CASPER Library

Reference Manual

Last Updated May 24, 2008

Contents

Chapter 1

Signal Processing Blocks

1.1 Adder Tree *(adder_tree)*

Block Author: Aaron Parsons
Document Author: Aaron Parsons

Summary

Sums all inputs using a tree of adds and delays.

Mask Parameters

Parameter	Variable	Description
No. of inputs.	*n_inputs*	The number of inputs to be summed.
Add Latency	*latency*	The latency of each stage through the adder tree.

Ports

Port	Dir.	Data Type	Description
sync	in	Boolean	Indicates the next clock cycle containing valid data
din	in	Inherited	A number to be summed.

Description

Sums all inputs using a tree of adds and delays. Total latency is $ceil(log_2(n\_inputs)) * latency$

1.2 Barrel Switcher *(barrel_switcher)*

Block Author: Aaron Parsons
Document Author: Aaron Parsons

Summary

Maps a number of inputs to a number of outputs by rotating In(N) to Out(N+M) (where M is specified on the sel input), wrapping around to Out1 when necessary.

Mask Parameters

Parameter	**Variable**	**Description**
Number of inputs	*n_inputs*	The number of parallel inputs (and outputs).

Ports

Port	**Dir.**	**Data Type**	**Description**
sync	in	Boolean	Indicates the next clock cycle contains valid data
In	in	Inherited	The stream(s) to be transposed.
sync_out	out	Boolean	Indicates that data out will be valid next clock cycle.
Out	out	Inherited	The transposed stream(s).

Description

Maps a number of inputs to a number of outputs by rotating In(N) to Out(N+M) (where M is specified on the sel input), wrapping around to Out1 when necessary.

1.3 Bit reverser *(bit_reverse)*

Block Author: Aaron Parsons
Document Author: Aaron Parsons

Summary

Reverses the bit order of the input. Input must be unsigned with binary point at position 0. Costs nothing in hardware.

Mask Parameters

Parameter	Variable	Description
No. of bits.	*n_bits*	Specifies the width of the input.

Ports

Port	Dir.	Data Type	Description
in	in	UFix_x_0	The input signal.
out	out	UFix_x_0	The output.

Description

Reverses the bit order of the input. Input must be unsigned with binary point at position 0. Costs nothing in hardware.

1.4 Complex to Real-Imag Block *(c_to_ri)*

Block Author: Aaron Parsons
Document Author: Aaron Parsons

Summary

Outputs real and imaginary components of a complex input. Useful for simplifying interconnects. See also Real-Imag to Complex. (ri_to_c, 1.42)

Mask Parameters

Parameter	Variable	Description
Bit Width	*n_bits*	Specifies width of real/imag components. Assumed equal for both components.
Binary Point	*bin_pt*	Specifies the binary point location in the real/imaginary components. Assumed equal for both components.

Ports

Port	Dir.	Data Type	Description
c	in	UFix_x_0	Complex input, real in MSB, imaginary in LSB.
r	out	Fix_x_y	Real signed output, binary point specified by parameter.
i	out	Fix_x_y	Imaginary signed output, binary point specified by parameter.

Description

Outputs real and imaginary components of a complex input. Useful for simplifying interconnects. See also Real-Imag to Complex.

1.5 Complex 4-bit Multiplier Implemented in Block RAM *(cmult_4bit_br)*

Block Author: Block Author
Document Author: Document Author

Summary

Perform a complex multiplication *(a+bi)(c-di)=(ac-bd)+(ad+bc)i.* Implements the logic in Block RAM.

Mask Parameters

Parameter	**Variable**	**Description**
Multiplier Latency	*mult_latency*	The latency through a multiplier.
Add Latency	*add_latency*	The latency through an adder.

Ports

Port	**Dir.**	**Data Type**	**Description**
a	in	Inherited	The real component of input 1.
b	in	Inherited	The imaginary component of input 1.
c	in	Inherited	The real component of input 2.
d	in	Inherited	The imaginary component of input 2.
real	out	Inherited	ac-bd
imag	out	Inherited	ad-bc

Description

Perform a complex multiplication *(a+bi)(c-di)=(ac-bd)+(ad+bc)i.* Implements the logic in Block RAM.
Each 4 bit real multiplier is implemented as a lookup table with 4b+4b=8b of address.

1.6 Conjugating Complex 4-bit Multiplier Implemented in Block RAM *(cmult_4bit_br*)*

Block Author: Block Author
Document Author: Document Author

Summary

Perform a conjugating complex multiplication *(a+bi)(c-di)=(ac+bd)+(bc-ad)i*. Implements the logic in Block RAM.

Mask Parameters

Parameter	Variable	Description
Multiplier Latency	*mult_latency*	The latency through a multiplier.
Add Latency	*add_latency*	The latency through an adder.

Ports

Port	Dir.	Data Type	Description
a	in	Inherited	The real component of input 1.
b	in	Inherited	The imaginary component of input 1.
c	in	Inherited	The real component of input 2.
d	in	Inherited	The imaginary component of input 2.
real	out	Inherited	ac+bd
imag	out	Inherited	-ad+bc

Description

Perform a conjugating complex multiplication *(a+bi)(c-di)=(ac+bd)+(bc-ad)i*. Implements the logic in Block RAM. Each 4 bit real multiplier is implemented as a lookup table with 4b+4b=8b of address.

1.7 Complex 4-bit Multiplier Implemented in Embedded Multipliers *(cmult_4bit_em)*

Block Author: Block Author
Document Author: Document Author

Summary

Perform a complex multiplication *(a+bi)(c-di)=(ac-bd)+(ad+bc)i*. Implements the logic in embedded multipliers.

Mask Parameters

Parameter	**Variable**	**Description**
Multiplier Latency	*mult_latency*	The latency through a multiplier.
dd Latency	*add_latency*	The latency through an adder.

Ports

Port	**Dir.**	**Data Type**	**Description**
a	in	Inherited	The real component of input 1.
b	in	Inherited	The imaginary component of input 1.
c	in	Inherited	The real component of input 2.
d	in	Inherited	The imaginary component of input 2.
real	out	Inherited	ac-bd
imag	out	Inherited	ad+bc

Description

Perform a complex multiplication *(a+bi)(c-di)=(ac-bd)+(ad+bc)i*. Implements the logic in embedded multipliers.

1.8 Conjugating Complex 4-bit Multiplier Implemented in Dedicated Multipliers. *(cmult_4bit_em*)*

Block Author: Block Author
Document Author: Vinayak Nagpal

Summary

Perform a conjugating complex multiplication *(a+bi)(c-di)=(ac+bd)+(bc-ad)i*. Implements the logic in dedicated multipliers.

Mask Parameters

Parameter	**Variable**	**Description**
Multiplier Latency	*mult_latency*	The latency through a multiplier.
dd Latency	*add_latency*	The latency through an adder.

Ports

Port	**Dir.**	**Data Type**	**Description**
a	in	Inherited	The real component of input 1.
b	in	Inherited	The imaginary component of input 1.
c	in	Inherited	The real component of input 2.
d	in	Inherited	The imaginary component of input 2.
real	out	Inherited	ac+bd
imag	out	Inherited	-ad+bc

Description

Perform a conjugating complex multiplication *(a+bi)(c-di)=(ac+bd)+(bc-ad)i*. Implements the logic in dedicated multipliers.

1.9 Complex 4-bit Multiplier Implemented in Slices *(cmult_4bit_sl)*

Block Author: Aaron Parsons
Document Author: Vinayak Nagpal

Summary

Perform a complex multiplication *(a+bi)(c-di)=(ac-bd)+(ad+bc)i.* Implements the logic in Slices.

Mask Parameters

Parameter	Variable	Description
Multiplier Latency	*mult_latency*	The latency through a multiplier.
Add Latency	*add_latency*	The latency through an adder.

Ports

Port	Dir.	Data Type	Description
a	in	Inherited	The real component of input 1.
b	in	Inherited	The imaginary component of input 1.
c	in	Inherited	The real component of input 2.
d	in	Inherited	The imaginary component of input 2.
real	out	Inherited	ac-bd
imag	out	Inherited	ad+bc

Description

Perform a complex multiplication *(a+bi)(c-di)=(ac-bd)+(ad+bc)i.* Implements the logic in Slices.

1.10 Conjugating Complex 4-bit Multiplier Implemented in Slices *(cmult_4bit_sl*)*

Block Author: Aaron Parsons
Document Author: Vinayak Nagpal

Summary

Perform a conjugating complex multiplication *(a+bi)(c-di)=(ac+bd)+(bc-ad)i*. Implements the logic in Slices.

Mask Parameters

Parameter	**Variable**	**Description**
Multiplier Latency	*mult_latency*	The latency through a multiplier.
Add Latency	*add_latency*	The latency through an adder.

Ports

Port	**Dir.**	**Data Type**	**Description**
a	in	Inherited	The real component of input 1.
b	in	Inherited	The imaginary component of input 1.
c	in	Inherited	The real component of input 2.
d	in	Inherited	The imaginary component of input 2.
real	out	Inherited	ac+bd
imag	out	Inherited	-ad+bc

Description

Perform a conjugating complex multiplication *(a+bi)(c-di)=(ac+bd)+(bc-ad)i*. Implements the logic in Slices.

1.11 Complex Adder/Subtractor *(complex_addsub)*

Block Author: Aaron Parsons
Document Author: Ben Blackman

Summary

This block does a complex addition and subtraction of 2 complex numbers, *a* and *b*, and spits out 2 complex numbers, *a+b* and *a-b*.

Mask Parameters

Parameter	Variable	Description
Bit Width	*BitWidth*	The number of bits in its input.
Add Latency	*add_latency*	The latency of the adders/subtractors.

Ports

Port	Dir.	Data Type	Description
a	in	2*BitWidth Fixed point	The first complex number whose higher BitWidth bits are its real part and lower BitWidth bits are its imaginary part.
b	in	2*BitWidth Fixed point	The second complex number whose higher BitWidth bits are its real part and lower BitWidth bits are its imaginary part.
a+b	out	2*BitWidth Fixed point	Upper BitWidth bits are real(a)+real(b). Lower BitWidth bits are imaginary(a)-imaginary(b).
a-b	out	2*BitWidth Fixed point	Upper BitWidth bits are imaginary(a)+imaginary(b). Lower BitWidth bits are real(b)-real(a).

Description

Usage The top output, *a+b*, is a complex output whose real part equals the sum of the real parts of *a* and *b*. The imaginary part of *a+b* equals the difference of the imaginary parts of *a* and *b*. The bottom output, *a-b*, is a complex output whose real part equals the sum of the imaginary parts of *a* and *b*.The imaginary part of *a-b* equals the difference of the real parts of *b* and *a*. The latency of this block is 2**add_latency*.

1.12 DDS *(dds)*

Block Author: Aaron Parsons
Document Author: Ben Blackman

Summary

Generates sines and cosines of different phases and outputs them in parallel.

Mask Parameters

Parameter	**Variable**	**Description**
Frequency Divisions (M)	*freq_div*	Denominator of the frequency.
Frequency (? /M*2*pi)	*freq*	Numerator of the frequency.
Parallel LOs	*num_lo*	Number of parallel local oscillators.
Bit Width	*n_bits*	Bit width of the outputs.
Latency	*latency*	Description

Ports

Port	**Dir.**	**Data Type**	**Description**
sinX	out	Fix_(n_bits)_(n_bits-1)	Sine output corresponding to the Xth local oscillator.
cosX	out	Fix_(n_bits)_(n_bits-1)	Cosine output corresponding to the Xth local oscillator.

