psi_fix

Documentation

Content

Table of Contents

Figures

1 Introduction

The purpose of this library is to provide HDL implementations for common fixed-point signal processing components along with bittrue Python models. The Python models are also callable from MATLAB.

This document serves as description of the RTL implementation for all components.

2 RTL Descriptions

2.1 psi_fix_bin_div

2.1.1 Description

This component implements a fixed point binary divider.

$$Quotient = \frac{Nomerator}{Denominator}$$

2.1.2 Generics

NumFmt_g	Numerator format
DenomFmt_g	Denominator format
QuotFmt_g	Quotient format
Round_g	Rounding mode at the output (round or truncate)
Sat_g	Saturation mode at the output (saturate of wrap)

2.1.3 Interfaces

Signal	Direction	Width	Description
Control Signals			
Clk	Input	1	Clock
Rst	Input	1	Reset
Input			
InVld	Input	1	AXI-S handshaking signal
InRdy	Output	1	AXI-S handshaking signal
InNum	Input	NumFmt_g	Numerator input
InDenom	Input	DenomFmt_g	Denominator input
Output			
OutVld	Output	1	AXI-S handshaking signal
OutQuot	Output	QuotFmt_g	Quotient output

At the input a handshaking for handling backpressure (incl. Rdy) is implemented since the binary divider is quite slow and may be the limiting component in offline data processing systems. At the output no handling for backpressure is implemented for simplicity reasons.

2.1.4 Architecture

The component converts numerator and denominator to unsigned numbers, so a standard binary divider can be implemented. At the output, the sign is restored correctly.

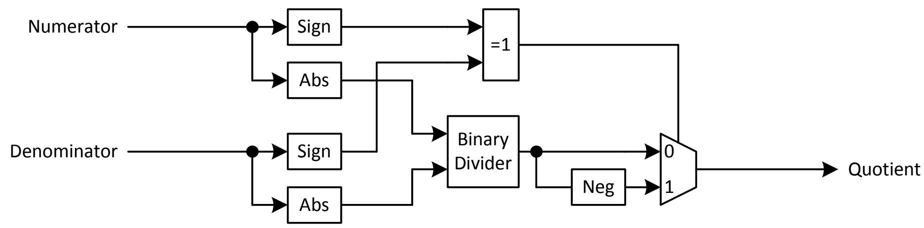

Figure 1: psi_fix_bin_div Architecture

2.2 psi_fix_cic_dec_fix_1ch

2.2.1 Description

This component implements a simple CIC decimator for a single channel. The decimation ratio must be known at compile time.

The CIC component always corrects the CIC gain roughly by shifting. As a result, the gain of the component is always between 0.5 and 1.0. Additionally a multiplier for exact gain adjustment can be added by setting the generic *AutoGainCorr_g* to true. In this case the gain is corrected to exactly 1.0.

2.2.2 Generics

Order_g	Order of the CIC filter (number of integrator/comb pairs)
Ratio_g	Decimation ratio
DiffDel_g	Delay for the comb sections (1 or 2)
InFmt_g	Input format
OutFmt_g	Output format
AutoGainCorr_g	True = compensate gain to 1.0, False = gain is between 0.5 and 1.0

2.2.3 Interfaces

Signal	Direction	Width	Description
Control Signals			
Clk	Input	1	Clock
Rst	Input	1	Reset
Input			
InVld	Input	1	AXI-S handshaking signal
InData	Input	InFmt_g	Denominator input
Output			
OutVld	Output	1	AXI-S handshaking signal
OutData	Output	InFmt_g	Quotient output

The CIC is able to process one input sample per clock cycle. Therefore no backpressure handling is implemented on the input.

CIC are most commonly used in streaming signal processing systems that require processing or storing the data at the full speed anyway. So no backpressure handling is implemented on the output side for simplicity

2.2.4 Architecture

The figure below shows the architecture of the CIC decimation filter.

Since the integrators are responsible for most of the CIC gain, the numbers are shifted and truncated after the integrator sections to the width required for producing less than 1 LSB error at the output. This allows saving some resources in the differentiator sections.

Note that the number format for the differentiator sections has one additional fractional bit (compared to the output format) per section. This results from the fact that depending on the signal frequency, the differentiators can have a gain up to two. This way the least significant bit at the input of the differentiators that can change the output by one LSB is preserved.

If the gain correction multiplier is used, signal path is chosen to be 25 bits wide and the gain correction coefficient is 17 bits (unsigned). For most implementations this design decisions are sufficient. If other requirements exist (e.g. very wide signal path), a project specific implementation of the CIC is required.

