

CASPER Library
Reference Manual

Last Updated June 23, 2007

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.	<i>n_inputs</i>	The number of inputs to be summed.
Add Latency	<i>latency</i>	The latency of each stage through the adder tree.

Ports

Port	Direction	Data Type	Description
<i>sync</i>	IN	Boolean	Indicates the next clock cycle containing valid data
<i>din</i>	IN	Inherited	A number to be summed.

Description

Sums all inputs using a tree of adds and delays. Total latency is $\text{ceil}(\log_2(n_inputs)) * \text{latency}$

1.2 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.	<i>n_bits</i>	Specifies the width of the input.

Ports

Port	Direction	Data Type	Description
<i>in</i>	IN	UFix_x_0	The input signal.
<i>out</i>	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.

1.3 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*, 1.24)

Mask Parameters

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

Ports

Port	Direction	Data Type	Description
<i>c</i>	IN	UFix_x_0	Complex input, real in MSB, imaginary in LSB.
<i>r</i>	OUT	Fix_x_y	Real signed output, binary point specified by parameter.
<i>i</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.

1.4 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	<i>mult_latency</i>	The latency through a multiplier.
Add Latency	<i>add_latency</i>	The latency through an adder.

Ports

Port	Direction	Data Type	Description
<i>a</i>	IN	Inherited	The real component of input 1.
<i>b</i>	IN	Inherited	The imaginary component of input 1.
<i>c</i>	IN	Inherited	The real component of input 2.
<i>d</i>	IN	Inherited	The imaginary component of input 2.
<i>real</i>	OUT	Inherited	ac-bd
<i>imag</i>	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.

1.5 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	<i>mult_latency</i>	The latency through a multiplier.
Add Latency	<i>add_latency</i>	The latency through an adder.

Ports

Port	Direction	Data Type	Description
<i>a</i>	IN	Inherited	The real component of input 1.
<i>b</i>	IN	Inherited	The imaginary component of input 1.
<i>c</i>	IN	Inherited	The real component of input 2.
<i>d</i>	IN	Inherited	The imaginary component of input 2.
<i>real</i>	OUT	Inherited	$ac+bd$
<i>imag</i>	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.

1.6 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	<i>mult_latency</i>	The latency through a multiplier.
Add Latency	<i>add_latency</i>	The latency through an adder.

Ports

Port	Direction	Data Type	Description
<i>a</i>	IN	Inherited	The real component of input 1.
<i>b</i>	IN	Inherited	The imaginary component of input 1.
<i>c</i>	IN	Inherited	The real component of input 2.
<i>d</i>	IN	Inherited	The imaginary component of input 2.
<i>real</i>	OUT	Inherited	$ac+bd$
<i>imag</i>	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	<i>mult_latency</i>	The latency through a multiplier.
dd Latency	<i>add_latency</i>	The latency through an adder.

Ports

Port	Direction	Data Type	Description
<i>a</i>	IN	Inherited	The real component of input 1.
<i>b</i>	IN	Inherited	The imaginary component of input 1.
<i>c</i>	IN	Inherited	The real component of input 2.
<i>d</i>	IN	Inherited	The imaginary component of input 2.
<i>real</i>	OUT	Inherited	$ac-bd$
<i>imag</i>	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	<i>mult_latency</i>	The latency through a multiplier.
dd Latency	<i>add_latency</i>	The latency through an adder.

Ports

Port	Direction	Data Type	Description
<i>a</i>	IN	Inherited	The real component of input 1.
<i>b</i>	IN	Inherited	The imaginary component of input 1.
<i>c</i>	IN	Inherited	The real component of input 2.
<i>d</i>	IN	Inherited	The imaginary component of input 2.
<i>real</i>	OUT	Inherited	ac+bd
<i>imag</i>	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 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	<i>mult_latency</i>	The latency through a multiplier.
dd Latency	<i>add_latency</i>	The latency through an adder.