Description

Usage There are *sin* and *cos* outputs each equal to the minimum of *num_lo* and *freq_div*. If *num_lo* >= *freq_div*/*freq*, then the outputs will be *lo_const*s. Otherwise each output will oscillate depending on the values of *freq_div* and *freq*. If the outputs oscillate, then there will be a latency of *latency* and otherwise there will be zero latency.

1.13 Decimating FIR Filter *(dec_fir)*

Block Author: Aaron Parsons
Document Author: Aaron Parsons, Ben Blackman

Summary

FIR filter which can handle multiple time samples in parallel and decimates down to 1 time sample. If coefficiencts are symmetric, it will automatically fold before multiplying.

Mask Parameters

Parameter	**Variable**	**Description**
Number of Parallel Streams	*n_inputs*	The number of time samples which arrive in parallel.
Coefficients	*coeff*	The FIR coefficients. If this vector is symmetric, the FIR will automatically fold before multiplying.
Bit Width Out	*n_bits*	The number of bits in each real/imag sample of the complex number that is output.
Quantization Behavior	*quantization*	The quantization behavior used in converting to the output bit width.
Add Latency	*add_latency*	The latency of adders/converters.
Mult Latency	*mult_latency*	The latency of multipliers.

Ports

Port	**Dir.**	**Data Type**	**Description**
sync_in	in	boolean	Takes an impulse 1 cycle before input is valid.
realX	in	Fix_(n_bits)_(n_bits-1)	Real input X
inagX	in	Fix_(n_bits)_(n_bits-1)	Imaginary input X
sync_out	out	boolean	Will be high the clock cycle before *dout* is valid.

Description

Usage User specifies the number of parallel streams to be decimated to one complex number. Inputs are multiplied by the coefficients and added together to form *dout*. Bit Width Out specifies the widths of the real and imaginary components of the complex number to be output (Ex. if Bit Width Out = 8, then dout will be 16 bits, 8 for the real and imaginary components).

1.14 The Delay in BRAM Block *(delay_bram)*

Block Author: Aaron Parsons
Document Author: Aaron Parsons

Summary

A delay block that uses BRAM for its storage.

Mask Parameters

Parameter	Variable	Description
Delay Length (DelayLen)	*DelayLen*	The length of the delay.
BRAM Latency	*bram_latency*	The latency of the underlying storage BRAM.

Ports

Port	Dir.	Data Type	Description
in	in	???	The signal to be delayed.
out	out	???	The delayed signal.

Description

A delay block that uses BRAM for its storage.

1.15 The Enabled Delay in BRAM Block *(delay_bram_en_plus)*

Block Author: Aaron Parsons
Document Author: Aaron Parsons

Summary

A delay block that uses BRAM for its storage and only shifts when enabled. However, BRAM latency cannot be enabled, so output appears bram_latency clocks after an enable.

Mask Parameters

Parameter	**Variable**	**Description**
Enabled Delays	*DelayLen*	The length of the delay.
Extra (unenabled) delay for BRAM Latency	*bram_latency*	The latency of the underlying storage BRAM.

Ports

Port	**Dir.**	**Data Type**	**Description**
in	in	???	The signal to be delayed.
en	in	???	To be asserted when input is valid.
out	out	???	The delayed signal.
valid	out	???	Asserted when output is valid.

Description

A delay block that uses BRAM for its storage and only shifts when enabled. However, BRAM latency cannot be enabled, so output appears bram_latency clocks after an enable.

1.16 The Programmable Delay in BRAM Block *(delay_bram_prog)*

Block Author: Aaron Parsons
Document Author: Aaron Parsons

Summary

A delay block that uses BRAM for its storage and has a run-time programmable delay. When delay is changed, some randomly determined samples will be inserted/dropped from the buffered stream.

Mask Parameters

Parameter	**Variable**	**Description**
Max Delay ($2^?$)	*MaxDelay*	The maximum length of the delay (i.e. the BRAM Size).
BRAM Latency	*bram_latency*	The latency of the underlying storage BRAM.

Ports

Port	**Dir.**	**Data Type**	**Description**
din	in	???	The signal to be delayed.
delay	in	???	The run-time programmable delay length.
dout	in	???	The delayed signal.

Description

A delay block that uses BRAM for its storage and has a run-time programmable delay. When delay is changed, some randomly determined samples will be inserted/dropped from the buffered stream.

1.17 The Complex Delay Block *(delay_complex)*

Block Author: Aaron Parsons
Document Author: Aaron Parsons

Summary

A delay block that treats its input as complex, splits it into real and imaginary components, delays each component by a specified amount, and then re-joins them into a complex output. The underlying storage is user-selectable (either BRAM or SLR16 elements). The reason for this is wide (36 bit) delays make adjacent multipliers in multiplier-bram pairs unusable.

Mask Parameters

Parameter	**Variable**	**Description**
Delay Depth	*delay_depth*	The length of the delay.
Bit Width	*n_bits*	Specifies the width of the real/imaginary components. Width of each component is assumed equal.
Use BRAM	*use_bram*	Set to 1 to implement the delay using BRAM. If 0, the delay will be implemented using SLR16 elements.

Ports

Port	**Dir.**	**Data Type**	**Description**
in	in	???	The complex signal to be delayed.
out	out	???	The delayed complex signal.

Description

A delay block that treats its input as complex, splits it into real and imaginary components, delays each component by a specified amount, and then re-joins them into a complex output. The underlying storage is user-selectable (either BRAM or SLR16 elements). The reason for this is wide (36 bit) delays make adjacent multipliers in multiplier-bram pairs unusable.

1.18 The Delay in Slices Block *(delay_slr)*

Block Author: Aaron Parsons
Document Author: Aaron Parsons

Summary

A delay block that uses slices (SLR16s) for its storage.

Mask Parameters

Parameter	Variable	Description
Delay Length	*DelayLen*	The length of the delay.

Ports

Port	Dir.	Data Type	Description
in	in	???	The signal to be delayed.
out	out	???	The delayed signal.

Description

A delay block that uses slices (SLR16s) for its storage.

1.19 DRAM Vector Accumulator *(dram_vacc)*

Block Author: Arash Parsa
Document Author: Jason Manley

Summary

A vector accumulator for very large vector lengths using the BEE2's DRAM.

Mask Parameters

Parameter	**Variable**	**Description**
Parameter Name	*Variable Name*	Parameter Description
Parameter Name	*Variable Name*	Parameter Description

Ports

Port	**Dir.**	**Data Type**	**Description**
Port Name	port direction	Port data type	Port Description
Port Name	in	ufix_x_y	Port Description
Port Name	in	inherited	Port Description

Description

Block Description can also include arbitrary LaTeX like math $a^2 + b^2 = \phi^2$.

1.20 DRAM Vector Accumulator Test Vector Generator *(dram_vacc_tvg)*

Block Author: Jason Manley, Arash Parsa
Document Author: Jason Manley

Summary

Comprehensive TVG for the DRAM Vector Accumulator.

Mask Parameters

Parameter	Variable	Description
Parameter Name	*Variable Name*	Parameter Description
Parameter Name	*Variable Name*	Parameter Description

Ports

Port	Dir.	Data Type	Description
Port Name	port direction	Port data type	Port Description
Port Name	in	ufix_x_y	Port Description
Port Name	in	inherited	Port Description

Description

Block Description can also include arbitrary LaTeX like math $a^2 + b^2 = \phi^2$.

1.21 The Edge Detect Block *(edge)*

Block Author: Aaron Parsons
Document Author: Aaron Parsons

Summary

Outputs true if a boolean input signal is not equal to its value last clock.

Mask Parameters

Parameter	**Variable**	**Description**

Ports

Port	**Dir.**	**Data Type**	**Description**
in	in	Boolean	Input boolean signal.
out	out	Boolean	Edge detected output boolean signal.

Description

Outputs true if a boolean input signal is not equal to its value last clock.

1.22 FFT *(fft)*

Block Author: Aaron Parsons
Document Author: Aaron Parsons

Summary

Computes the Fast Fourier Transform with 2^N channels for time samples presented 2^P at a time in parallel. Uses a biplex FFT architecture under the hood which has been extended to handled time samples in parallel. For $P = 0$, this block accepts two independent, parallel streams (labelled as pols) and computes the FFT of each independently (the biplex architecture provides this for free). Data is output in normal frequency order, meaning that channel 0 (corresponding to DC) is output first, followed by channel 1, on up to channel $2^N - 1$ (which can be interpreted as channel -1). When multiple time samples are presented in parallel on the input, multiple frequency samples are output in parallel.

Mask Parameters

Parameter	**Variable**	**Description**
Size of FFT: ($2^?$)	*FFTSize*	The number of channels in the FFT.
Bit Width	*BitWidth*	The number of bits in each real and imaginary sample as they are carried through the FFT. Each FFT stage will round numbers back down to this number of bits after performing a butterfly computation.
Number of Simultaneous Inputs: ($2^?$)	*n_inputs*	The number of parallel time samples which are presented to the FFT core each clock. The number of output ports are set to this same value.
Quantization Behavior	*quantization*	Specifies the rounding behavior used at the end of each butterfly computation to return to the number of bits specified above.
Overflow Behavior	*overflow*	Indicates the behavior of the FFT core when the value of a sample exceeds what can be expressed in the specified bit width.
Add Latency	*add_latency*	Latency through adders in the FFT.
Mult Latency	*mult_latency*	Latency through multipliers in the FFT.
BRAM Latency	*bram_latency*	Latency through BRAM in the FFT.

Ports

Port	Dir.	Data Type	Description
sync	in	Boolean	Indicates the next clock cycle contains valid data
shift	in	Unsigned	Sets the shifting schedule through the FFT. Bit 0 specifies the behavior of stage 0, bit 1 of stage 1, and so on. If a stage is set to shift (with bit = 1), that every sample is divided by 2 at the output of that stage.
In	in	Inherited	The time-domain stream(s) to channelized.
sync_out	out	Boolean	Indicates that data out will be valid next clock cycle.
of	out	Boolean	Indicates an overflow occurred at some stage in the FFT.
Out	out	Inherited	The frequency channels.

Description

Computes the Fast Fourier Transform with 2^N channels for time samples presented 2^P at a time in parallel. Uses a biplex FFT architecture under the hood which has been extended to handled time samples in parallel. For $P = 0$, this block accepts two independent, parallel streams (labelled as pols) and computes the FFT of each independently (the biplex architecture provides this for free). Data is output in normal frequency order, meaning that channel 0 (corresponding to DC) is output first, followed by channel 1, on up to channel $2^N - 1$ (which can be interpreted as channel -1). When multiple time samples are presented in parallel on the input, multiple frequency samples are output in parallel.

1.23 Real-sampled Biplex FFT (with output demuxed by 2) *(fft_biplex_rea*

Block Author: Aaron Parsons
Document Author: Aaron Parsons

Summary

Computes the real-sampled Fast Fourier Transform using the standard Hermitian conjugation trick to use a complex core to transform a two real streams. Thus, a biplex core (which can do 2 complex FFTs) can transform 4 real streams. Only positive frequencies are output (negative frequencies are the mirror images of their positive counterparts). Data is output in normal frequency order, meaning that channel 0 (corresponding to DC) is output first, followed by channel 1, on up to channel $2^{N-1} - 1$. Real inputs 1 and 2 share one output port (with the data for 1 coming first, then the data for 2), and likewise for inputs 3 and 4.

