Type	Component Name	XPartition Versions					Selectable Partitions	Input Types Supported	Inputs	Outputs	Functionality	Generation Time Configurable Parameters	Run Time Programmable Parameters	
		2022	2023	2024	2025	2026							AXI4-Lite Internal Registers	Using External Memory Mapped Registers
Arithmetic Components	Adder	●	●	●	●	●	Hardware Software	Constant Axis Simple Streams	in1 (integer) in2 (integer)	out (integer)	out = in1 + in2	out: Bit width in1: Bit width. (Ins) in2: Bit width. (Ins)	None.	None.
	Subtractor	●	●	●	●	●	Hardware Software	Constant Axis Simple Streams	in1 (integer) in2 (integer)	out (integer)	out = in1 - in2	out: Bit width. in1: Bit width. (Ins) in2: Bit width. (Ins)	None.	None.
	Multiplier	●	●	●	●	●	Hardware Software	Constant Axis Simple Streams	in1 (integer) in2 (integer)	out (integer)	out = in1 * in2	out: Bit width. in1: Bit width. (Ins) in2: Bit width. (Ins)	None.	None.
	Absolute Value	●	●	●	●	●	Hardware Software	Constant Axis Simple Streams	in sign_out (hierarchy)	out sign_out (hierarchy)	out = ((* in) if in >1, else out = in sign_out = sign bit of out (see below)	in: Bit width (= out Bit width).	None.	None.
	Negate	●	●	●	●	●	Hardware Software	Constant Axis Simple Streams	in enable	out sign_out	out = ((* in) if enable = 1, else out = in sign_out = sign bit of out	in: Bit width (= out Bit width).	None.	Enable input (enable).
	Scaler		●	●	●	●	Hardware	Constant Axis Simple Streams	in (integer) inv_scale (integer) fixed_scale (integer)	out (integer)	out = in * (inv_scale > fixed_scale ? 1/inv_scale : fixed_scale)	in: Bit width. out: Bit width.	None.	Scale Factor (integer input (inv_scale). Scale Factor Fractional input (fixed_scale).
	Fractional Adder					●	Hardware	Sample Streams	in1 (integer + Fraction) in2 (integer + Fraction)	out (integer + Fraction)	out = in1 + in2 Inputs and output are fractional streams with an integer part and a fractional part.	Integer Bit width (in). Fraction Bit width. (in/out).	None.	None.
	Fractional Subtractor					●	Hardware	Sample Streams	in1 (integer + Fraction) in2 (integer + Fraction)	out (integer + Fraction)	out = in1 - in2 The inputs and output are fractional streams with an integer part and a fractional part.	Integer Bit width (in). Fraction Bit width. (in/out).	None.	None.
	Fractional Multiplier					●	Hardware	Sample Streams	in1 (integer + Fraction) in2 (integer + Fraction)	out (integer + Fraction)	out = in1 * in2 The inputs and output are fractional streams with an integer part and a fractional part.	Integer Bit width (in). Fraction Bit width. (in/out).	None.	None.
	Divider					●	Hardware Software	Sample Streams	in1 (integer) in2 (integer)	out (integer + Fraction)	out = in1 / in2 Inputs are integers. The output is a fractional stream with an integer part and a fractional part.	Integer Bit width. (in/out) Fraction Bit width. (out).	None.	None.
	Reciprocal					●	Hardware Software	Sample Streams	in (integer)	out (integer)	out = (1 << bit_f)*2^exponte_out_width)	in: Bit width. out: Bit width.	None.	None.
	Square Root					●	Hardware Software	Sample Streams	in (integer) num_in	out (integer) num_out	out = $\sqrt{num_in}$ num_in = num_in_bit_in[sizeof(num)]	in: Bit width (= out Bit width).	None.	None.
	Root-of-Sum-of-Squares					●	Hardware Software	Sample Streams	in1 (integer) in2 (integer)	out (integer)	out = $\sqrt{(in1^2 + in2^2)}$	in1: Bit width (= out Bit width).	None.	None.
	Normalizer				●	●	Hardware	Sample Streams	in	out num_out	out = in >> (max_zero_bit_in-1-N) num_out = (*)(max_zero_bit_in-N)	in: Bit width (= out Bit width). Example: Each 16-bit Input can fill 16-bit Numbers. Save 16-bit Only on 32-bit Slots.	None.	None.
	Concatenator				●	●	Hardware	Sample Streams	in1 in2 ... inN num_ins_in	out	out = $\{in1, in2, ..., inN\}$ num_ins_in = N	in: Bit width (= out Bit width).	None.	None.
	Root-Determinator					●	Hardware	Sample Streams	in num_ins_in	out	out = $(-1)^{(1/num_ins_in)}$ * in	in: Bit width (= out Bit width).	None.	None.