Ports

Port	Direction	Data Type	Description
<i>a</i>	IN	Inherited	The real component of input 1.
<i>b</i>	IN	Inherited	The imaginary component of input 1.
<i>c</i>	IN	Inherited	The real component of input 2.
<i>d</i>	IN	Inherited	The imaginary component of input 2.
<i>real</i>	OUT	Inherited	$ac+bd$
<i>imag</i>	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.

1.10 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	<i>mult_latency</i>	The latency through a multiplier.
Add Latency	<i>add_latency</i>	The latency through an adder.

Ports

Port	Direction	Data Type	Description
<i>a</i>	IN	Inherited	The real component of input 1.
<i>b</i>	IN	Inherited	The imaginary component of input 1.
<i>c</i>	IN	Inherited	The real component of input 2.
<i>d</i>	IN	Inherited	The imaginary component of input 2.
<i>real</i>	OUT	Inherited	ac-bd
<i>imag</i>	OUT	Inherited	ad+bc

Description

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

1.11 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	<i>mult_latency</i>	The latency through a multiplier.
Add Latency	<i>add_latency</i>	The latency through an adder.

Ports

Port	Direction	Data Type	Description
<i>a</i>	IN	Inherited	The real component of input 1.
<i>b</i>	IN	Inherited	The imaginary component of input 1.
<i>c</i>	IN	Inherited	The real component of input 2.
<i>d</i>	IN	Inherited	The imaginary component of input 2.
<i>real</i>	OUT	Inherited	ac+bd
<i>imag</i>	OUT	Inherited	-ad+bc

Description

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

1.12 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	<i>mult_latency</i>	The latency through a multiplier.
Add Latency	<i>add_latency</i>	The latency through an adder.

Ports

Port	Direction	Data Type	Description
<i>a</i>	IN	Inherited	The real component of input 1.
<i>b</i>	IN	Inherited	The imaginary component of input 1.
<i>c</i>	IN	Inherited	The real component of input 2.
<i>d</i>	IN	Inherited	The imaginary component of input 2.
<i>real</i>	OUT	Inherited	ac+bd
<i>imag</i>	OUT	Inherited	-ad+bc

Description

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

1.13 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)	<i>DelayLen</i>	The length of the delay.
BRAM Latency	<i>bram_latency</i>	The latency of the underlying storage BRAM.

Ports

Port	Direction	Data Type	Description
<i>in</i>	IN	???	The signal to be delayed.
<i>out</i>	OUT	???	The delayed signal.

Description

A delay block that uses BRAM for its storage.

1.14 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	<i>DelayLen</i>	The length of the delay.
Extra (unenabled) delay for BRAM Latency	<i>bram_latency</i>	The latency of the underlying storage BRAM.

Ports

Port	Direction	Data Type	Description
<i>in</i>	IN	???	The signal to be delayed.
<i>en</i>	IN	???	To be asserted when input is valid.
<i>out</i>	OUT	???	The delayed signal.
<i>valid</i>	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.

1.15 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 ($2^?$)	<i>MaxDelay</i>	The maximum length of the delay (i.e. the BRAM Size).
BRAM Latency	<i>bram_latency</i>	The latency of the underlying storage BRAM.

Ports

Port	Direction	Data Type	Description
<i>din</i>	IN	???	The signal to be delayed.
<i>delay</i>	IN	???	The run-time programmable delay length.
<i>dout</i>	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.

1.16 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	<i>delay_depth</i>	The length of the delay.
Bit Width	<i>n_bits</i>	Specifies the width of the real/imaginary components. Width of each component is assumed equal.
Use BRAM	<i>use_bram</i>	Set to 1 to implement the delay using BRAM. If 0, the delay will be implemented using SLR16 elements.

Ports

Port	Direction	Data Type	Description
<i>in</i>	IN	???	The complex signal to be delayed.
<i>out</i>	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.

1.17 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	<i>DelayLen</i>	The length of the delay.

Ports

Port	Direction	Data Type	Description
<i>in</i>	IN	???	The signal to be delayed.
<i>out</i>	OUT	???	The delayed signal.

Description

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

1.18 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	Direction	Data Type	Description
<i>in</i>	IN	Boolean	Input boolean signal.
<i>out</i>	OUT	Boolean	Edge detected output boolean signal.

