngd -o
sp_blk_acc_core_map.ncd sp_blk_acc_core.pcf
Target Device  : x2v3000
Target Package : fg676
Target Speed   : -5
Stepping Level : 1
Mapper Version : virtex2 -- $Revision: 1.4 $
Mapped Date    : Thu Jan 15 14:41:27 2004
```

Design Summary

```
-----
Number of errors:      0
Number of warnings:   134
Logic Utilization:
  Number of Slice Flip Flops:      712 out of 28,672    2%
  Number of 4 input LUTs:         391 out of 28,672    1%
Logic Distribution:
  Number of occupied Slices:              389 out of 14,336    2%
  Number of Slices containing only related logic:  389 out of   389  100%
  Number of Slices containing unrelated logic:    0 out of   389    0%
  *See NOTES below for an explanation of the effects of unrelated logic
Total Number 4 input LUTs:              401 out of 28,672    1%
  Number used as logic:                  391
  Number used as a route-thru:           9
  Number used as Shift registers:         1

  Number of Block RAMs:                  4 out of   96    4%
  Number of MULT18X18s:                  4 out of   96    4%
```



Total equivalent gate count for design: 287,795
Peak Memory Usage: 111 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 16124 paths, 0 nets, and 2194 connections (93.7% coverage)

Design statistics:

Minimum period: 6.784ns (Maximum frequency: 147.406MHz)

Analysis completed Thu Jan 15 14:42:13 2004

SIMULATED POWER DISSIPATION REPORT

Release 5.2.03i - XPower SoftwareVersion:F.31
Copyright (c) 1995-2002 Xilinx, Inc. All rights reserved.

Design: sp_blk_acc_wrapper
Preferences: working\pandr\sp_blk_acc_wrapper.pcf
VCD File: working\vital_sim\sp_blk_acc_wrapper.vcd
Part: 2v3000fg676-5
Data version: ADVANCED 1.114 2002-12-13

Table with 3 columns: Power summary, I (mA), P (mW). Rows include Total estimated power consumption (389), Vccint 1.5V (200, 300), and Vccaux 3.3V (25, 83).



Vcco 3.3V:	2	7

Clocks:	0	0
Nets:	0	0
Logic:	0	0
Inputs:	0	0
Outputs:		0
Quiescent 1.5V:	200	300
Quiescent 3.3V:	25	83
Quiescent 3.3V:	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 using the supplied VHDL test bench, and compiled behavioural and VITAL models.

The test bench applies random stimulus to the core, and automatically checks that the accumulated results are correct. An error flag is latched active if any converted result is not correct.

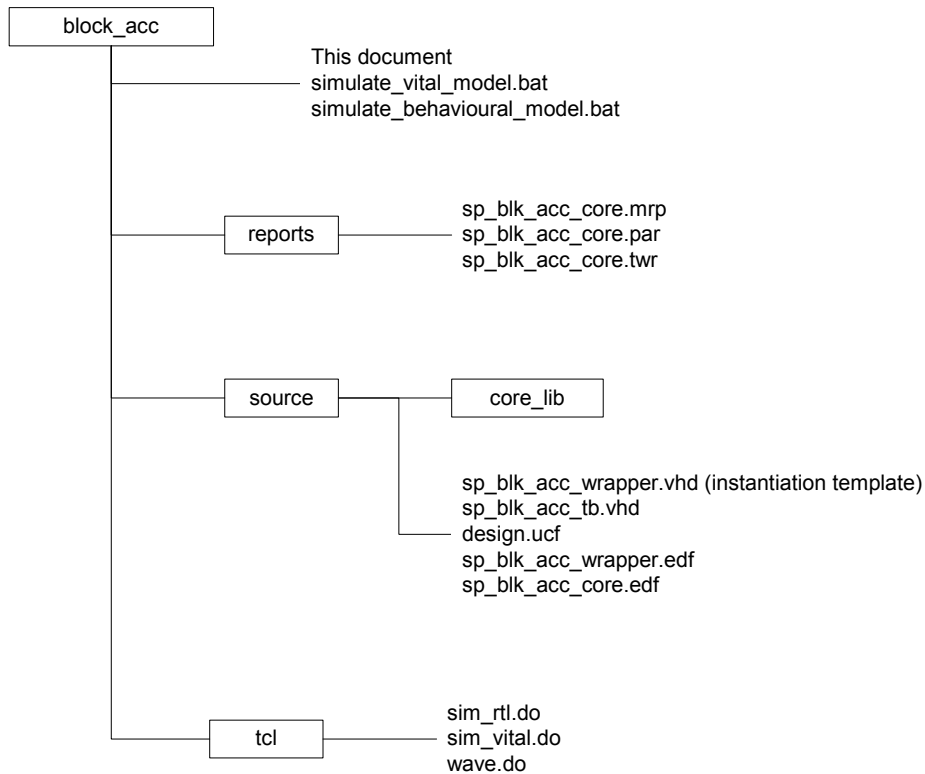
TOOLS

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: 5.2.03i
Matlab	Version: 6.5

DELIVERED FILE DIRECTORY STRUCTURE

The delivered file structure is shown in Figure 4.

**Figure 4: 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 '*block_acc/source/sp_blk_acc_wrapper.vhd*' as a reference).
2. Place the core EDIF file '*block_acc/source/sp_blk_acc_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 '*block_acc/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. Random stimuli and expected results are generated within the test bench. The test bench produces a latched pass / fail indication by automatically comparing the core outputs against the expected values during simulation.

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



N.B. The `sp_blk_acc_tb.vhd` and `sp_blk_acc_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

Before simulation can take place, the user can edit and update the script '`block_acc/tcl/sim_rtl.do`' to change the number of blocks (frames) to accumulate and the number of accumulations. The following text within the script should be edited appropriately:

```
#####  
#-- EDIT THIS TEXT TO SET DATAFILES FOR SIMULATION  
#-- Maximum 256 Minimum 1  
#####  
  
set frames 3;  
#number of frames of data per accumulation  
set frame_gap 2;  
#gap between each frame  
set accs 5;  
#number of accumulations of frames
```

The example shown will test the core with 15 blocks (frames) of input data, with output accumulation every 3 blocks (frames). A gap between each input block will be 2 interleaved complex samples.

When this is done, running the DOS BAT file '`block_acc/simulate_behavioural_model.bat`' will perform a behavioural simulation. The script performs the following:

1. If the directory '`block_acc/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 '`block_acc/source/core_lib`' is copied into '`block_acc/working/behavioural_sim`'.
3. Modelsim is launched in GUI mode, and the TCL script '`block_acc/tcl/sim_rtl.do`' is run. Modelsim's transcript path is set to '`block_acc/working/behavioural_sim/transcript.txt`' so that it may be viewed after simulation if required.
4. When Modelsim has launched, the TCL script '`block_acc/tcl/sim_rtl.do`' performs the following:
 - a. Modelsim's working directory is set to '`block_acc/working/behavioural_sim`'.
 - b. A working library '`block_acc/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 '`sp_blk_acc_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 '`block_acc/source/sp_blk_acc_wrapper.vhd`' and the VHDL test bench '`block_acc/source/sp_blk_acc_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, the user defined values to test accumulation constant and number of sets of input data.
 - f. The TCL script '`block_acc/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. The latched 'sim_error' signal should be a '0' to indicate no errors.

RUNNING THE TEST BENCH FOR VITAL SIMULATION

Before VITAL simulation can take place, the user is required to edit and update the script '*block_acc/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  
#####
```

Also the number of blocks (frames) to accumulate, the number of accumulations and the gaps between each block need to be set. The following text within the script should be edited appropriately:

```
#####  
#-- EDIT THIS TEXT TO SET DATAFILES FOR SIMULATION  
#-- Maximum 256 Minimum 1  
#####  
  
set frames 3;  
#number of frames of data per accumulation  
set frame_gap 1;  
#gap between each frame  
set accs 2;  
#number of accumulations of frames
```

The example shown will test the core with 6 blocks (frames) of input data, with output accumulation every 3 blocks (frames). A gap between each input block will be 1 interleaved complex sample.

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

1. If the directory '*block_acc/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 '*block_acc/working/pandr*' is created. This is where all of the wrapped core place and route files will be written.
 - b. '*block_acc/source/sp_blk_acc_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 '*block_acc/source/sp_blk_acc_core.edf*' is automatically included within the wrapper because '*block_acc/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 '*block_acc/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 '*block_acc/tcl/sim_vital.do*' is run. ModelSim's transcript path is set to '*block_acc/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 '*block_acc/working/vital_sim/done*' is written by ModelSim to indicate that simulation is complete..
5. When ModelSim has launched, the TCL script '*block_acc/tcl/sim_vital.do*' performs the following:
 - a. Modelsim's working directory is set to '*block_acc/working/vital_sim*'.
 - b. A working library '*block_acc/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 '*block_acc/working/pandr/sp_blk_acc_wrapper_ba.vhd*' VITAL model and the VHDL test bench '*block_acc/source/sp_blk_acc_tb.vhd*' are compiled, and the test bench is loaded.
 - e. The SDF file '*block_acc/working/pandr/sp_blk_acc_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, the user defined values to test accumulation constant and number of sets of input data.
 - g. The TCL script '*block_acc/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 '*block_acc/working/vital_sim/done*' is created to tell '*block_acc/simulate_vital_model.bat*' to proceed with power analysis.
6. When '*block_acc/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 '*block_acc/working/vital_sim/sp_blk_acc_wrapper.pwr*', and are displayed on the Command Prompt window.
7. A full simulation will not have been performed as this simulation is targeted at power analysis only.

CLB	Configurable Logic Block
DFT	Discrete Fourier Transform
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
PFFT	Pipelined Fast Fourier Transform
RFEL	RF Engines Limited
RTL	Register Transfer Level
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