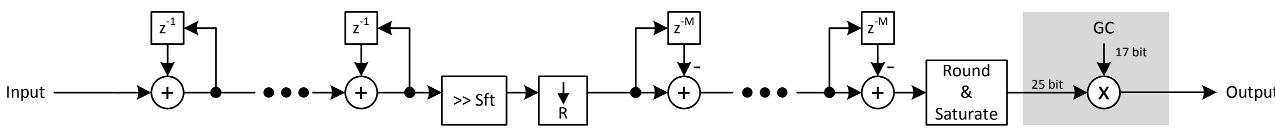

Figure 2: psi_fix_cic_dec_fix_1ch Architecture

The symbols are defined as follows:

R Decimation ratio
M Differential delay
N CIC order
Sft Number of bits to shift (to compensate overall gain to 0.5 < gain < 1.0)
GC Gain correction factor to compensate overall gain to 1.0

Some of the most common formulas are given below.

$$Gain_{CIC} = (R \cdot M)^N$$

$$Sft = ceil(\log_2(Gain_{CIC}))$$

For the case that the gain correction amplifier is disabled, the overall gain of the CIC is:

$$GainOverallNoGc = \frac{Gain_{CIC}}{2^{Sft}}$$

Since this formula evaluates to 1.0 for the case $R = x^2$ (decimation ratio is a power of two), the gain correction multiplier is not required in this case.

The optimal setting for the differential delay depends on the use case. Only the values 1 and 2 are supported. Other values are uncommon in real-life. Usually 1 is used if an FIR filter follows the CIC to further reduce the passband. If no FIR follows the CIC, a value 2 to is more optimal to avoid strong aliasing.

2.3 psi_fix_cic_int_fix_1ch

2.3.1 Description

This component implements a simple CIC interpolator for a single channel. The interpolation ratio must be known at compile time.

The CIC component always corrects the CIC gain roughly by shifting. As a result, the gain of the component is always between 0.5 and 1.0. Additionally a multiplier for exact gain adjustment can be added by setting the generic *AutoGainCorr_g* to true. In this case the gain is corrected to exactly 1.0.

2.3.2 Generics

Order_g	Order of the CIC filter (number of integrator/comb pairs)
Ratio_g	Interpolation ratio
DiffDel_g	Delay for the comb sections (1 or 2)
InFmt_g	Input format
OutFmt_g	Output format
AutoGainCorr_g	True = compensate gain to 1.0, False = gain is between 0.5 and 1.0

2.3.3 Interfaces

Signal	Direction	Width	Description
Control Signals			
Clk	Input	1	Clock
Rst	Input	1	Reset
Input			
InVld	Input	1	AXI-S handshaking signal
InRdy	Output	1	AXI-S handshaking signal
InData	Input	InFmt_g	Denominator input
Output			
OutVld	Output	1	AXI-S handshaking signal
OutRdy	Input	1	AXI-S handshaking signal
OutData	Output	InFmt_g	Quotient output

The CIC interpolator requires full handshaking including the handling of back-pressure at the input since it can only take one sample every N clock cycles. As a result, the *InRdy* signal is required to signal when an input sample was processed.

Full handshaking at the output side was implemented mainly to allow equally spaced output samples (in time). By nature the filter calculates multiple output samples back-to-back after an input sample arrived. For output rates lower than the clock-speed, this leads to a bursting behavior which is often (but not always) undesirable. By controlling the *OutRdy* signal, the user can control the output sample-rate and –spacing exactly.

2.3.4 Architecture

The figure below shows the architecture of the CIC interpolation filter.

Note that the number format for the differentiator sections has one additional integer bit (compared to the input format) per section. This results from the fact that depending on the signal frequency, the differentiators can have a gain up to two.

If the gain correction multiplier is used, signal path is chosen to be 25 bits wide and the gain correction coefficient is 17 bits (unsigned). For most implementations this design decisions are sufficient. If other requirements exist (e.g. very wide signal path), a project specific implementation of the CIC is required.