Mask Parameters

Parameter	Variable	Description
Size of FFT: ($2^?$)	*FFTSize*	The number of channels computed in the complex FFT core. The number of channels output for each real stream is half of this.
Bit Width	*BitWidth*	The number of bits in each real and imaginary sample as they are carried through the FFT. Each FFT stage will round numbers back down to this number of bits after performing a butterfly computation.
Quantization Behavior	*quantization*	Specifies the rounding behavior used at the end of each butterfly computation to return to the number of bits specified above.
Overflow Behavior	*overflow*	Indicates the behavior of the FFT core when the value of a sample exceeds what can be expressed in the specified bit width.
Add Latency	*add_latency*	Latency through adders in the FFT.
Mult Latency	*mult_latency*	Latency through multipliers in the FFT.
BRAM Latency	*bram_latency*	Latency through BRAM in the FFT.

Ports

Port	Dir.	Data Type	Description
sync	in	Boolean	Indicates the next clock cycle contains valid data
shift	in	Unsigned	Sets the shifting schedule through the FFT. Bit 0 specifies the behavior of stage 0, bit 1 of stage 1, and so on. If a stage is set to shift (with bit = 1), that every sample is divided by 2 at the output of that stage.
pol	in	Inherited	The time-domain stream(s) to channelized.
sync_out	out	Boolean	Indicates that data out will be valid next clock cycle.
of	out	Boolean	Indicates an overflow occurred at some stage in the FFT.
pol_out	out	Inherited	The frequency channels.

Description

Computes the real-sampled Fast Fourier Transform using the standard Hermitian conjugation trick to use a complex core to transform a two real streams. Thus, a biplex core (which can do 2 complex FFTs) can transform 4 real streams. Only positive frequencies are output (negative frequencies are the mirror images of their positive counterparts). Data is output in normal frequency order, meaning that channel 0 (corresponding to DC) is output first, followed by channel 1, on up to channel $2^{N-1} - 1$. Real inputs 1 and 2 share one output port (with the data for 1 coming first, then the data for 2), and likewise for inputs 3 and 4.

1.24 Real-sampled Biplex FFT (with output demuxed by 4) *(fft_biplex_rea*

Block Author: Aaron Parsons
Document Author: Aaron Parsons

Summary

Computes the real-sampled Fast Fourier Transform using the standard Hermitian conjugation trick to use a complex core to transform a two real streams. Thus, a biplex core (which can do 2 complex FFTs) can transform 4 real streams. All frequencies (both positive and negative) are output (negative frequencies are the mirror images of their positive counterparts). Data is output in normal frequency order, meaning that channel 0 (corresponding to DC) is output first, followed by channel 1, on up to channel $2^N - 1$.

Mask Parameters

Parameter	**Variable**	**Description**
Size of FFT: ($2^?$)	*FFTSize*	The number of channels computed in the FFT core.
Bit Width	*BitWidth*	The number of bits in each real and imaginary sample as they are carried through the FFT. Each FFT stage will round numbers back down to this number of bits after performing a butterfly computation.
Quantization Behavior	*quantization*	Specifies the rounding behavior used at the end of each butterfly computation to return to the number of bits specified above.
Overflow Behavior	*overflow*	Indicates the behavior of the FFT core when the value of a sample exceeds what can be expressed in the specified bit width.
Add Latency	*add_latency*	Latency through adders in the FFT.
Mult Latency	*mult_latency*	Latency through multipliers in the FFT.
BRAM Latency	*bram_latency*	Latency through BRAM in the FFT.

Ports

Port	**Dir.**	**Data Type**	**Description**
sync	in	Boolean	Indicates the next clock cycle contains valid data
shift	in	Unsigned	Sets the shifting schedule through the FFT. Bit 0 specifies the behavior of stage 0, bit 1 of stage 1, and so on. If a stage is set to shift (with bit = 1), that every sample is divided by 2 at the output of that stage.
pol	in	Inherited	The time-domain stream(s) to channelized.
sync_out	out	Boolean	Indicates that data out will be valid next clock cycle.
of	out	Boolean	Indicates an overflow occurred at some stage in the FFT.
pol_out	out	Inherited	The frequency channels.

Description

Computes the real-sampled Fast Fourier Transform using the standard Hermitian conjugation trick to use a complex core to transform a two real streams. Thus, a biplex core (which can do 2 complex FFTs) can transform 4 real streams. All frequencies (both positive and negative) are output (negative frequencies are the mirror images of their positive counterparts). Data is output in normal frequency order, meaning that channel 0 (corresponding to DC) is output first, followed by channel 1, on up to channel $2^N - 1$.

1.25 Real-sampled Wideband FFT *(fft_wideband_real)*

Block Author: Aaron Parsons
Document Author: Aaron Parsons

Summary

Computes the real-sampled Fast Fourier Transform using the standard Hermitian conjugation trick to use a complex core to transform a single real stream using half the normal resources (this requires at least 4 time samples in parallel). Only positive frequencies are output (negative frequencies are the mirror images of their positive counterparts), so there the number of output ports is half the number of input ports. Uses a biplex FFT architecture under the hood which has been extended to handled time samples in parallel. Data is output in normal frequency order, meaning that channel 0 (corresponding to DC) is output first, followed by channel 1, on up to channel $2^{N-1} - 1$.

Mask Parameters

Parameter	**Variable**	**Description**
Size of FFT: ($2^?$)	*FFTSize*	The number of channels in the complex FFT core. The number of positive frequency channels output is half of this.
Bit Width	*BitWidth*	The number of bits in each real and imaginary sample as they are carried through the FFT. Each FFT stage will round numbers back down to this number of bits after performing a butterfly computation.
Number of Simultaneous Inputs: ($2^?$)	*n_inputs*	The number of parallel time samples which are presented to the FFT core each clock. This must be at least 2^2. The number of output ports is half of this value.
Quantization Behavior	*quantization*	Specifies the rounding behavior used at the end of each butterfly computation to return to the number of bits specified above.
Overflow Behavior	*overflow*	Indicates the behavior of the FFT core when the value of a sample exceeds what can be expressed in the specified bit width.
Add Latency	*add_latency*	Latency through adders in the FFT.
Mult Latency	*mult_latency*	Latency through multipliers in the FFT.
BRAM Latency	*bram_latency*	Latency through BRAM in the FFT.

Ports

Port	Dir.	Data Type	Description
sync	in	Boolean	Indicates the next clock cycle contains valid data
shift	in	Unsigned	Sets the shifting schedule through the FFT. Bit 0 specifies the behavior of stage 0, bit 1 of stage 1, and so on. If a stage is set to shift (with bit = 1), that every sample is divided by 2 at the output of that stage.
In	in	Inherited	The time-domain stream(s) to channelized.
sync_out	out	Boolean	Indicates that data out will be valid next clock cycle.
of	out	Boolean	Indicates an overflow occurred at some stage in the FFT.
Out	out	Inherited	The frequency channels.

Description

Computes the real-sampled Fast Fourier Transform using the standard Hermitian conjugation trick to use a complex core to transform a single real stream using half the normal resources (this requires at least 4 time samples in parallel). Only positive frequencies are output (negative frequencies are the mirror images of their positive counterparts), so there the number of output ports is half the number of input ports. Uses a biplex FFT architecture under the hood which has been extended to handled time samples in parallel. Data is output in normal frequency order, meaning that channel 0 (corresponding to DC) is output first, followed by channel 1, on up to channel $2^{N-1} - 1$.

1.26 FIR Column *(fir_col)*

Block Author: Aaron Parsons
Document Author: Ben Blackman

Summary

Takes in real and imaginary numbers to be multiplied by the coefficients and then the filter sums the real and imaginary parts separately. Then both sums are output as well as a delayed version of the unchanged inputs.

Mask Parameters

Parameter	**Variable**	**Description**
Inputs	*n_inputs*	The number of real inputs and the number of imaginary inputs.
Coefficients	*coeff*	A vector of coefficients of this FIR. Should be the same number of coefficients as inputs.
Add Latency	*add_latency*	The latency of the internal adders.
Mult Latency	*mult_latency*	The latency of the internal multipliers.

Ports

Port	**Dir.**	**Data Type**	**Description**
realX	in	Inherited	This is real input X. Its data type is inherited from the previous block.
imagX	in	Inherited	This is imaginary input X. Its data type is inherited from the previous block.
real_outX	out	Inherited	This output is *realX* delayed by 1 cycle.
imag_outX	out	Inherited	This output is *imagX* delayed by 1 cycle.
real_sum	out	Inherited	This is the sum of all the *realX* * coefficient X.
imag_sum	out	Inherited	This is the sum of all the *imagX* * coefficient X.

Description

Usage This block takes in a number of inputs in parallel and outputs a delayed version of them and also multiplies the inputs by the coefficients. Then *real_sum* and *imag_sum* are computed and are delayed due to the latency in the adders which depends both on the *add_latency* and the number of inputs.

1.27 FIR Double Column *(fir_dbl_col)*

Block Author: Aaron Parsons
Document Author: Ben Blackman

Summary

Takes in real and imaginary numbers to be multiplied by the coefficients and then the filter sums the real and imaginary parts separately. Then both sums are output as well as a delayed version of the unchanged inputs.

Mask Parameters

Parameter	**Variable**	**Description**
Inputs	*n_inputs*	The number of real inputs and the number of imaginary inputs.
Coefficients	*coeff*	A vector of coefficients of this FIR. Should be the same number of coefficients as inputs.
Add Latency	*add_latency*	The latency of the internal adders.
Mult Latency	*mult_latency*	The latency of the internal multipliers.

Ports

Port	**Dir.**	**Data Type**	**Description**
real	in	Inherited	This real input is to be multiplied by one of the coefficients.
imag	in	Inherited	This imaginary input is to be multiplied by one of the coefficients.
real_back	in	Inherited	These real inputs correspond to the second half of the input stream. They get added to one of the *real* inputs before being multiplied by the coefficient.
imag_back	in	Inherited	These imaginary inputs correspond to the second half of the input stream. They get added to one of the *imag* inputs before being multiplied by the coefficient.
real_out	out	Inherited	This output is *real* delayed by 1 cycle.
imag_out	out	Inherited	This output is *imag* delayed by 1 cycle.
real_back_out	out	Inherited	This output is *real_back* delayed by 1 cycle.
imag_back_out	out	Inherited	This output is *imag_back* delayed by 1 cycle.
real_sum	out	Inherited	This is the sum of all the multiplications between *real* and *real_back* and their corresponding coefficients.
imag_sum	out	Inherited	This is the sum of all the multiplications between *imag* and *imag_back* and their corresponding coefficients.

Description

Usage This block takes in a number of inputs in parallel and outputs a delayed version of them and also multiplies the inputs by the coefficients. Then *real_sum* and *imag_sum* are computed and are delayed due to the latency in the adders which depends both on the *add_latency* and the number of inputs. For example, if you choose the number of inputs to be 2, then there will be 2 *real* and 2 *real_back* input ports along with 2 *imag* and 2 *imag_back* input ports. The FIR Double Column blocks takes advantage of the symmetric filter tap coefficients by adding the first and last inputs together before multiplying by the coefficient. This results in a more efficient FIR filter column.

1.28 FIR Tap *(fir_tap)*

Block Author: Aaron Parsons
Document Author: Ben Blackman

Summary

This block multiplies both inputs by *factor* and outputs the result immediately after the multiply and outputs a delayed copy of the input by 1 cycle,

Mask Parameters

Parameter	**Variable**	**Description**
Factor	*factor*	The value that multiplies both inputs.
Mult latency	*latency*	The latency of the multiplier.