Description

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

1.19 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^i)	<i>CounterBits</i>	Specifies the number of bits (and the final count output of 2^{bits-1}).

Ports

Port	Direction	Data Type	Description
<i>en</i>	IN	???	Step the counter by 1 unless $addr=2^{bits-1}$.
<i>rst</i>	IN	???	Reset counter to 0.
<i>addr</i>	OUT	???	Current output of the counter.
<i>we</i>	OUT	Boolean	Outputs boolean true just before <i>addr</i> is incremented.
<i>done</i>	OUT	Boolean	Outputs boolean true when a final <i>en</i> 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.

1.20 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	Direction	Data Type	Description
<i>in</i>	IN	Boolean	Input boolean signal.
<i>out</i>	OUT	Boolean	Negative-edge detected output boolean signal.

Description

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

1.21 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.	<i>n_inputs</i>	The number of parallel inputs.
Mux Latency	<i>latency</i>	The latency of each mux block.

Ports

Port	Direction	Data Type	Description
<i>sync</i>	???	???	Indicates the next clock cycle containing valid data
<i>din</i>	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

1.22 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	Direction	Data Type	Description
<i>in</i>	IN	Boolean	Input boolean signal.
<i>out</i>	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.

1.23 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	<i>pulse_len</i>	Specifies number of clocks after the last high input for which the output is held high.

Ports

Port	Direction	Data Type	Description
<i>in</i>	IN	Boolean	Input boolean signal.
<i>out</i>	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.

1.24 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, 1.3)

Mask Parameters

Parameter	Variable	Description
-----------	----------	-------------

Ports

Port	Direction	Data Type	Description
<i>r</i>	IN	Fix_x_y	Real data
<i>i</i>	IN	Fix_x_y	Imaginary signed output, binary point specified by parameter.
<i>c</i>	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, 1.3)

1.25 Display name for Block (*simulink_name_for_block*)

Block Author: Block Author

Document Author: Document Author

Summary

Block Summary can contain arbitrary L^AT_EX like lists

- list item 1
- list item 2

Mask Parameters

Parameter	Variable	Description
Parameter Name	<i>Variable Name</i>	Parameter Description
Parameter Name	<i>Variable Name</i>	Parameter Description

Ports

Port	Direction	Data Type	Description
<i>Port Name</i>	PORT DIRECTION	Port data type	Port Description
<i>Port Name</i>	IN	ufix_x_y	Port Description
<i>Port Name</i>	IN	inherited	Port Description

Description

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

1.26 Display name for Block (*simulink_name_for_block*)

Block Author: Block Author

Document Author: Document Author

Summary

Block Summary can contain arbitrary L^AT_EX like lists

- list item 1
- list item 2

Mask Parameters

Parameter	Variable	Description
Parameter Name	<i>Variable Name</i>	Parameter Description
Parameter Name	<i>Variable Name</i>	Parameter Description

Ports

Port	Direction	Data Type	Description
<i>Port Name</i>	PORT DIRECTION	Port data type	Port Description
<i>Port Name</i>	IN	ufix_x_y	Port Description
<i>Port Name</i>	IN	inherited	Port Description

Description

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

1.27 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	<i>DelayLen</i>	The length of the delay.

Ports

Port	Direction	Data Type	Description
<i>in</i>	IN	boolean	The boolean signal to be delayed.
<i>en</i>	IN	boolean	To be asserted when input is valid.
<i>out</i>	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.

1.28 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 [?]):	<i>MaxDelay</i>	The maximum length of the delay.

Ports

Port	Direction	Data Type	Description
<i>sync</i>	IN	???	The boolean signal to be delayed.
<i>delay</i>	IN	???	The run-time programmable delay length.
<i>sync_out</i>	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.

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.