Signal Analysis Components	Period and Duty-cycle Detector					●	Hardware	Sample Streams	enable reset_both_bits continuous_mode_sel in (stream)	period_period * duty_period * period_amplitude * duty_amplitude * threshold_value counts_ready	period_period = Number of clocks counted during input pulse waveform active high duration. duty_period = Number of clocks counted during input pulse waveform active high duration. period_amplitude = Average amplitude during active low duration. duty_amplitude = Average amplitude during active high duration. threshold_value = Amplitude threshold to detect positive and negative pulse waveform transitions from low and full. counts_ready = Indicates when period and duty_period values are ready to be read.	in: Bit width. Supports AXI4-Stream & Continuous input waveform mode selection. Input sumpler average count. Sample Rate Shifts (only on AXI4-Stream) for pulse waveform threshold point and period learning.	None.	None.
Bit Operators	Shift	●	●	●	●	●	Hardware Software	Constant Axis Simple Streams	in	out	out = in << shift_count (if left_shift) out = in >> shift_count (if right_shift)	in: Bit width (= out Bit width). shift_count: Bit width. Left-shift or Right-shift.	None.	None.
	Cycle	●	●	●	●	●	Hardware	Constant Axis	in	out*	out[i] = in[i-1] out[0] = in[sample_width-1]	in/bit width.	None.	None.
	Merge	●	●	●	●	●	Hardware	Constant Axis	in*	out	out = { (msb:sample_width-1) ... (lsb) }	Number of Input Bits to be merged.	None.	None.
	OR	●	●	●	●	●	Hardware Software	Constant Axis Simple Streams	in1 in2	out	out = (in1 | in2) (Bit-wise OR function)	out: Bit width (= in1 Bit width = in2 Bit width).	None.	None.
	AND	●	●	●	●	●	Hardware Software	Constant Axis Simple Streams	in1 in2	out	out = (in1 & in2) (Bit-wise AND function)	out: Bit width (= in1 Bit width = in2 Bit width).	None.	None.
	XOR	●	●	●	●	●	Hardware Software	Constant Axis Simple Streams	in1 in2	out	out = (in1 ^ in2) (Bit-wise XOR function)	out: Bit width (= in1 Bit width = in2 Bit width).	None.	None.
	Invert	●	●	●	●	●	Hardware	Constant Axis Simple Streams	in	out	out = ~in (Bit-wise NOT function)	in: Bit width (= out Bit width).	None.	None.
Generic DSP Components	Moving Average				●	●	Hardware Software	Sample Streams	in enable avg_count avg_round_bool	out	out = moving_average[N] Division by average count is optimized by multiplying by reciprocal of the average count.	in: Bit width (= out Bit width).	None.	Enable input (enable). Average count input (avg_count). Reciprocal of the average Count Fixed-point. exponentiont input (exp_count_fixed).
	Moving Average Shift		●	●	●	●	Hardware Software	Sample Streams	in enable	out	out = moving_average[N] Division by average count is optimized by right-shift in order to avoid multiplication. Only powers of 2 average counts supported.	in: Bit width (= out Bit width). Average count (LOG2(N)/32).	None.	Enable input (enable).
	Delay Buffer		●	●	●	●	Hardware Software	Sample Streams	in	out	out = (in [delayed by specified number of samples])	in/bit width (= out Bit width). Sample: Delay.	None.	None.
	Decimator	●	●	●	●	●	Hardware Software	Sample Streams	in	out	out = (in decimated by a specific ratio[N]).	in: Bit width (= out Bit width). None (N).	None.	None.
	FIR Filter	●	●	●	●	●	Hardware Software	Sample Streams	in coeff_set	out	out = Convolution of (sound F.I.R. Coefficients.	in: Bit width (= out Bit width). Number of Taps. Filter Type (Single/Multi- MAC, Systolic/Transposed). MAC type (Auto-map, select MAC architectures, explicit instantiation of FPGA DSP slices (Use DSP48E1).	Fifo Coefficients loaded into Coeff_set. Number of taps. coeff_set: AXI4-Lite interface.	None.
	Mixer-Modulator		●	●	●	●	Hardware Software	Sample Streams	in1 (Q-Line) in2 (LO)	out	out = in1 * in2 When functionality: out = in1 * (-1 * in2) Modulator functionality:	out: Bit width. in1: Bit width. in2: Bit width. out: Bit width. Mixer or Modulator selection.	None.	None.
	CIC Decimating Filter				●	●	Hardware Software	Sample Streams	in	out	out = (a integrated series N stages, decimated buy rate R and then taken through N Comb (differentiated) stages.	in: Bit width (= out Bit width). Number of Stages (N). Rate (R).	None.	None.
Data Operators	Merge	●	●	●	●	●	Hardware	Constant Axis Simple Streams	in_ln in_hh	out	out = in_ln[:32b] merged with in_hh[:32bits]	in_ln: Bit width. in_hh: Bit width.	None.	None.