Ports

Port	**Dir.**	**Data Type**	**Description**
a	in	Inherited	The first number to be multiplied by *factor*. It usually is the real component of an input.
b	in	Inherited	The second number to be multiplied by *factor*. It usually is the imaginary component of an input.
a_out	out	Inherited	The input *a* delayed by 1 cycle.
b_out	out	Inherited	The input *b* delayed by 1 cycle.
real	out	Inherited	The result of the multiplication of *a* with *factor*.
imag	out	Inherited	The result of the multiplication of *b* with *factor*.

Description

Usage *a_out* and *b_out* are 1 cycle delayed versions of *a* and *b*, respectively. *real* and *imag* are the results of *a* * *factor* and *b* * *factor*, respectively. The delay from *a* to *real* or *b* to *imag* is equal to *latency*.

1.29 The Freeze Counter Block *(freeze_cntr)*

Block Author: Aaron Parsons
Document Author: Aaron Parsons

Summary

A freeze counter is an enabled counter which holds its final value (regardless of enables) until it is reset.

Mask Parameters

Parameter	Variable	Description
Counter Length ($2^?$)	*CounterBits*	Specifies the number of bits (and the final count output of 2^{bits-1}).

Ports

Port	Dir.	Data Type	Description
en	in	???	Step the counter by 1 unless addr=2^{bits-1}.
rst	in	???	Reset counter to 0.
addr	out	???	Current output of the counter.
we	out	Boolean	Outputs boolean true just before addr is incremented.
done	out	Boolean	Outputs boolean true when a final en is asserted and addr=2^{bits-1}.

Description

A freeze counter is an enabled counter which holds its final value (regardless of enables) until it is reset. Thus, a 2^5 freeze counter will count from 0 to 31 on 31 enables, but will hold 31 thereafter until a reset occurs. This block is useful for writing data in a single pass to memory without looping.

1.30 Local Oscillator Constant *(lo_const)*

Block Author: Aaron Parsons
Document Author: Ben Blackman

Summary

Gives the sine and cosine of a desired constant phase.

Mask Parameters

Parameter	**Variable**	**Description**
Output Bitwidth	*n_bits*	Bitwidth of the outputs.
Phase (0 to 2*pi)	*phase*	The phase value for which the sine and cosine are evaluated.

Ports

Port	**Dir.**	**Data Type**	**Description**
sin	out	Fix_(n_bits)_(n_bits-1)	The sine of the given phase value.
cos	out	Fix_(n_bits)_(n_bits-1)	The cosine of the given phase value.

Description

Usage This block gives the sine and cosine of a user-specified, constant phase value with a user-specified bitwidth.

1.31 Local Oscillator *(lo_osc)*

Block Author: Aaron Parsons
Document Author: Ben Blackman

Summary

Generates an oscillating sine and cosine.

Mask Parameters

Parameter	**Variable**	**Description**
Output Bitwidth	*n_bits*	Bitwidth of the outputs.
Counter Step	*counter_step*	Step size of the internal counter.
Counter Start Value	*counter_start*	Initial value of the internal counter.
Counter Bitwidth	*counter_width*	Bitwidth of the internal counter.
Latency	*latency*	The latency of the block.

Ports

Port	**Dir.**	**Data Type**	**Description**
sin	out	Fix_(n_bits)_(n_bits-1)	Sine of the current phase, which is given by the counter.
cos	out	Fix_(n_bits)_(n_bits-1)	Cosine of the current phase, which is given by the counter.

Description

Usage This block generates the sine and cosine of an oscillator with user-defined spacing (based on *counter_step* and *counter_width*) and bitwidth.

1.32 Mixer *(mixer)*

Block Author: Aaron Parsons
Document Author: Aaron Parsons, Ben Blackman

Summary

Digitally mixes an input signal (which can be several samples in parallel) with an LO of the indicated frequency (which is some fraction of the native FPGA clock rate).

Mask Parameters

Parameter	**Variable**	**Description**
Frequency Divisions	*freq_div*	The (power of 2) denominator of the mixing frequency.
Mixing Frequency	*freq*	The numerator of the mixing frequency.
Number of Parallel Streams	*nstreams*	The number of samples that arrive in parallel.
Bit Width	*n_bits*	The bitwidth of LO samples.
BRAM Latency	*bram_latency*	The latency of sin/cos lookup table.
MULT Latency	*mult_latency*	The latency of mixing multipliers.

Ports

Port	**Dir.**	**Data Type**	**Description**
sync	in	boolean	Takes in an impulse the cycle before the *din*s are valid.
dinX	in	Fix_8_7	Input X to be mixed and output on *realX* and *imagX*.
sync_out	out	boolean	This signal will be high the cycle before the data coming out is valid.
realX	out	Fix_(n_bits)_(n_bits-1)	Real output of mixed *dinX*.
imagX	out	Fix_(n_bits)_(n_bits-1)	Imaginary output of mixed *dinX*.

Description

Usage *Mixer* mixes the incoming data and produces both real and imaginary outputs.
M = Frequency Divisions
F = Mixing Frequency
M and F must both be integers, and M must be a power of 2. The ratio F/M should equal the ratio f/r where r is the data rate of the sampled signal. For example, an F/M of 3/16 would downmix an 800Msps signal with an LO of 150MHz.

1.33 The Negative Edge Detect Block *(negedge)*

Block Author: Aaron Parsons
Document Author: Aaron Parsons

Summary

Outputs true if a boolean input signal is currently false, but was true last clock.

Mask Parameters

Parameter	Variable	Description

Ports

Port	Dir.	Data Type	Description
in	in	Boolean	Input boolean signal.
out	out	Boolean	Negative-edge detected output boolean signal.

Description

Outputs true if a boolean input signal is currently false, but was true last clock.

1.34 The Partial Delay Block *(partial_delay)*

Block Author: Aaron Parsons
Document Author: Aaron Parsons

Summary

For a set of parallel inputs which represent consecutive time samples of the same input signal, this block delays the stream by a dynamically selectable number of samples between 0 and (n_inputs-1).

Mask Parameters

Parameter	**Variable**	**Description**
No. of inputs.	*n_inputs*	The number of parallel inputs.
Mux Latency	*latency*	The latency of each mux block.

Ports

Port	**Dir.**	**Data Type**	**Description**
sync	???	???	Indicates the next clock cycle containing valid data
din	in	???	A number to be summed.

Description

For a set of parallel inputs which represent consecutive time samples of the same input signal, this block delays the stream by a dynamically selectable number of samples between 0 and (n_inputs-1). This is useful for blocks such as the ADC that present several samples in parallel because sampling occurs at a higher clock rate than that of the FPGA.

...	4	0	...	→	6	2	...	...
...	5	1	...	→	7	3	...	...
...	6	2	...	→	...	4	0	...
...	7	3	...	→	...	5	1	...

Table 1.1: Mapping of 4 parallel input samples to output for delay = 2

1.35 Polyphase FIR Filter (frontend for a full PFB) *(pfb_fir)*

Block Author: Aaron Parsons
Document Author: Aaron Parsons

Summary

This block, combined with an FFT, implements a Polyphase Filter Bank which uses longer windows of data to improve the shape of channels within a spectrum.

Mask Parameters

Parameter	Variable	Description
Size of PFB: $(2^?)$	*PFBSize*	The number of channels in the PFB (this should also be the size of the FFT which follows).
Total Number of Taps:	*TotalTaps*	The number of taps in the PFB FIR filter. Each tap uses 2 real multiplier cores and requires buffering the real and imaginary streams for $2^{PFBSize}$ samples.
Windowing Function	*WindowType*	Which windowing function to use (this allows trading passband ripple for steepness of rolloff, etc).
Number of Simultaneous Inputs: $(2^?)$	*n_inputs*	The number of parallel time samples which are presented to the FFT core each clock. The number of output ports are set to this same value.
Make Biplex	*MakeBiplex*	Double up the inputs to match with a biplex FFT.
Input Bit Width	*BitWidthIn*	The number of bits in each real and imaginary sample input to the PFB.
Output Bit Width	*BitWidthOut*	The number of bits in each real and imaginary sample output from the PFB. This should match the bit width in the FFT that follows.
Coefficient Bit Width	*CoeffBitWidth*	The number of bits in each coefficient. This is usually chosen to match the input bit width.
Use Distributed Memory for Coefficients	*CoeffDistMem*	Store the FIR coefficients in distributed memory (if = 1). Otherwise, BRAMs are used to hold the coefficients.
Add Latency	*add_latency*	Latency through adders in the FFT.
Mult Latency	*mult_latency*	Latency through multipliers in the FFT.
BRAM Latency	*bram_latency*	Latency through BRAM in the FFT.
Quantization Behavior	*quantization*	Specifies the rounding behavior used at the end of each butterfly computation to return to the number of bits specified above.
Bin Width Scaling (normal = 1)	*fwidth*	PFBs give enhanced control over the width of frequency channels. By adjusting this parameter, you can scale bins to be wider (for values ¿ 1) or narrower (for values ¡ 1).

Ports

Port	Dir.	Data Type	Description
sync	in	Boolean	Indicates the next clock cycle contains valid data
pol_in	in	Inherited	The (complex) time-domain stream(s).
sync_out	out	Boolean	Indicates that data out will be valid next clock cycle.
pol_out	out	Inherited	The (complex) PFB FIR output, which is still a time-domain signal.

Description

This block, combined with an FFT, implements a Polyphase Filter Bank which uses longer windows of data to improve the shape of channels within a spectrum.

1.36 Polyphase Real FIR Filter *(pfb_fir_real)*

Block Author: Henry Chen
Document Author: Ben Blackman

Summary

This block, combined with an FFT, implements a real Polyphase Filter Bank which uses longer windows of data to improve the shape of channels within a spectrum.

Mask Parameters

Parameter	Variable	Description
Size of PFB ($2^?$ pnts)	*PFBSize*	The number of channels in the PFB (this should also be the size of the FFT which follows).
Total Number of Taps	*TotalTaps*	The number of taps in the PFB FIR filter. Each tap uses 2 real multiplier cores and requires buffering the real and imaginary streams for $2^{PFBSize}$ samples.
Windowing Function	*WindowType*	Which windowing function to use (this allows trading passband ripple for steepness of rolloff, etc).
Number of Simultaneous Inputs ($2^?$)	*n_inputs*	The number of parallel time samples which are presented to the FFT core each clock. The number of output ports are set to this same value.
Make Biplex	*MakeBiplex*	Double up the inputs to match with a biplex FFT.
Input Bitwidth	*BitWidthIn*	The number of bits in each real and imaginary sample input to the PFB.
Output Bitwidth	*BitWidthOut*	The number of bits in each real and imaginary sample output from the PFB. This should match the bit width in the FFT that follows.
Coefficient Bitwidth	*CoeffBitWidth*	The number of bits in each coefficient. This is usually chosen to match the input bit width.
Use Distributed Memory for Coeffs	*CoeffDistMem*	Store the FIR coefficients in distributed memory (if $= 1$). Otherwise, BRAMs are used to hold the coefficients.
Add Latency	*add_latency*	Latency through adders in the FFT.
Mult Latency	*mult_latency*	Latency through multipliers in the FFT.
BRAM Latency	*bram_latency*	Latency through BRAM in the FFT.
Quantization Behavior	*quantization*	Specifies the rounding behavior used at the end of each butterfly computation to return to the number of bits specified above.
Bin Width Scaling (normal=1)	*fwidth*	PFBs give enhanced control over the width of frequency channels. By adjusting this parameter, you can scale bins to be wider (for values $>$ 1) or narrower (for values $<$ 1).