Mask Parameters

Parameter	Variable	Description
Port	<i>port</i>	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	<i>mac_lite</i>	Toggles the use of a lightweight MAC implementation, which does not perform checksum validation.
Pre-emphasis	<i>pre_emph</i>	Selects the pre-emphasis to use over the physical link. Default: 3 (see Xilinx documentation)
Differential Swing	<i>swing</i>	Selects the size of the differential swing to use in mV. Default: 800 (see Xilinx documentation)

Ports

Port	Direction	Data Type	Description
<i>rst</i>	IN	boolean	Resets the transceiver when pulsed high
<i>tx_data</i>	IN	UFix_64_0	Accepts the data stream to be transmitted
<i>tx_valid</i>	IN	boolean	Must be pulsed high to accept the 64 bit data on <i>tx_data</i> into the buffer
<i>tx_dest_ip</i>	IN	UFix_32_0	Selects the IP address of the receiving device
<i>tx_dest_port</i>	IN	UFix_16_0	Selects the listening port of the receiving device (UDP port)
<i>tx_end_of_frame</i>	IN	boolean	Signals the transceiver to begin transmitting the buffered frame (ie signals end of the frame)
<i>tx_discard</i>	IN	boolean	Dumps the buffered packet and empties the FIFO buffer
<i>rx_ack</i>	IN	boolean	Used to acknowledge reception of the data currently on <i>rx_data</i> and signals the transceiver to produce the next 64 bits from the receiver FIFO.
<i>led_up</i>	OUT	boolean	Indicates a link on the port
<i>led_rx</i>	OUT	boolean	Represents received traffic on the port
<i>led_tx</i>	OUT	boolean	Represents transmitted traffic on the port
<i>tx_ack</i>	OUT	boolean	Indicates that the data just clocked-in was accepted (will not acknowledge when buffer is full).
<i>rx_data</i>	OUT	UFix_64_0	Outputs the received data stream.
<i>rx_valid</i>	OUT	boolean	Indicates that the data on <i>rx_data</i> is valid (indicates a packet, or partial packet is in the RX buffer).
<i>rx_source_ip</i>	OUT	UFix_32_0	Represents the IP address of the sender of the current packet.
<i>rx_source_port</i>	OUT	UFix_16_0	Represents the sender's UDP port of the current packet.
<i>rx_end_of_frame</i>	OUT	boolean	Extended Port Description wrapping up many many many many many many lines
<i>rx_size</i>	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. 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 `tx_end_of_frame` is pulsed high. The sending port field in the UDP packet contains the listen port address (see below for configuration).

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.

2.2 Display name for Block (*simulink_name_for_block*)

Block Author: Block Author

Document Author: Document Author

Summary

Block Summary can contain arbitrary L^AT_EX like lists

- list item 1
- list item 2

Mask Parameters

Parameter	Variable	Description
Parameter Name	<i>Variable Name</i>	Parameter Description
Parameter Name	<i>Variable Name</i>	Parameter Description

Ports

Port	Direction	Data Type	Description
<i>Port Name</i>	PORT DIRECTION	Port data type	Port Description
<i>Port Name</i>	IN	ufix_x.y	Port Description
<i>Port Name</i>	IN	inherited	Port Description

Description

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

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

Ports

Port	Direction	Data Type	Description
<i>sim_in</i>	IN	double	The analog signal to be digitized if interleave mode is selected. Note: For simulation only.
<i>sim_i</i>	IN	double	The first analog signal to be digitized if interleave mode is unselected. Note: For simulation only.
<i>sim_q</i>	IN	double	The second analog signal to be digitized if interleave mode is unselected. Note: For simulation only.
<i>sim_sync</i>	IN	double	Takes a pulse to be observed at the output to measure the delay through the block. Note: For simulation only.
<i>sim_data_valid</i>	IN	double	A signal that is high when inputs are valid. Note: For simulation only.
<i>oX</i>	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.
<i>iX</i>	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.
<i>qX</i>	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.
<i>outofrangeX</i>	OUT	boolean	A signal that represents when samples are outside the valid range.
<i>syncX</i>	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).
<i>data_valid</i>	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 *adcs* 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	<i>dac_brd</i>	Select which IBOB port to run this <i>dac</i> .
DAC clock rate (MHz)	<i>dac_clk_rate</i>	The clock rate to run the <i>dac</i> . Must be 4x FPGA clock rate.
Sample period	<i>sample_period</i>	Sets the period at which the <i>dac</i> outputs samples (ie 2 means every other cycle).
Show Implementation Parameters	<i>show_param</i>	Allows the user to set the implementation parameters.
Invert output clock phase	<i>invert_clock</i>	When unchecked, the <i>dac</i> samples the data aligned with the clock. When checked, the <i>dac</i> samples the data aligned with an inverted clock.