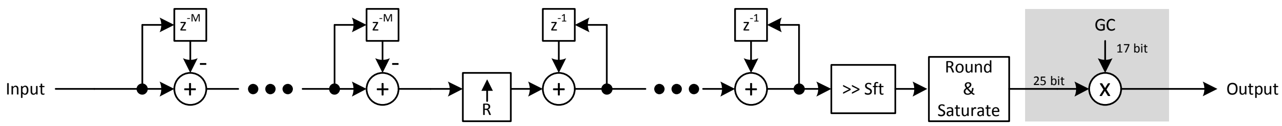

Figure 3: psi_fix_cic_int_fix_1ch Architecture

The symbols are defined as follows:

R	Interpolation ratio
M	Differential delay
N	CIC order
Sft	Number of bits to shift (to compensate overall gain to 0.5 < gain < 1.0)
GC	Gain correction factor to compensate overall gain to 1.0

Some of the most common formulas are given below.

$$Gain_{CIC} = \frac{(R \cdot M)^N}{R}$$

$$Sft = ceil(\log_2(Gain_{CIC}))$$

For the case that the gain correction amplifier is disabled, the overall gain of the CIC is:

$$GainOverallNoGc = \frac{Gain_{CIC}}{2^{Sft}}$$

Since this formula evaluates to 1.0 for the case $R = x^2$ (interpolation ratio is a power of two), the gain correction multiplier is not required in this case.

The optimal setting for the differential delay depends on the use case. Only the values 1 and 2 are supported. Other values are uncommon in real-life. Usually 1 is used if the input signal is already oversampled (does not contain frequency components close to $\frac{fs}{2}$) and 2 is used otherwise.

Note that the CIC does not control timing on its own. This means by default, the CIC outputs one sample per clock cycle. If the input sample rate is slow, the output is bursting. If the time between two output samples has to be constant, the timing can be controlled by applying pulses at the desired frequency to the *OutRdy* handshaking signal. The reason for the CIC to not control any timing at the output is that this is a library component and it may also be used in offline processing algorithms.

2.4 psi_fix_cordic_abs_pl

2.4.1 Description

This component implements the absolute value calculation based on the CORDIC algorithm. Depending on the parameters, up to one pipeline stage per iteration can be implemented. This allows achieving even highest performance requirements.

Note that this component does not compensate the CORDIC gain. If this is required, the compensation of the CORDIC gain must be implemented externally.

2.4.2 Generics

InFmt_g Input format (must be signed)
OutFmt_g Output format (must be unsigned since this is an absolute value)
InternalFmt_g Number format used for all CORDIC calculations
Iterations_g Number of CORDIC iterations to execute
PipelineFactor_g A pipeline stage is implemented after every N iterations (1 = fully pipelined)
Round_g Rounding mode at the output (round or truncate)
Sat_g Saturation mode at the output (saturate of wrap)

2.4.3 Interfaces

Signal	Direction	Width	Description
Control Signals			
Clk	Input	1	Clock
Rst	Input	1	Reset
Input			
InVld	Input	1	AXI-S handshaking signal
InI	Input	InFmt_g	In-phase signal input
InQ	Input	InFmt_g	Quadrature-phase signal input
Output			
OutVld	Output	1	AXI-S handshaking signal
OutAbs	Output	OutFmt_g	Result output

The CORDIC implementation is fully pipelined. This means it can take one input sample every clock cycle. As a result the handling of backpressure was not implemented.

2.4.4 Architecture

The CORDIC algorithm for the calculation of the absolute value is defined by the formulas below.

$$x_{i+1} = x_i - y_i \cdot d_i \cdot 2^{-i}$$

$$y_{i+1} = y_i + x_i \cdot d_i \cdot 2^{-i}$$

$$d_i = +1\ if\ y_i < 0, else - 1$$

The algorithm only works for $x \geq 0$, therefore the absolute value of x is calculated prior to executing the algorithm.

The CORDIC gain can be calculated by the formula below:

$$G_{CORDIC} = \prod_{i=0}^{N-1} \sqrt{1 + 2^{-2i}}$$

Where:

G_{CORDIC} Cordic Gain
N Number of iterations

The formula converges towards 1.646760 with high numbers of iterations.

The amount of Pipelining to be implemented can be chosen using the generic *PipelineFactor_g*. However, the amount of logic (LUT) required does not change much with reduced pipelining. The main reason for reducing the amount of pipelining is latency reduction.

2.5 psi_fix_fir_dec_ser_nch_chpar_conf

2.5.1 Description

This entity was initially implemented as multi-channel filter with configurable coefficients. ***However, it can also be used efficiently for single-channel FIRs and for filters with fixed coefficients.***

This entity implements a multi-channel decimating FIR filter. All channels are processed in parallel (not TDM) but there is only one multiplier for each channel, so the taps of a channel are calculated one after the other. The filter coefficients, the order and the decimation rate are runtime configurable.