Ports

Port	Dir.	Data Type	Description
sync	in	Boolean	Indicates the next clock cycle contains valid data
pol_in	in	Inherited	The (real) time-domain stream(s).
sync_out	out	Boolean	Indicates that data out will be valid next clock cycle.
pol_out	out	Inherited	The (real) PFB FIR output, which is still a time-domain signal.

Description

Usage This block, combined with an FFT, implements a real Polyphase Filter Bank which uses longer windows of data to improve the shape of channels within a spectrum.

1.37 The Positive Edge Detect Block *(posedge)*

Block Author: Aaron Parsons
Document Author: Aaron Parsons

Summary

Outputs true if a boolean input signal is true this clock and was false last clock.

Mask Parameters

Parameter	**Variable**	**Description**

Ports

Port	**Dir.**	**Data Type**	**Description**
in	in	Boolean	Input boolean signal.
out	out	Boolean	Positive-edge detected output boolean signal.

Description

Outputs true if a boolean input signal is true this clock and was false last clock.

1.38 Power *(power)*

Block Author: Aaron Parsons
Document Author: Ben Blackman

Summary

Computes the power of a complex number.

Mask Parameters

Parameter	Variable	Description
Bit Width	*BitWidth*	The number of bits in its input.

Ports

Port	Dir.	Data Type	Description
c	in	2*BitWidth Fixed point	A complex number whose higher BitWidth bits are its real part and lower BitWidth bits are its imaginary part.
power	out	UFix_(2*BitWidth)_(2*BitWidth-1)	The computed power of the input complex number.

Description

Usage The power block typically has a latency of 5 and will compute the power of its input by taking the sum of the squares of its real and imaginary components.

1.39 The Pulse Extender Block *(pulse_ext)*

Block Author: Aaron Parsons
Document Author: Aaron Parsons

Summary

Extends a boolean signal to be high for the specified number of clocks after the last high input.

Mask Parameters

Parameter	Variable	Description
Length of Pulse	*pulse_len*	Specifies number of clocks after the last high input for which the output is held high.

Ports

Port	Dir.	Data Type	Description
in	in	Boolean	Input boolean signal.
out	out	Boolean	Pulse-extended boolean signal.

Description

Extends a boolean signal to be high for the specified number of clocks after the last high input. If a new in pulse (input high) occurs, the counter determining the output pulse length is reset.

1.40 RC Multiplier *(rcmult)*

Block Author: Aaron Parsons
Document Author: Ben Blackman

Summary

Takes an input and sine and cosine value and gives out both real and imaginary outputs.

Mask Parameters

Parameter	**Variable**	**Description**
Latency	*latency*	The latency of the multipliers and of the *rcmult* block.

Ports

Port	**Dir.**	**Data Type**	**Description**
d	in	Inherited	The input to be multiplied by sine and cosine values.
sin	in	Inherited	The sine value used to multiply d and generate the *imag* output.
cos	in	Inherited	The cosine value used to multiply d and generate the *real* output.
real	out	Inherited	The result of multiplying d with *cos*.
imag	out	Inherited	The result of multiplying d with *sin*.

Description

Usage This *rcmult* block takes an input value, d, and computes the real and imaginary components by multiplying by the *cos* and *sin*, respectively. The block has a delay of *latency* associated with it.

1.41 Reorder *(reorder)*

Block Author: Aaron Parsons
Document Author: Aaron Parsons

Summary

Permutes a vector of samples to into the desired order.

Mask Parameters

Parameter	Variable	Description
Output Order	*map*	Assuming an input order of 0, 1, 2, ..., this is a vector of the desired output order (e.g. [0 1 2 3]).
No. of inputs.	*n_inputs*	The number of parallel streams to which this reorder should be applied.
BRAM Latency	*bram_latency*	The latency of the BRAM buffer.
Map Latency	*map_latency*	The latency allowed for the combinatorial logic required for mapping a counter to the desired output order. If your permutation can be acheived by simply reordering bits (as is the case for bit reversed order, reverse order, and matrix tranposes with power-of-2 dimensions), a map latency of 0 is appropriate. Otherwise, 1 or 2 is a good idea.
Double Buffer	*double_buffer*	By default, this block uses single buffering (meaning it uses a buffer only the size of the vector, and permutes the data order in place). You can override this by setting this parameter to 1, in which case 2 buffers are used to permute the vector (saving logic resources at the expense of BRAM).

Ports

Port	Dir.	Data Type	Description
sync	in	Boolean	Indicates the next clock cycle contains valid data
en	in	Boolean	Indicates the current input data is valid.
din	in	Inherited	The data stream(s) to be permuted.
sync_out	out	Boolean	Indicates that data out will be valid next clock cycle.
valid	out	Boolean	Indicates the current output data is valid.
dout	out	Inherited	The permuted data stream(s).

Description

Permutes a vector of samples into the desired order. By default, this block uses a single buffer to do this. As vectors are permuted, the data placement in memory will go through several orders before it repeats. For large orders (> 16) you should consider using double buffering, but otherwise, this block saves BRAM resources with only a modest increase in logic resources.

1.42 The Real-Imag to Complex Block *(ri_to_c)*

Block Author: Aaron Parsons
Document Author: Aaron Parsons

Summary

Concatenates real and imaginary inputs into a complex output. Useful for simplifying interconnects. See also: Complex to Real-Imag Block (c_to_ri, 1.4)

Mask Parameters

Parameter	Variable	Description

Ports

Port	Dir.	Data Type	Description
r	in	Fix_x_y	Real data
i	in	Fix_x_y	Imaginary signed output, binary point specified by parameter.
c	out	UFix_x_0	Complex input, real in MSB, imaginary in LSB.

Description

Conveniently combines real and imaginary components of a number into a single wire. See also: Complex to Real-Imag Block (c_to_ri, 1.4)

1.43 Square Transposer *(square_transposer)*

Block Author: Aaron Parsons
Document Author: Aaron Parsons

Summary

Presents a number of parallel inputs serially on the same number of output lines.

Mask Parameters

Parameter	Variable	Description
Number of inputs	*n_inputs*	The number of parallel inputs (and outputs).

Ports

Port	Dir.	Data Type	Description
sync	in	Boolean	Indicates the next clock cycle contains valid data
In	in	Inherited	The stream(s) to be transposed.
sync_out	out	Boolean	Indicates that data out will be valid next clock cycle.
Out	out	Inherited	The transposed stream(s).

Description

Presents a number of parallel inputs serially on the same number of output lines. After a sync pulse, all of the parallel streams input to the square transposer will appear serially on Out1. The all parallel data from the following clock cycle will appear serially on Out2, and so on. All the data output (Out1, Out2, etc.) appear aligned:

In1	d12	d8	d4	d0	→	d3	d2	d1	d0	Out1
In2	d13	d9	d5	d1	→	d7	d6	d5	d4	Out2
In3	d14	d10	d6	d2	→	d11	d10	d9	d8	Out3
In4	d15	d11	d7	d3	→	d15	d14	d13	d12	Out4

1.44 Stopwatch *(stopwatch)*

Block Author: Jason Manley
Document Author: Jason Manley

Summary

Counts the number of clocks between a start and stop pulse.

Mask Parameters

Parameter	Variable	Description
None	*None*	This block has no parameters

Ports

Port	Dir.	Data Type	Description
start	in	boolean	Start counting
stop	in	boolean	Stop counting and hold value until reset received
reset	in	boolean	Reset back to zero.
count_out	out	ufix_32_0	Number of clocks elapsed since start pulse received.

Description

This block counts the number of clocks between a start and stop pulse. This value is held until a reset is received. If another start pulse is received before the reset, counting resumes from where it left-off. If a reset is received mid-way through a count (ie before a stop pulse) then the stopwatch will be reset and await another start pulse before it restarts counting. Test Model: test_stopwatch.mdl

1.45 Display name for Block *(simulink_name_for_block)*

Block Author: Block Author
Document Author: Document Author

Summary

Block Summary can contain arbitrary LaTeX like lists

- list item 1
- list item 2

Mask Parameters

Parameter	**Variable**	**Description**
Parameter Name	*Variable Name*	Parameter Description
Parameter Name	*Variable Name*	Parameter Description

Ports

Port	**Dir.**	**Data Type**	**Description**
Port Name	port direction	Port data type	Port Description
Port Name	in	ufix_x_y	Port Description
Port Name	in	inherited	Port Description

Description

Block Description can also include arbitrary LaTeX like math $a^2 + b^2 = \phi^2$.

1.46 The Enabled Sync Delay Block *(sync_delay_en)*

Block Author: Aaron Parsons
Document Author: Aaron Parsons

Summary

Delay an infrequent boolean pulse by the specified number of enabled clocks.

Mask Parameters

Parameter	**Variable**	**Description**
Delay Length	*DelayLen*	The length of the delay.

Ports

Port	**Dir.**	**Data Type**	**Description**
in	in	boolean	The boolean signal to be delayed.
en	in	boolean	To be asserted when input is valid.
out	out	boolean	The delayed boolean signal, output 1 clock after en.

Description

Delay an infrequent boolean pulse by the specified number of enabled clocks. If the input pulse repeats before the output pulse is generated, an internal counter resets and that output pulse is never generated.

1.47 The Programmable Sync Delay Block *(sync_delay_prog)*

Block Author: Aaron Parsons
Document Author: Aaron Parsons

Summary

Delay an infrequent boolean pulse by a run-time programmable number of enabled clocks. If the input pulse repeats before the output pulse is generated, an internal counter resets and that output pulse is never generated. When delay is changed, some randomly determined samples will be inserted/dropped from the buffered stream.

Mask Parameters

Parameter	Variable	Description
Max Delay ($2^?$):	*MaxDelay*	The maximum length of the delay.

Ports

Port	Dir.	Data Type	Description
sync	in	???	The boolean signal to be delayed.
delay	in	???	The run-time programmable delay length.
sync_out	out	???	The delayed boolean signal.

Description

Delay an infrequent boolean pulse by a run-time programmable number of enabled clocks. If the input pulse repeats before the output pulse is generated, an internal counter resets and that output pulse is never generated. When delay is changed, some randomly determined samples will be inserted/dropped from the buffered stream.

1.48 Windowed X-Engine *(win_x_engine)*

Block Author: Jason Manley, Aaron Parsons, Terry Filiba
Document Author: Jason Manley

Summary

CASPER X engine with added internal valid data masking functionality. Based on Aaron Parsons' design.

Mask Parameters

Parameter	Variable	Description
Number of antennas	*n_ants*	Number of antennas to process.
Bit width of samples in	*n_bits*	Bit width of each input sample number. Usually set to 4, resulting in 16 bit input numbers (2 polarizations, complex numbers).
Accumulation length	*acc_len*	Specified per antenna.
Adder latency	*add_latency*	Used to set the latency of internal adders.
Multiplier latency	*mult_latency*	Used to set the latency of internal multipliers.
BRAM latency	*bram_latency*	Used to set the latency of internal BRAMs.
Implementation: Multiplier type	*use_ded_mult*	Select the type of multipliers to use. Can be a single number or array - see below.
Implementation: Delay type	*use_bram_delay*	Selects the type of delays to implement. Single number configures all internal taps.

Ports

Port	Dir.	Data Type	Description
ant	in	variable width. see below.	Input port for incoming antenna data.
sync_in	in	boolean	Synchronization pulse. New window begins clock cycle after sync received.
window_valid	in	boolean	Indicates incoming antenna data is valid. Must remain constant for acc_len*n_ants.
acc	out	variable width. see below.	Output data.
valid	out	boolean	Indicates data on acc is valid.
sync_out	out	boolean	Passthrough for sync pulses.