Ports

Port	Direction	Data Type	Description
<i>dataX</i>	IN	Fix_9_8	One of 4 digital inputs to be converted to analog.
<i>sim_out</i>	OUT	double	Analog output of <i>dac</i> . 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	<i>dimm</i>	Selects which physical DIMM to use (four per user FPGA).
Data Type	<i>arith_type</i>	Inform Simulink how it should interpret the stored data.
Data binary point	<i>bin_pt</i>	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)	<i>ip_clock</i>	Clock rate for DRAM. Default: 200MHz (400DDR).
Sample period	<i>sample_period</i>	Is significant for clocking the block. Default: 1
Simulate DRAM using ModelSim	<i>use_sim</i>	Requires the addition of the “ModelSim” block at the top level of the design. Used to simulate DRAM block only.
Enable bank management	<i>bank_mgt</i>	<i>Advise leave off</i> Changes the way the banks are addressed. Clarification required.
Use wide data bus (288 bits)	<i>wide_data</i>	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	<i>half_burst</i>	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	Direction	Data Type	Description
<i>rst</i>	IN	boolean	Resets the block when pulsed high
<i>address</i>	IN	UFix_32_0	A signal which accepts the address. See below for details.
<i>data_in</i>	IN	144 or 288 bit unsigned	Accepts data to be saved to DRAM.
<i>wr_be</i>	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.
<i>RWn</i>	IN	boolean	Selects read or not-write. <i>1</i> for read, <i>0</i> for write.
<i>cmd_tag</i>	IN	UFix_32_0	Accepts a user-defined tag for labelling entered commands.
<i>cmd_valid</i>	IN	boolean	Clocks data into the command buffer.
<i>rd_ack</i>	OUT	boolean	Used to acknowledge that the last <i>data_out</i> value has been read.
<i>cmd_ack</i>	OUT	boolean	Acknowledges that the last command was accepted (when buffer is full, will not accept additional commands).
<i>data_out</i>	OUT	UFix_144_0	Outputs data from DRAM, 144 bits at a time. Reads are in groups of 288 bits (ie 2 clocks).
<i>rd_tag</i>	OUT	UFix_32_0	Outputs the identifier for the data on <i>data_out</i> (as submitted on <i>cmd_tag</i> when the command was issued).
<i>rd_valid</i>	OUT	boolean	Indicates that the data on <i>data_out</i> 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 it’s 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. $4 \times 72\text{bits} = 288$ 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:

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

Each group of 8 addresses selects a 144 bit logical location (the lowest 3 bits are ignored). For example, address *0x00* through *0x7F* 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 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 ($2^?$)	<i>nsamples</i>	Specifies the depth of the Shared BRAM(s); i.e. the number of 32bit samples which are stored per capture.

Ports

Port	Direction	Data Type	Description
<i>din</i>	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.
<i>trig</i>	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.
<i>we</i>	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 ($2^?$)	<i>nsamples</i>	Specifies the depth of the Shared BRAM(s); i.e. the number of 64bit samples which are stored per capture.

Ports

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

Ports

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

3.7 Display name for Block (*simulink_name_for_block*)

Block Author: Block Author

Document Author: Document Author

Summary

Block Summary can contain arbitrary L^AT_EX like lists

- list item 1
- list item 2

Mask Parameters

Parameter	Variable	Description
Parameter Name	<i>Variable Name</i>	Parameter Description
Parameter Name	<i>Variable Name</i>	Parameter Description

Ports

Port	Direction	Data Type	Description
<i>Port Name</i>	PORT DIRECTION	Port data type	Port Description
<i>Port Name</i>	IN	ufix_x_y	Port Description
<i>Port Name</i>	IN	inherited	Port Description

Description

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