2.5.2 Generics

InFmt_g	Input format
OutFmt_g	Output format
CoefFmt_g	Coefficient format
Channels_g	Number of parallel channels
MaxRatio_g	Maximum decimation ratio supported
MaxTaps_g	Maximum number of taps supported
Rnd_g	Rounding mode at the output (round or truncate)
Sat_g	Saturation mode at the output (saturate of wrap)
UseFixCoefs_g	If true, fixed coefficients instead of configurable coefficients are implemented.
FixCoefs_g	Coefficients to use for *UseFixCoefs_g* = true.

2.5.3 Interfaces

Signal	Direction	Width	Description
Control Signals			
Clk	Input	1	Clock
Rst	Input	1	Reset
Input			
InVld	Input	1	AXI-S handshaking signal
InData	Input	$InFmt\_g \cdot Channels\_g$	Input data in parallel - Channel 0 [N-1:0] - Channel 1 [2*N-1:0] - …
Output			
OutVld	Output	1	AXI-S handshaking signal
OutAbs	Output	$OutFmt\_g \cdot Channels\_g$	Output data in parallel (see *InData*)
Configuration			
Ratio	Input	$ceil(\log_2(\mathrm{MaxRatio\_g}))$	Decimation ratio -1 0 → no decimation 1 → decimation by 2) This port is optional. If it is not connected, *MaxRatio_g* is used as fixed ratio.

Taps	Input	$ceil(\log_2(\mathrm{MaxTaps\_g}))$	Taps – 1 0 → 1 Tap (order 0 filter) 63 → 64 Taps (order 63 filter) This port is optional. If it is not connected, *MaxTaps_g* is used as fixed tap count.
Coefficient Interface			
CoefClk	Input	1	Clock for the coefficient interface. This port can be left unconnected for fixed coefficient implementation (*UseFixCoefs_g* = true)
CoefWr	Input	1	Coefficient write enable signal This port can be left unconnected for fixed coefficient implementation (*UseFixCoefs_g* = true)
CoefAddr	Input	$ceil(\log_2(\mathrm{MaxTaps\_g}))$	Address of the coefficient to access This port can be left unconnected for fixed coefficient implementation (*UseFixCoefs_g* = true)
CoefWrData	Input	CoefFmt_g	Coefficient value for write access (*CoefWr = 1)* This port can be left unconnected for fixed coefficient implementation (*UseFixCoefs_g* = true)
CoefRdData	Output	CoefFmt_g	Coefficient read data (valid 1 cycle after applying the address) This port can be left unconnected for fixed coefficient implementation (*UseFixCoefs_g* = true)

The coefficient interface has a separate clock since often the data processing clock is coupled to an ADC clock but the main bus system that configures the filter is running on a different clock.

The filter can continue taking new input data even if a calculation is ongoing. As a result, the handling of packpressure is not required as long as the processing power of the filter is sufficient to handle all input data. For the calculation, see below.

Note that the behavior of the filter is undefined if the maximum input rate that can be handles is exceeded.

2.5.4 Architecture

The figure below roughly shows the architecture of the FIR filter. Since the filter assumes all channels arrive in parallel with the same timing, the coefficient RAM is shared between all channels to save resources.

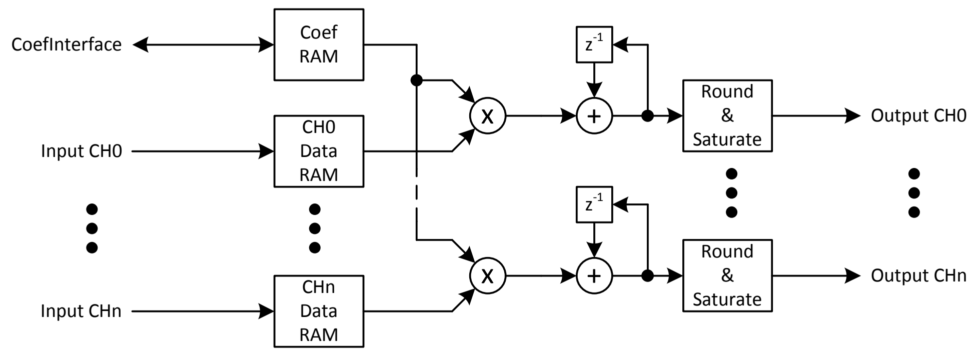

Figure 4: psi_fix_fix_dec_ser_nch_chpar_conf Architecture

A state machine (not shown in the figure for simplicity) starts a new calculation whenever all required input samples for the next calculation arrived.

The accumulation is executed at the full output precision of the multiplication. This matches the implementation of the DSP slices in Xilinx devices, so they can be fully utilized.