Description

Introduction The CASPER X engine is a streaming architecture block where complex antenna data is input and accumulated products (for all cross-multiplications) are output in conjugated form. Because it is streaming with valid data expected on every clock cycle, data is logically divided into windows. These windows can either be valid (in which case the computation yields valid, outputted results) or invalid (in which case computation still occurs, but the results are ignored and not presented to the user).

Input format Data is input serially: *antenna A, antenna B, antenna C* etc. Each antenna's data consists of dual polarization, complex data. The bit width of each component number can be set as a parameter, *n_bits*. The X-engine thus expects these four numbers of *n_bits* to be concatenated into a single, unsigned number. CASPER convention dictates that complex numbers are represented with higher bits as real and lower bits as imaginary. The top half of the input number is polarization one and the lower half polarization two.

The internals of the block are reset with the reception of a sync pulse. A new window begins on the very next clock cycle. Each window is $int\_len \times n\_ants$ clock cycles long. The data for each antenna is input for *acc_len* clock cycles.

For example, for *n_bits* of 4 and *acc_len* of 2, the input to the X-engine would be 16 bits every clock cycle mapped as follows:

...	t_4	t_3	t_2	t_1	t_0	$\rightarrow$
...	C_{1real}	B_{1real}	B_{1real}	A_{1real}	A_{1real}	$most\_sig4b \rightarrow$
...	C_{1imag}	B_{1imag}	B_{1imag}	A_{1imag}	A_{1imag}	$4b \rightarrow$
...	C_{2real}	B_{2real}	B_{2real}	A_{2real}	A_{2real}	$4b \rightarrow$
...	C_{2imag}	B_{2imag}	B_{2imag}	A_{2imag}	A_{2imag}	$least\_sig4b \rightarrow$

Table 1.2: X-engine input with *acc_len* of 2.

The *window_valid* line is expected to remain constant for the duration of each window. If it is high, the output is considered valid and captured into the output FIFO buffer. With the close of that window, the output will be presented to the user as valid data on every second clock pulse. If *window_valid* was held low, the data is ignored.

With the close of one window, anther begins directly afterwards. Data can thus be streamed in and out continuously, while a sync pulse will force the start of a new window.

Output Format The windowed X-engine will produce $num\_baselines = n\_ants \times \frac{n\_ants+1}{2}$ valid outputs. The unwindowed x engine produces $num\_baselines = n\_ants \times (\frac{n\_ants}{2} + 1)$ results. The extra valids are a result of the algorithm employed and are masked out by the internal *x_engine_mask*.

Generally, the output of the X-engine configured for N antennas can be mapped into a table with $\frac{n\_ants}{2}+1$ columns and N rows as follows:

1^{st}	0×0	$(0 \times N)$	$(0 \times (N-1))$	$(0 \times (N-2))$	...	$\rightarrow$
2^{nd}	1×1	0×1	$(1 \times N)$	$(1 \times (N-1))$	...	$\rightarrow$
3^{rd}	2×2	1×2	0×2	$(2 \times N)$	...	$\rightarrow$
4^{th}	3×3	2×3	1×3	0×3	...	$\rightarrow$
5^{th}	4×4	3×4	2×4	1×4	...	$\rightarrow$
6^{th}	5×5	4×5	3×5	2×5	...	$\rightarrow$
...	...	...	...	...	...	$\rightarrow$

Table 1.3: Each table entry represents a valid output. Data is read out right to left, top to bottom. Bracketed values are from previous window.

As an example, consider the output for a 4 antenna system (with antennas numbered A through D):

1st	**AA**	prev win DA	prev win CA
2nd	**BB**	**AB**	prev win BD
3rd	**CC**	**BC**	**AC**
4th	**DD**	**CD**	**BD**
5th	next win AA	**DA**	CA
6th	next win BB	next win AB	DB

Table 1.4: Boldfaced type represents current valid window of data. Data is read out right to left, top to bottom. Non-boldfaced data is masked.

Thanks to the inclusion of the *x_engine_mask* block, X-engine output duplicates (observed in rows 5 and 6 of Table 1.4) are automatically removed. The output of a 4 antenna windowed X-engine is thus *AA, AB, BB, AC, BC, CC, BD, CD, DD, DA*.

1.49 X-Engine TVG *(xeng_tvg)*

Block Author: Jason Manley
Document Author: Jason Manley

Summary

Basic test vector generator for CASPER X-engines.

Mask Parameters

Parameter	Variable	Description
Number of Antennas (2^n)	*ant_bits*	Bitwidth of the number of antennas in the system.
Bitwidth of Samples in	*bits_in*	Bitwidth of component of the input.
X integration length (2^n)	*x_int_bits*	Bitwidth of X-engine accumulation length.
Sync Pulse Period (2^n)	*sync_period*	Bitwidth of number of valids between sync pulses.

Ports

Port	Dir.	Data Type	Description
tvg_sel	in	ufix_2_0	TVG selection. 0=off (passthrough), 1-3=TVG select.
data_in	in	inherited: bits_in*4	Data in for passthrough.
valid_in	in	boolean	Valid in made available for passthrough.
sync_in	in	boolean	Sync in made available for passthrough.
data_out	out	inherited: bits_in*4	Port Description
sync_out	out	boolean	Port Description
valid_out	out	boolean	Port Description

Description

This block generates data in a format suitable for input to a CASPER X-engine. The *tvg_sel* line selects the TVG. If set to zero, it is configured for passthrough and all input signals are propagated to the output (TVG is off). Values one through three select a TVG pattern. In this case, sync pulses are generated internally and valid data is output all the time. The three patterns are as follows:

1. Inserts a counter representing the antenna number. All real values count up from zero and imaginary values counting down from zero. ie antenna four would have the value *4 - 4i* inserted.

2. Inserts the same constant for all antennas: $Pol_{1real} = 0.125$, $Pol_{1imag} = -0.75$, $Pol_{2real} = 0.5$ and $Pol_{2imag} = -0.25$

3. User selectable values for each antenna. Input registers named *tv0* through *tv7* are input cyclically. Each value is input for *x_int_bits* clocks.

Chapter 2

Communication Blocks

2.1 10GbE Transceiver *(ten_GbE)*

Block Author: Pierre Yves Droz
Document Author: Jason Manley

Summary

This block sends and receives UDP frames (packets). It accepts a 64 bit wide data stream with user-determined frame breaks. The data stream is wrapped in a UDP frame for transmission. Incoming UDP packets are unwrapped and the data presented as a 64 bit wide stream. Only tested for the BEE2 platform.

Mask Parameters

Parameter	**Variable**	**Description**
Port	*port*	Selects the physical CX4 port on the iBOB or BEE2. The iBOB has two ports; the BEE2 has two for the control FPGA and four for each of the user FPGAs. CORR is not used by CASPER.
Use lightweight MAC	*mac_lite*	Toggles the use of a lightweight MAC implementation, which does not perform checksum validation.
Pre-emphasis	*pre_emph*	Selects the pre-emaphasis to use over the physical link. Default: 3 (see Xilinx documentation)
Differential Swing	*swing*	Selects the size of the differential swing to use in mV. Default: 800 (see Xilinx documentation)

Ports

Port	Dir.	Data Type	Description
rst	in	boolean	Resets the transceiver when pulsed high
tx_data	in	UFix_64_0	Accepts the data stream to be transmitted
tx_valid	in	boolean	The core accept the data on *tx_data* into the buffer while this line is high
tx_dest_ip	in	UFix_32_0	Selects the IP address of the receiving device
tx_dest_port	in	UFix_16_0	Selects the listening port of the receiving device (UDP port)
tx_end_of_frame	in	boolean	Signals the transceiver to begin transmitting the buffered frame (ie signals end of the frame)
tx_discard	in	boolean	Dumps the buffered packet and empties the FIFO buffer
rx_ack	in	boolean	Used to acknowledge reception of the data currently on rx_data and signals the transceiver to produce the next 64 bits from the receiver FIFO.
led_up	out	boolean	Indicates a link on the port
led_rx	out	boolean	Represents received traffic on the port
led_tx	out	boolean	Represents transmitted traffic on the port
tx_ack	out	boolean	Indicates that the data just clocked-in was accepted (will not acknowledge when buffer is full).
rx_data	out	UFix_64_0	Outputs the received data stream.
rx_valid	out	boolean	Indicates that the data on rx_data is valid (indicates a packet, or partial packet is in the RX buffer).
rx_source_ip	out	UFix_32_0	Represents the IP address of the sender of the current packet.
rx_source_port	out	UFix_16_0	Represents the sender's UDP port of the current packet.
rx_end_of_frame	out	boolean	Goes high to indicate the end of the received frame.
rx_size	out	UFix_16_0	Represents the total size of the packet currently in the RX buffer

Description

This document is a draft and requires verification.

Configuration The transceiver is configured through BORPH. Each transceiver instance has an entry in BORPH's */proc* filesystem. A simple way to modify configuration is to generate a text file and copy the contents into the */proc* entry. The text file's contents should be as follows:

```
begin
    mac = 10:10:10:10:10:10
    ip = 10.0.0.220
    gateway = 10.0.0.1
    port = 50000
end
```

Then use *cp /home/user/setup_GbE.txt /proc/PID/hw/ioreg/ten_GbE* to copy the file over the previous configuration. Please note that *ioreg_mode* must be in mode *1* for this to work.

Transmitting To transmit, data is clocked into a TX buffer through *tx_data* in 64 bit wide words using *tx_valid*. When ready to transmit, pulse the *tx_end_of_frame* line; the transceiver will add a UDP wrapper addressed to *tx_dest_ip:tx_dest_port* and begin transmission immediately. The *tx_end_of_frame* line must be brought high simultaneously with the last valid data word to be transmitted. Ie the *tx_valid* and *tx_end_of_frame* lines must be pulsed together to effect an end-of-frame.

If you do not wish to send the packet (and discard the data already clocked in), pulse *tx_discard* instead of *tx_end_of_frame*. The *tx_dest_ip* and *tx_dest_port* lines are ignored until a valid *tx_end_of_frame* is received. The sending port field in the UDP packet contains the listen port address (see below for configuration). Bear in mind that if the board is running at much over 120MHz, you cannot clock data into the core on every clock cycle (maximum transmission rate is 10Gbps and there is additional UDP packetization and ARP overhead). Maximum packet size appears to be in the order of 1100 words (of 64 bits each).

Receiving Upon receipt of a packet, *rx_valid* will go high and *rx_size* will indicate the length of the packet in 64 bit words. The received data is presented on *rx_data* in 64 bit wide words. You acknowledge receipt of this data using *rx_ack*, at which point the next data word will be presented. When the end of the packet is reached, *rx_end_of_frame* will go high.

Addressing To transmit, the IPv4 address is represented as a 32 bit binary number (whereas it's usually represented as four 8 bit decimal numbers). For example, if you wanted to send all packets to 192.168.1.1, a constant of 3 232 235 777 could be entered ($192 \times 2^{24} + 168 \times 2^{16} + 1 \times 2^8 + 1$) as the IP address. The port is represented by a 16 bit number, allowing full addressing of the UDP port range. Ports below 1024 are generally reserved for Linux kernel and Internet functions. Ports 1024 - 49151 are registered for specific applications and may not be used without IANA registration. To ensure inter-operability and compatibility, we recommend using dynamic (private) ports 49152 through 65535.

To receive, the MAC address, IP address and listen port of each transceiver can be configured through BORPH or TinySH. Transceivers may have different IP addresses and listen ports, however, it is only possible for any given transceiver to listen on one port at a time. This can be reconfigured while running.

