psi\_fix

Documentation

**Content**

**Table of Contents**

[1 Introduction 3](#_Toc507059976)

[2 RTL Descriptions 4](#_Toc507059977)

[2.1 psi\_fix\_bin\_div 4](#_Toc507059978)

[2.2 psi\_fix\_cic\_dec\_fix\_1ch 6](#_Toc507059979)

[2.3 psi\_fix\_cic\_int\_fix\_1ch 8](#_Toc507059980)

[2.4 psi\_fix\_cordic\_abs\_pl 10](#_Toc507059981)

[2.5 psi\_fix\_fir\_dec\_ser\_nch\_chpar\_conf 12](#_Toc507059982)

[2.6 psi\_fix\_fir\_dec\_ser\_nch\_chtdm\_conf 14](#_Toc507059983)

[2.7 psi\_fix\_lin\_approx\_<function> 16](#_Toc507059984)

**Figures**

[Figure 1: psi\_fix\_bin\_div Architecture 5](#_Toc507059985)

[Figure 2: psi\_fix\_cic\_dec\_fix\_1ch Architecture 7](#_Toc507059986)

[Figure 3: psi\_fix\_cic\_int\_fix\_1ch Architecture 9](#_Toc507059987)

[Figure 4: psi\_fix\_fix\_dec\_ser\_nch\_chpar\_conf Architecture 13](#_Toc507059988)

[Figure 5: psi\_fix\_fix\_dec\_ser\_nch\_chtdm\_conf Architecture 15](#_Toc507059989)

[Figure 6: psi\_fix\_lin\_approx Interpolation Principle 17](#_Toc507059990)

[Figure 7: psi\_fix\_lin\_approx Architecture 17](#_Toc507059991)

# 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.

# RTL Descriptions

## psi\_fix\_bin\_div

### Description

This component implements a fixed point binary divider.

### 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)

### 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.

### 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

## psi\_fix\_cic\_dec\_fix\_1ch

### 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.

### 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

### 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

### 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:

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

Some of the most common formulas are given below.

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

Since this formula evaluates to 1.0 for the case (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.

## psi\_fix\_cic\_int\_fix\_1ch

### 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.

### 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

### 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.

### 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:

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

Some of the most common formulas are given below.

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

Since this formula evaluates to 1.0 for the case (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 ) 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.

## psi\_fix\_cordic\_abs\_pl

### 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.

### 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)

### 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.

### Architecture

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

The algorithm only works for , therefore the absolute value of is calculated prior to executing the algorithm.

The CORDIC gain can be calculated by the formula below:

Where:  
 Cordic Gain  
 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.

## psi\_fix\_fir\_dec\_ser\_nch\_chpar\_conf

### Description

This component 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.

### 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)

### 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 |  | 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 |  | Output data in parallel (see *InData*) |
| ***Configuration*** | | | |
| Ratio | Input |  | Decimation ratio -1 0 🡪 no decimation 1 🡪 decimation by 2) |
| Taps | Input |  | Taps – 1 0 🡪 1 Tap (order 0 filter) 63 🡪 64 Taps (order 63 filter) |
| ***Coefficient Interface*** | | | |
| CoefClk | Input | 1 | Clock for the coefficient interface |
| CoefWr | Input | 1 | Coefficient write enable signal |
| CoefAddr | Input |  | Address of the coefficient to access |
| CoefWrData | Input | CoefFmt\_g | Coefficient value for write access (*CoefWr = 1)* |
| CoefRdData | Output | CoefFmt\_g | Coefficient read data (valid 1 cycle after applying the address) |

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.

### 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 ( ) 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 , the input sample rate and the decimation ratio .

## psi\_fix\_fir\_dec\_ser\_nch\_chtdm\_conf

### Description

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.

### 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)

### 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 |  | Decimation ratio -1 0 🡪 no decimation 1 🡪 decimation by 2) |
| Taps | Input |  | Taps – 1 0 🡪 1 Tap (order 0 filter) 63 🡪 64 Taps (order 63 filter) |
| ***Coefficient Interface*** | | | |
| CoefClk | Input | 1 | Clock for the coefficient interface |
| CoefWr | Input | 1 | Coefficient write enable signal |
| CoefAddr | Input |  | Address of the coefficient to access |
| CoefWrData | Input | CoefFmt\_g | Coefficient value for write access (*CoefWr = 1)* |
| CoefRdData | Output | CoefFmt\_g | Coefficient read data (valid 1 cycle after applying the address) |

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.

### 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 ( ) 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 clock frequency , the input sample rate and the decimation ratio .

**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.

## psi\_fix\_lin\_approx\_<function>

### 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()*.

### Generics

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

### 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.

### 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.