The accumulator contains one guard bit compared to the output format to detect overflows. However, the user (designer who integrates the filter) is responsible to choose coefficients in a way that the output format is never exceeded by more than a factor of two. This this is not possible the filter output format must be chosen large enough ($Range_{Output} \geq 0.5 \cdot MaximumOutput$) and saturated externally.

Obviously the architecture requires one clock cycle per tap calculation. As a result the maximum number of filter taps depends on the clock frequency F_{clk}, the input sample rate $F_{s,in}$ and the decimation ratio R.

$$Taps_{max} = \frac{F_{clk} \cdot R}{F_{s,in}}$$

In case of fixed coefficient implementation, the coefficient RAM is replaced by a ROM automatically.

2.6 psi_fix_fir_dec_ser_nch_chtdm_conf

2.6.1 Description

This entity was initially implemented as filter with configurable coefficients. ***However, it can also be used efficiently for filters with fixed coefficients.***

This component implements a multi-channel decimating FIR filter. All channels are processed TDM (one after the other). The multiplications are all executed using the same multiplier, so the taps of a channel are calculated one after the other. The filter coefficients, the order and the decimation rate are runtime configurable.

2.6.2 Generics

InFmt_g	Input format
OutFmt_g	Output format
CoefFmt_g	Coefficient format
Channels_g	Number of parallel channels (1 is not supported, must be >= 2)
MaxRatio_g	Maximum decimation ratio supported
MaxTaps_g	Maximum number of taps supported
Rnd_g	Rounding mode at the output (round or truncate)
Sat_g	Saturation mode at the output (saturate of wrap)
UseFixCoefs_g	If true, fixed coefficients instead of configurable coefficients are implemented.
FixCoefs_g	Coefficients to use for *UseFixCoefs_g* = true.

2.6.3 Interfaces

Signal	Direction	Width	Description
Control Signals			
Clk	Input	1	Clock
Rst	Input	1	Reset
Input			
InVld	Input	1	AXI-S handshaking signal
InData	Input	InFmt_g	Input data, one channel is passed after the other
Output			
OutVld	Output	1	AXI-S handshaking signal
OutAbs	Output	OutFmt_g	Output data, one channel is passed after the other
Configuration			
Ratio	Input	$ceil(\log_2(\text{MaxRatio\_g}))$	Decimation ratio -1 0 → no decimation 1 → decimation by 2) This port is optional. If it is not connected, *MaxRatio_g* is used as fixed ratio.

Taps	Input	$ceil(\log_2(\text{MaxTaps\_g}))$	Taps – 1 0 → 1 Tap (order 0 filter) 63 → 64 Taps (order 63 filter) This port is optional. If it is not connected, *MaxTaps_g* is used as fixed tap count.
Coefficient Interface			
CoefClk	Input	1	Clock for the coefficient interface This port can be left unconnected for fixed coefficient implementation (*UseFixCoefs_g* = true)
CoefWr	Input	1	Coefficient write enable signal This port can be left unconnected for fixed coefficient implementation (*UseFixCoefs_g* = true)
CoefAddr	Input	$ceil(\log_2(\text{MaxTaps\_g}))$	Address of the coefficient to access This port can be left unconnected for fixed coefficient implementation (*UseFixCoefs_g* = true)
CoefWrData	Input	CoefFmt_g	Coefficient value for write access (*CoefWr = 1)* This port can be left unconnected for fixed coefficient implementation (*UseFixCoefs_g* = true)
CoefRdData	Output	CoefFmt_g	Coefficient read data (valid 1 cycle after applying the address) This port can be left unconnected for fixed coefficient implementation (*UseFixCoefs_g* = true)

The coefficient interface has a separate clock since often the data processing clock is coupled to an ADC clock but the main bus system that configures the filter is running on a different clock.

The filter can continue taking new input data even if a calculation is ongoing. As a result, the handling of packpressure is not required as long as the processing power of the filter is sufficient to handle all input data. For the calculation, see below.

Note that the behavior of the filter is undefined if the maximum input rate that can be handles is exceeded.

2.6.4 Architecture

The figure below roughly shows the architecture of the FIR filter. Since the channels arrive one after the other, the one dual-port RAM is sufficient to store all data. The RAM is split into different regions (i.e. the higher address bits select the region reserved for a given channel).