LED Outputs The LED lines indicate port activity and can be connected to external GPIO LED interfaces. Bear in mind that even if no packets are being transmitted or received through the Simulink interface block, miscellaneous configuration packets are still sent and may be received by the microprocessor core. This activity will also be reflected on the activity LEDs.

Operation Apart from configuring the block, the processor is also used to map the routing tables. ARP requests and responses are handled by the microprocessor. All packets to the block's IP address that are not on the configured port are redirected to the processor running TinySH for management.

2.2 XAUI Transceiver *(XAUI)*

Block Author: Pierre Yves Droz, Henry Chen
Document Author: Jason Manley

Summary

XAUI block for sending and receiving point-to-point, streaming data over the BEE2 and iBOB's CX4 connectors. NOTE: A new version of this block is in development.

Mask Parameters

Parameter	**Variable**	**Description**
Demux	*demux*	Selects the width of the data bus. 1 for 64 bits, 2 for 32 bits.
Port	*port*	Selects the physical CX4 port on the iBOB or BEE2. The iBOB has two ports; the BEE2 has two for the control FPGA and four for each of the user FPGAs. CORR is not used by CASPER.
Pre-emphasis	*pre_emph*	Selects the pre-emaphasis to use over the physical link. Default: 3 (see Xilinx documentation)
Differential Swing	*swing*	Selects the size of the differential swing to use in mV. Default: 800 (see Xilinx documentation)

Ports

Port	**Dir.**	**Data Type**	**Description**
rx_get	in	boolean	Used to request the next data word from the RX buffer.
rx_reset	in	boolean	Resets the receive subsystem.
tx_data	in	ufix_64_0 or ufix_32_0	Accepts the next data word (64 or 32 bits) to be transmitted.
tx_outofband	in	ufix_8_0 or ufix_4_0	Accepts the next data word (8 bits if demux=1, 4 bits if demux=2) to be transmitted through the out-of-band channel.
tx_valid	out	boolean	Clocks the transmit data into the transceiver. Data is clocked into the buffer while this line is high.
rx_data	out	ufix_64_0	Outputs the received data stream.
rx_outofband	out	ufix_8_0 or ufix_4_0	Outputs the out-of-band received data stream.
rx_empty	out	boolean	Indicates that the receive buffer is empty.
rx_valid	out	boolean	Indicates that data has been received.
rx_linkdown	out	boolean	Indicates that the link is down (eg. faulty cable).
tx_full	out	boolean	Indicates the transmit buffer is full.
rx_almost_full	boolean	inherited	Indicates the receive buffer is full.

Description

This block is due to be deprecated soon. It will be replaced by a new XAUI block in the CASPER library.

Demux Perhaps a misnomer, this parameter describes the width of the data bus rather than a selection of two muxed streams on one channel. At 156MHz XAUI clock, the maximum transmission speed is 64bits * 156.25 MHz = 10Gbit/s. For BEE or iBOB designs clocked at rates above 156MHz, clocking-in 64 bit data on every clock cycle would cause the XAUI block's FIFO buffers to overflow. The *demux* option is provided which halves the input data bus width to 32 bits and enables data to be clocked-in on every FPGA clock cycle. Along with the data bus width, the *out of band* bus width is also halved to 4 bits.

Out of band signals Out of band signals are guaranteed to arrive at the same time as the data word with which they were sent. Out-of-band data is only transmitted across the physical link if the input to *tx_outofband* changes and is clocked in as valid (*tx_valid*). In other words, if you keep *tx_outofband* constant, no additional bandwidth is consumed (the in-band signals are transmitted as normal). When data is clocked into the transmitter, it will appear out the receiver as if the *tx_outofband* and *tx_data* arrived simultaneously. Care should be taken to ensure that the data clocked into *tx_outofband* and *tx_data* does not exceed the XAUI's maximum transmission rate (approximately 10Gbps for 156.25MHz clock). Each change of *tx_outofband* (be it one bit or eight bits) requires 64 bits (a full word) to transmit. This bus width is 8 bits if *demux* is not selected (set to 1), and 4 bits if it is set to 2.

Chapter 3

System Blocks

3.1 ADC *(adc)*

Block Author: Pierre Yves Droz
Document Author: Ben Blackman

Summary

The ADC block converts analog inputs to digital outputs. Every clock cycle, the inputs are sampled and digitized to 8 bit binary point numbers in the range of [-1, 1) and are then output by the adc.

Mask Parameters

Parameter	**Variable**	**Description**
ADC board	*adc_brd*	Select which ADC port to use on the IBOB.
ADC clock rate (MHz)	*adc_clk_rate*	Sets the clock rate of the ADC, must be at least 4x the IBOB clock rate.
ADC interleave mode	*adc_interleave*	Check for 1 input, uncheck for 2 inputs.
Sample period	*sample_period*	Sets the period at which the adc outputs samples (ie 2 means every other cycle).

Ports

Port	Dir.	Data Type	Description
sim_in	in	double	The analog signal to be digitized if interleave mode is selected. Note: For simulation only.
sim_i	in	double	The first analog signal to be digitized if interleave mode is unselected. Note: For simulation only.
sim_q	in	double	The second analog signal to be digitized if interleave mode is unselected. Note: For simulation only.
sim_sync	in	double	Takes a pulse to be observed at the output to measure the delay through the block. Note: For simulation only.
sim_data_valid	in	double	A signal that is high when inputs are valid. Note: For simulation only.
oX	out	Fix_8_7	A signal that represents sample X+1 (Ex. o0 is the 1st sample, o7 is the 8th sample). Used if interleave mode is on.
iX	out	Fix_8_7	A signal that represents sample X+1 (Ex. i0 is the 1st sample, o3 is the 4th sample). Used if interleave mode is off.
qX	out	Fix_8_7	A signal that represents sample X+1 (Ex. q0 is the 1st sample, q3 is the 4th sample). Used if interleave mode is off.
outofrangeX	out	boolean	A signal that represents when samples are outside the valid range.
syncX	out	boolean	A signal that is high when the sync pulse offset by X if interleave mode is unselected, or 2X if interleave mode is selected is high (Ex. sync2 is the pulse offset by 2 if interleave is off or offset by 4 if interlave is on).
data_valid	out	boolean	A signal that is high when the outputs are valid.

Description

Usage The ADC block can take 1 or 2 analog input streams. The first input should be connected to input i and the second to input q if it is being used. The inputs will then be digitized to *Fix_8_7* numbers between [-1, 1). For a single input, the *adc* samples its input 8 times per IBOB clock cycle and outputs the 8 samples in parallel with o0 being the first sample and o7 the last sample. For 2 inputs, the *adc* samples both inputs 4 times per IBOB clock cycle and then outputs them in parallel with i0-i3 corresponding to input i and q0-q3 corresponding to input q. In addition to having 2 possible inputs, each IBOB can interface with 2 *adc*s for a total of 4 inputs or 2 8-sample inputs per IBOB.

Connecting the Hardware To hook up the ADC board, attach the clock SMA cable to the clk_i port, the first input to the I+ port, and the second input to the Q+ port. Check the hardware on the ADC board near the input pins. There should be for 4 square chips in a straight line. If there are only 3, the second input, Q+, may not work. Note that if you chose *adc0_clk*, make sure to plug the ADC board in to the adc0 port. The same applies if you chose *adc1_clk* to plug the board into adc1 port. If you are using both ADCs, then you need to plug a clock into both clk_i inputs and you should probably run them off of the same signal generator.

ADC Background Information The ADC board was designed to mate directly to an IBOB board through ZDOK connectors for high-speed serial data I/O. Analog data is digitized using an Atmel AT84AD001B dual 8-bit ADC chip which can digitize two streams at 1 Gsample/sec or a single stream at 2 Gsample/sec. This board may be driven with either single-ended or differential inputs.

3.2 DAC *(dac)*

Block Author: Henry Chen
Document Author: Ben Blackman

Summary

The DAC block converts 4 digital inputs to 1 analog output. The *dac* runs at 4x FPGA clock frequency, outputting analog converted samples 0 through 3 each FPGA clock cycle.

Mask Parameters

Parameter	Variable	Description
DAC board	*dac_brd*	Select which IBOB port to run this *dac*.
DAC clock rate (MHz)	*dac_clk_rate*	The clock rate to run the *dac*. Must be 4x FPGA clock rate.
Sample period	*sample_period*	Sets the period at which the *dac* outputs samples (ie 2 means every other cycle).
Show Implementation Parameters	*show_param*	Allows the user to set the implementation parameters.
Invert output clock phase	*invert_clock*	When unchecked, the *dac* samples the data aligned with the clock. When checked, the *dac* samples the data aligned with an inverted clock.

Ports

Port	Dir.	Data Type	Description
dataX	in	Fix_9_8	One of 4 digital inputs to be converted to analog.
sim_out	out	double	Analog output of *dac*. Note: For simulation only.

Description

Usage The *dac* takes 4 *Fix_9_8* inputs and outputs an analog stream. The *dac* runs at 4x the FPGA clock speed.

To be updated.

3.3 DRAM *(dram)*

Block Author: Pierre Yves Droz
Document Author: Jason Manley

Summary

This block interfaces to the BEE2's 1GB DDR2 ECC DRAM modules. Commands that are clocked-in are executed with an unknown delay, however, execution order is maintained.

Mask Parameters

Parameter	**Variable**	**Description**
DIMM	*dimm*	Selects which physical DIMM to use (four per user FPGA).
Data Type	*arith_type*	Inform Simulink how it should interpret the stored data.
Data binary point	*bin_pt*	Inform Simulink how it should interpret the stored data - specifically, the bit position in the word where it should place the binary point.
Datapath clock rate (MHz)	*ip_clock*	Clock rate for DRAM. Default: 200MHz (400DDR).
Sample period	*sample_period*	Is significant for clocking the block. Default: 1
Simulate DRAM using ModelSim	*use_sim*	Requires the addition of the "ModelSim" block at the top level of the design. Used to simulate DRAM block only.
Enable bank management	*bank_mgt*	*Advise leave off* Changes the way the banks are addressed. Clarification required.
Use wide data bus (288 bits)	*wide_data*	Burst writes require 288 bits. If not selected, provide a 144 bit bus which needs to be supplied with data in consecutive clock cycles to form the 288 bits. 288 bit bus can make for challenging routing!
Use half-burst	*half_burst*	Only store 144 bits per burst (wastes half capacity as the second 144 bits are unusable). If enabled, requires at least two clock cycles to store 144 bits. Second clock cycle's data is forfeited.

Ports

Port	Dir.	Data Type	Description
rst	in	boolean	Resets the block when pulsed high
address	in	UFix_32_0	A signal which accepts the address. See below for details.
data_in	in	144 or 288 bit unsigned	Accepts data to be saved to DRAM.
wr_be	in	UFix_18_0 or UFix_36_0	Selects bytes for writing (write byte enable). It is normally 18 bits wide for a 144 bit data bus, but if 288 bit data bus is selected, this becomes a 36 bit variable.
RWn	in	boolean	Selects read or not-write. *1* for read, *0* for write.
cmd_tag	in	UFix_32_0	Accepts a user-defined tag for labelling entered commands.
cmd_valid	in	boolean	Clocks data into the command buffer.
rd_ack	out	boolean	Used to acknowledge that the last *data_out* value has been read.
cmd_ack	out	boolean	Acknowledges that the last command was accepted (when buffer is full, will not accept additional commands).
data_out	out	UFix_144_0	Outputs data from DRAM, 144 bits at a time. Reads are in groups of 288 bits (ie 2 clocks).
rd_tag	out	UFix_32_0	Outputs the identifier for the data on *data_out* (as submitted on *cmd_tag* when the command was issued).
rd_valid	out	boolean	Indicates that the data on *data_out* is valid.