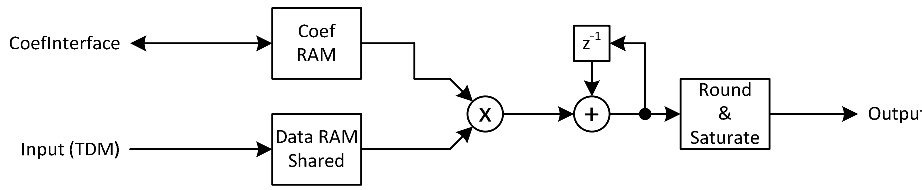

Figure 5: psi_fix_fix_dec_ser_nch_chtdm_conf Architecture

A state machine (not shown in the figure for simplicity) starts a new calculation whenever all required input samples for the next calculation arrived.

The accumulation is executed at the full output precision of the multiplication. This matches the implementation of the DSP slices in Xilinx devices, so they can be fully utilized.

The accumulator contains one guard bit compared to the output format to detect overflows. However, the user (designer who integrates the filter) is responsible to choose coefficients in a way that the output format is never exceeded by more than a factor of two. This this is not possible the filter output format must be chosen large enough ($Range_{Output} \geq 0.5 \cdot MaximumOutput$) and saturated externally.

Obviously the architecture requires one clock cycle per tap calculation of one channel. As a result the maximum number of filter taps depends on the number of channels N_{CH} clock frequency F_{clk}, the input sample rate $F_{s,in}$ and the decimation ratio R.

$$Taps_{max} = \frac{F_{clk} \cdot R}{F_{s,in} \cdot N_{CH}}$$

In case of fixed coefficient implementation, the coefficient RAM is replaced by a ROM automatically.

Important note: Changing the decimation rate and/or the filter order at runtime can temporarily lead to inconsistent settings because usually they are changed by register accesses that are executed one after the other. To avoid this problem, it is suggested to keep the filter in reset whenever the parameters are changed.

2.7 psi_fix_lin_approx_<function>

2.7.1 Description

This is actually not just one component but a whole family of components. They are all function approximations based on a table containing the function values for regularly spaced points and linear approximation between them.

All components are based on the same implementation of the approximation (*psi_fix_lin_approx_calc.vhd*) and they only vary in number formats and coefficient tables.

The code is not written by hand but generated from Python (*psi_fix_lin_approx.py)*. If a new function approximation shall be developed, it can first be designed using the function *psi_fix_lin_approx.Design()* that also helps finding the right settings. Afterwards VHDL code and a corresponding bittrueness testbench can be generated using *psi_fix_lin_approx.GenerateEntity()* and *psi_fix_lin_approx.GenerateTb()*.

2.7.2 Generics

Sinde each function approximation is built for an exact input range, precision and function, no parameters are required.

2.7.3 Interfaces

Signal	Direction	Width	Description
Control Signals			
Clk	Input	1	Clock
Rst	Input	1	Reset
Input			
InVld	Input	1	AXI-S handshaking signal
InData	Input	*	Signal input
Output			
OutVld	Output	1	AXI-S handshaking signal
OutData	Output	*	Result output

* The width of these ports depends on the specific function approximation.

The implementation of the linear approximation is fully pipelined. This means it can take one input sample every clock cycle. As a result the handling of backpressure was not implemented.

2.7.4 Architecture

The figure below shows the interpolation principle.

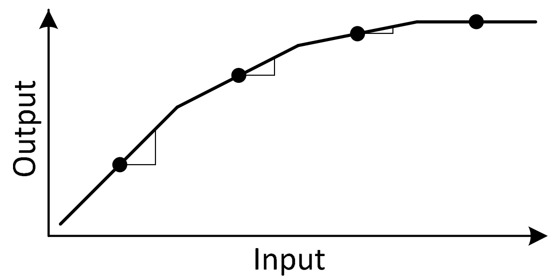

Figure 6: psi_fix_lin_approx Interpolation Principle

The complete range of the function is split into small sections. For each section the center point as well as the gradient are known and the output value is calculated from these two values (together with the difference between actual input and center point of the current segment).

The figure below shows the implementation of the approximation.

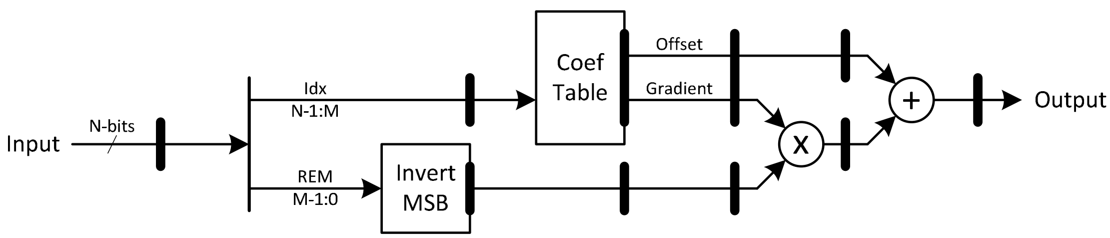

Figure 7: psi_fix_lin_approx Architecture

After splitting the input into index and reminder, the reminder is unsigned and related to the beginning of the segment. By inverting the MSB, the reminder is converted to the signed offset related to the center point of the segment.

2.8 psi_fix_dds_18b

2.8.1 Description

This entity implements an 18-bit DDS. The sine-wave is generated using the entity *psi_fix_lin_approx_sin_18b* and it has an error of less than one LSB for all values. As a result, there are no significant spurs in the generated spectrum (significant in terms of above the quantization noise floor) as shown in the figure below.

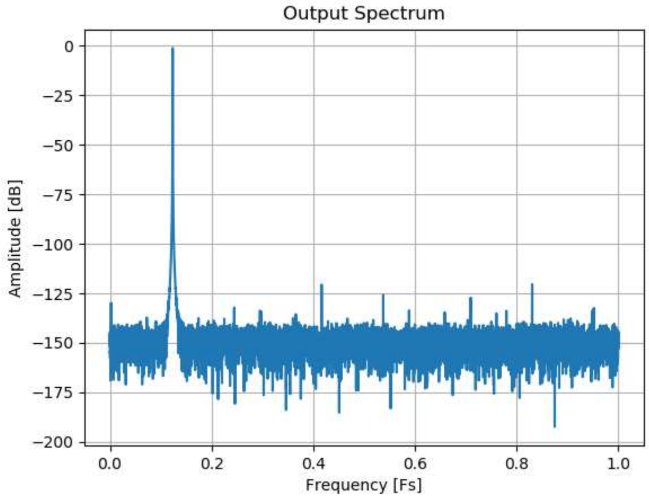

Figure 8: psi_fix_dds_18b Spectrum for PhaseStep=0.12345

2.8.2 Generics

PhaseFmt_g Phase accumulator format. This must be a number format with a range of 1.0 (either [0,0,x] or [1,-1,x]). A phase of 1.0 corresponds to 2π resp. one fully sine period.

2.8.3 Interfaces

Signal	Direction	Width	Description
Control Signals			
Clk	Input	1	Clock
Rst	Input	1	Reset
Configuration			
Restart	Input	1	This signal can be used to start the DDS again at the phase offset. This is useful if 100% reproducible outputs must be generated several times.
PhaseStep	Input	PhaseFmt_g	Phase step between two consecutive output samples. The phase step is given in 2π (0.5 corresponds to π). The phase step can be changed at runtime safely.
PhaseOffset	Input	PhaseFmt_g	Phase offset of the generated signal. The phase offset is given in 2π (0.5 corresponds to π). The phase offset can be changed at runtime safely.

Input			
InVld	Input	1	AXI-S handshaking signal that can be used to generate samples at any rate. For continuous operation (one sample per clock cycle) , the signal can be left unconnected.
Output			
OutVld	Output	1	AXI-S handshaking signal
OutSin	Output	18	Sine wave output in the format [1,0,17]
OutCos	Output	18	Cosine wave output in the format [1,0,17]

The total pipeline delay of the DDS is 9 clock cycles.

2.8.4 Architecture

The figure below shows the implementation of the DDS.

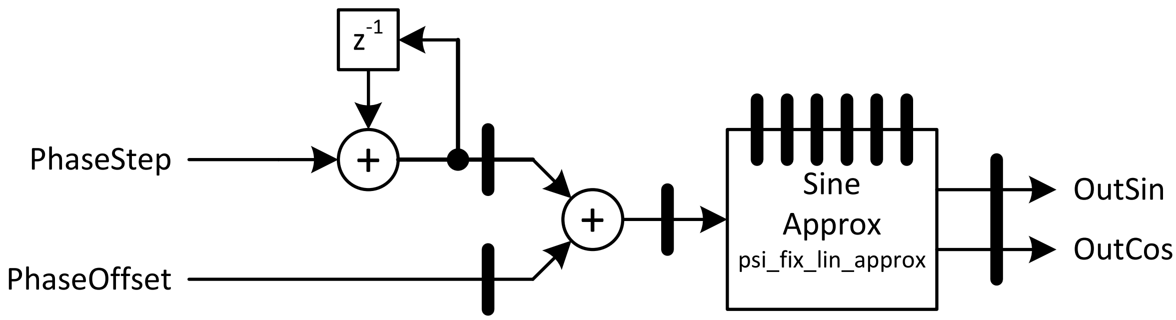

Figure 9: psi_fix_dds_18b Architecture

2.9 psi_fix_complex_mult

2.9.1 Description

The block performs multiplication on a complex number pair (*Inphase* & *Quadrature*, inputs of the block) or 2D matrix computation, let two complex numbers be:

$$x = (a + ib); y = (c + id)$$

The multiplication result comes:

$$x.y = (a + ib)(c + id) = (ac - bd) + i(ad + bc)$$

Where: In-phase input=a; Quadrature input=b; I1=c; I2=d; Q1=I2=d; Q2=I1=c

The block could be seen as well as 2D matrix multiplication, apart from the fact that a subtraction is hardcoded on the in-phase path and the given processing is equal as the one shown below:

$$\begin{bmatrix} Iout \\ Qout \end{bmatrix} = \begin{bmatrix} Inphase \\ Quadrature \end{bmatrix} \times \begin{bmatrix} I1 & -I2 \\ Q1 & Q2 \end{bmatrix} = \begin{matrix} Inphase \times I1 - Quadrature \times I2 \\ Inphase \times Q1 + Quadrature \times Q2 \end{matrix}$$

The total pipeline delay of the block is 3 clock cycles if no pipeline activation is set through generics, otherwise the pipeline is doubled (i.e. 6 stages)

2.9.2 Generics

RstPol_g set the reset polarity
Pipeline_g Add internal register pipeline to get higher clock frequency synthesis result
InFixFmt_g Input format
InternalFmt_g Internal format
CoefFmt_g Coefficient format
OutFmtr_g Output format

2.9.3 Interfaces

Signal	Direction	Width	Description
Control Signals			
clk_i	Input	1	Clock
rst_i	Input	1	Synchronous Reset
Input			
ipath_i	Input	InFixFmt_g	Real part of complex number input (in-phase data)
qpath_i	Input	InFixFmt_g	Imaginary part of complex number input (quadrature data)
vld_i	Input	1	Data strobe input
i1_i	Input	CoefFmt_g	Please refer to calculation description above §2.9.1
i2_i	Input	CoefFmt_g	Please refer to calculation description above §2.9.1
q1_i	Input	CoefFmt_g	Please refer to calculation description above §2.9.1

q2_i	Input	CoefFmt_g	Please refer to calculation description above §2.9.1
Output			
vld_o	Output	1	Data strobe output
iout_o	Output	OutFmt_g	Real part of complex number output (in-phase data)
out_o	Output	OutFmt_g	Imaginary part of complex number output (quadrature data)

2.9.4 Architecture

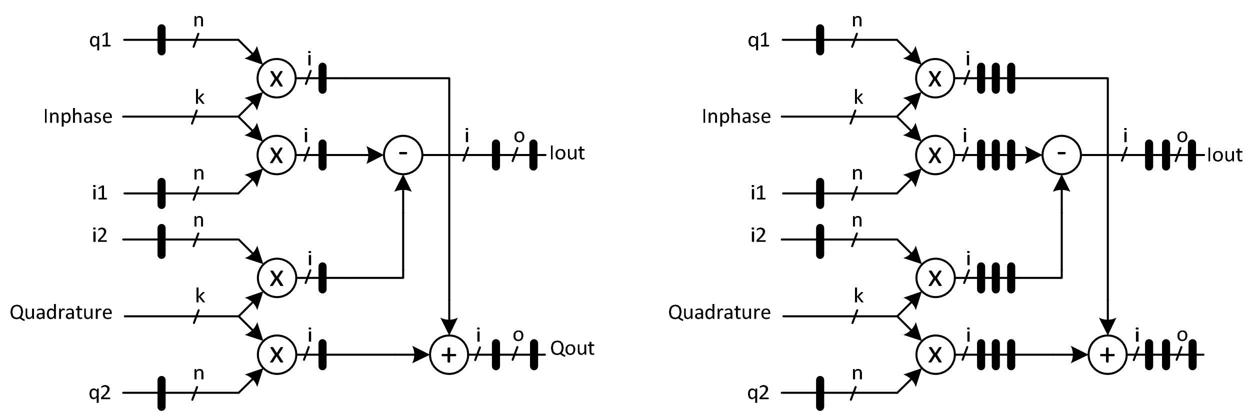

Figure 10: psi_fix_complex_mult Architecture – Pipeline_g = 0 (left) Pipeline_g = 1