Description

This document is a draft and requires verification.

Addressing The 1GB storage DIMMs have 18 512Mbit chips each. They are arranged as 64Mbit x 8 (bus width) x 9 (chips per side/rank) x 2 (sides/ranks). Two ranks (sides) per module with the 9 memory ICs connected in parallel, each holding 8 bits of the data bus width (72 bits). Each IC has four banks, with 13 bits of row addressing and 10 bits for column addressing. Normally, each address would hold 64 bits + parity (8 bits), however, the BEE2 uses the parity space as additional data storage giving a capacity of 1.125 GB per DIMM module.

From Micron's datasheet on the *MT47H64M8CD-37E* (as used by CASPER in its Crucial 1GB *CT12872AA53E* modules): The double data rate architecture is essentially a 4n-prefetch architecture, with an interface designed to transfer two data words per clock cycle at the I/O balls. A single read or write access effectively consists of a single 4n-bit-wide, one-clock-cycle data transfer at the internal DRAM core and four corresponding n-bit-wide, one-half-clock-cycle data transfers at the I/O balls.

Reads and writes must thus occur four-at-a-time. 4 x 72bits = 288 bits. Although the mapping of the logical to physical addressing is abstracted from the user, it is useful to know how the DRAM block's address bus is derived, as it impacts performance:

Each group of 8 addresses selects a 144 bit logical location (the lowest 3 bits are ignored). For example, address *0x00* through *0x7* all address the same 144 bit location. To address consecutive locations, increment the address port by eight. There are thus a total of 2^{27} possible addresses. The block supports 2GB DIMMs (UNCONFIRMED) since 14 bits of addressing are reserved for row selection. The 1GB DIMMs using Micron 512Mb chips, however, only use 13 bits for row selection which results in 2^{26} possible address locations. Care

Addressing	Assignment
Column	12 downto 3
Rank	13
Row	27 downto 14
Bank	29 downto 28
not used	31 downto 30

Table 3.1: Address bit assignments

should be taken when addressing the 1GB DIMMS as bit 27 of the address range is not valid. However, bits 28 and 29 are mapped. Since bit 27 is ignored, it results in overlapping memory spaces.

Data bus width The BEE2 uses ECC DRAM, however, the parity bits are used for data storage rather than parity storage. Thus, the data bus is 72 bits wide instead of the usual 64 bits.

The memory module has a DDR interface requiring two reads or writes per RAM clock cycle (200MHz), thus requiring the user to provide 144 bits per clock cycle. Furthermore, as outlined above, data has to be captured in batches of 288 bits. This can be done in one of two ways: in two consecutive blocks of 144 bits, or over a single 288 bit-wide bus. This is selectable as a Mask parameter. If half-burst is selected, only a 144 bit input is required. 288 bits are still written to DRAM, but the second 144 bits are not specified. Thus, half of the DRAM capacity is unusable.

Performance Issues The performance of the DRAM block is dependant on the relative location of the addressed data and whether or not the mode (read/write) is changed. For example, consecutive column addresses can be written without delay, however, changing rows or banks incur delay penalties. See above for the address bit assignment.

To obtain optimum performance, it is recommended that the least significant bits be changed first (ie address the memory from 0x00 through to address 0x20000000). This will increment column addresses first, followed by rank change, both of which incur little delay. Changing rows or banks can take twice as long. Further information can be found in Micron's datasheet for the *MT47H64M8*.

3.4 Snapshot Capture *(snap)*

Block Author: Aaron Parsons
Document Author: Aaron Parsons, Ben Blackman

Summary

The snap block provides a packaged solution to capturing data from the FPGA fabric and making it accessible from the CPU. snap captures to a 32 bit wide shared BRAM.

Mask Parameters

Parameter	Variable	Description
No. of Samples ($2^?$)	*nsamples*	Specifies the depth of the Shared BRAM(s); i.e. the number of 32bit samples which are stored per capture.

Ports

Port	Dir.	Data Type	Description
din	in	unsigned_32_0	The data to be captured. Regardless of type, the bit-level representation of these numbers are written as 32bit values to the Shared BRAM.
trig	in	boolean	When high, triggers the beginning of a data capture. Thereafter, every enabled data is written to the shared BRAM until it is full.
we	in	boolean	After a trigger is begun, enables a write to Shared BRAM.

Description

Usage Under TinySH/BORPH, this device will have 3 sub-devices: *ctrl*, *bram*, and *addr*. *ctrl* is an input register. Bit 0, when driven from low to high, enables a trigger/data capture to occur. Bit 1, when high, overrides *trig* to trigger instantly. Bit 2, when high, overrides *we* to always write data to bram. *addr* is an output register and records the last address of *bram* to which data was written. *bram* is a 32 bit wide Shared BRAM of the depth specified in *Parameters*.

3.5 64 Bit Snapshot *(snap64)*

Block Author: Aaron Parsons
Document Author: Aaron Parsons, Ben Blackman

Summary

The snap block provides a packaged solution to capturing data from the FPGA fabric and making it accessible from the CPU. snap64 captures to 2x32 bit wide shared BRAMs to effect a 64 bit capture.

Mask Parameters

Parameter	**Variable**	**Description**
No. of Samples ($2^?$)	*nsamples*	Specifies the depth of the Shared BRAM(s); i.e. the number of 64bit samples which are stored per capture.

Ports

Port	**Dir.**	**Data Type**	**Description**
din	in	unsigned_64_0	The data to be captured. Regardless of type, the bit-level representation of these numbers are written as 64bit values to the Shared BRAMs.
trig	in	boolean	When high, triggers the beginning of a data capture. Thereafter, every enabled data is written to the shared BRAM until it is full.
we	in	boolean	After a trigger is begun, enables a write to Shared BRAM.

Description

Usage Under TinySH/BORPH, this device will have 3 sub-devices: *ctrl*, *bram_msb*, *bram_lsb*, and *addr*. *ctrl* is an input register. Bit 0, when driven from low to high, enables a trigger/data capture to occur. Bit 1, when high, overrides *trig* to trigger instantly. Bit 2, when high, overrides *we* to always write data to bram. *addr* is an output register and records the last address of bram to which data was written. *bram_msb* and *bram_lsb* are 32 bit wide Shared BRAMs of the depth specified in *Parameters*. *bram_msb* holds the upper 32 bits of *din* while *bram_lsb* holds the lower 32 bits of *din*.

3.6 Software Register *(software register)*

Block Author: Pierre-Yves Droz
Document Author: Henry Chen

Summary

Inserts a unidirectional 32-bit register shared between the FPGA design and the PowerPC bus.

Mask Parameters

Parameter	Variable	Description
I/O direction	*io_dir*	Chooses whether register writes *to processor* or reads *from processor*.
Data Type	*arith_type*	Specifies data type of register.
Data bitwidth	*bitwidth*	Specifies data bitwidth. Hard-coded at 32 bits.
Data binary point	*bin_pt*	Specifies the binary point position of data.
Sample period	*sample_period*	Specifies sample period of interface.

Ports

Port	Dir.	Data Type	Description
reg_out	in	inherited	Output from design to processor bus. Only in *To Processor* mode.
sim_out	out	double	Simulation output of register value. Only in *To Processor* mode.
sim_in	in	double	Simulation input of register value. Only in *From Processor* mode.
reg_in	out	inherited	Input from processor bus to design. Only in *From Processor* mode.

Description

A software register is a *shared* interface, meaning that it is attached to both the FPGA fabric of the System Generator design as well as the PowerPC bus. The registers are unidirectional; the user must choose at design-time whether the register is in *To Processor* mode (written by the FPGA fabric and read by the PowerPC) or in *From Processor* mode (written by the PowerPC and read by the FPGA fabric).

The bitwidth is fixed at 32 bits, as it is attached to a 32-bit bus, but the Simulink interpretation of the data type and binary point is controllable by the user. The data type and binary point parameters entered into the mask are enforced by the block; the block will cast to the specified data type and binary point going in both directions.

3.7 SRAM *(sram)*

Block Author: Pierre Yves Droz, Henry Chen
Document Author: Ben Blackman

Summary

The sram block represents a 36x512k SRAM chip on the IBOB. It stores 36-bit words and requires 19 bits to access its address space.

Mask Parameters

Parameter	**Variable**	**Description**
SRAM	*sram*	Selects which SRAM chip this block represents.
Data Type	*arith_type*	Type to which the data is cast on both the input and output.
Data binary point (bitwidth is 36)	*bin_pt*	Position of the binary point of the data.
Sample period	*sample_period*	Sets the period with reference to the clock frequency.
Simulate SRAM using ModelSim	*use_sim*	Turns ModelSim simulation on or off.

Ports

Port	**Dir.**	**Data Type**	**Description**
we	in	boolean	A signal that when high, causes the data on data_in to be written to address.
be	in	unsigned_4_0	A signal that enables different 9-bit bytes of data_in to be written.
address	in	unsigned_19_0	A signal that specifies the address where either data_in is to be stored or from where data_out is to be read.
data_in	in	arith_type_36	A signal that contains the data to be stored.
data_out	out	arith_type_36	A signal that contains the data coming out of address.
data_valid	out	boolean	A signal that is high when data_out is valid.

Description

Usage The SRAM block is 36x512k, signifying that its input and output are 36-bit words and it can store 512k words. Each clock cyle, if *we* is high, then each bit of be determines whether each 9-bit chunk will be written to address. *be* is 4 bits with the highest bit corresponding to the most significant chunk (so if *be* is 1100, only the top 18 bits will be written). If *we* is low, then the SRAM block ignores *data_in* and be and reads the word stored at *address*.

3.8 XSG Core Config *(XSG core config)*

Block Author: Pierre-Yves Droz
Document Author: Henry Chen

Summary

The XSG Core Config block is used to configure the System Generator design for the *bee_xps* toolflow. Settings here are used to configure the Xilinx System Generator block parameters automatically, and control toolflow script execution. It needs to be at the top level of all designs being compiled with the *bee_xps* toolflow.

Mask Parameters

Parameter	**Variable**	**Description**
Hardware Platform	*hw_sys*	Selects the board/chip to compile for.
Include Linux add-on board support	*ibob_linux*	Includes BORPH-capable Linux for IBOB.
User IP Clock source	*clk_src*	Selects the clock on which to run the System Generator circuit.
GPIO Clock Pin I/O group	*gpio_clk_io_group*	Selects GPIO type to use as clock input if using user clock on an IBOB.
GPIO Clock Pin bit index	*gpio_clk_bit_index*	Selects GPIO pin to use as clock input if using user clock on an IBOB.
User IP Clock rate (MHz)	*clk_rate*	Generates timing constraints for the design.
Sample Period	*sample_period*	Sample period for Simulink simulations.
Synthesis Tool	*synthesis_tool*	Selects the tool to use for synthesizing the design's netlist.

Description

The function of the XSG Core Config block is to set parameters for the toolflow scripts. It supercedes the use of the Xilinx System Generator block and has supplemental options for board-level parameters. Although a System Generator block is still needed in all designs, the XSG Core Config block automatically changes the System Generator block settings based on its own parameters.

The settings in the XSG Core Config block are used to determine the system-level conditions of the SysGen design. It sets which of the toolflow-supported boards the design is being compiled for, from which it determines what FPGA to target, as well as clocking options like clock source and timing constraints. The Sample Period and Synthesis Tool parameters are included in the block so that all system-level options available in the System Generator block could be handled by this single block.