	Slice	●	●	●	●	●	Hardware	Constant Axis Sample Streams	in	out	out = in[start_bit : end_bit]	in/bit width. Start bit index of the slice from. Its last bit index of the slice from. Its	None.	None.
	Multiplexer	●	●	●	●	●	Hardware	Constant Axis Simple Streams	in0 in1 in2 ... inN sel	out	out = in* selected based on the value of sel input.	in*: Bit width (= out Bit width). Number of input channels. Number of input ports per or hold or free running.	None.	None.
	Packer	●	●	●	●	●	Hardware	Sample Streams	in	out	out = (a packed by samples of a specified width until specified number of samples per pack.	in/bit width. Bit width of each sample in the pack. Number of samples per pack.	None.	None.
Waveform Generators Sounds	Raster NCO - Pre Config					●	Hardware	NA	inc_out inc_depth inc_shift phase	out	out = Replay of the samples stored in the LUT, stepping a specific pitch, until wrap-around.	out: Bit width (Max 32 bits). Maximum LUT depth required. Block RAM type.	Sample values to be stored in the LUT using the inc_out AXI4-Lite interface.	Enable input (enable). LUT Depth input (inc_depth). LUT Replay-cursor pitch input (inc_shift). Start phase input (phase).
	Raster NCO-IQ - Pre Config					●	Hardware	NA	inc_out inc_depth inc_shift phase	out_ph out_qu	out_ph = Replay of the samples stored in the LUT, stepping a specific pitch, with wrap-around. out_qu = Same as above, but skipping 1/4h of the samples to the start for full phase shift.	out: Bit width (Max 32 bits). Maximum LUT depth required. Block RAM type.	Sample values to be stored in the LUT using the inc_out AXI4-Lite interface.	Enable input (enable). LUT Depth input (inc_depth). LUT Replay-cursor pitch input (inc_shift). Start phase input (phase).
	Raster NCO - Auto-Init Config					●	Hardware	NA	inc_out enable inc_depth inc_shift phase	out	out = Replay of the samples stored in the LUT, stepping a specific pitch, with wrap-around. Auto-initialization of the LUT with the values of the samples in the case for full phase shift.	out: Bit width (Max 32 bits). Maximum LUT depth required. Block RAM type.	Sample values to be stored in the LUT using the inc_out AXI4-Lite interface.	Enable input (enable). LUT Depth input (inc_depth). LUT Replay-cursor pitch input (inc_shift). Start phase input (phase).
	Raster NCO-IQ - Auto-Init Config		●	●	●	●	Hardware	NA	inc_out enable inc_depth inc_shift phase	out_ph out_qu	out_ph = Replay of the samples stored in the LUT, stepping a specific pitch, with wrap-around. out_qu = Same as above, but skipping 1/4h of the samples to the start for full phase shift.	out: Bit width (Max 32 bits). Maximum LUT depth required. Block RAM type.	Sample values to be stored in the LUT using the inc_out AXI4-Lite interface.	Enable input (enable). LUT Depth input (inc_depth). LUT Replay-cursor pitch input (inc_shift). Start phase input (phase).
	NCO - IIR-based tuned					●	Hardware	NA	enable sine cosine	out	out_sin = Continously computed values of Sin(2πft/fs), where f/fs is a per-defined increment to use for the sine output samples. out_cos = Continously computed values of Cos(2πft/fs), where f/fs is a per-defined increment to use for the cosine output samples.	out: Bit width. Sine/Cos/Sin/Cos bit-width.	None.	Enable input (enable). Sine/Cos fixed-point fraction input (sine). Sine/Cos fixed-point fraction input (cosine).
	NCO - DAI Algorithm	●	●	●	●	●	Hardware	NA	NA	out	out = Sine wave samples for specified frequency, sample rate and sample_width.	out: Bit width. Frequency. Sample Rate. Frequency res.		
	Linear Waveform Generator				●	●	Hardware	NA	arc trigger	out num_out	out = Samples corresponding to a linear waveform, per the programmed waveform parameters. num_out = A binary (square) wave with its rising edge coinciding with the start of rise edge of the arc waveform and its falling edge coinciding with the start of fall edge of the arc waveform.	out: Bit width.	Various parameters (32b fixed-point) of the waveform to be generated: - Rise time (Clock count) - Fall time (Clock count) - Rise value (Value) - Fall value (Value) - High duration (Clock count) - Low duration (Clock count) - High level (Value) - Low level (Value) Waveform enable or abort command.	The trigger input can be optionally driven from a Memory Mapped Register bit. If not driven by any source, this input need be tied low.
Data Servers	Constant Driver	●	●	●	●	●	Hardware Software	NA	NA	out	out = Constant value on a data bus (hardware) on demand (software) via a buffer (software) of specified width. (Fixed this data bus to 0S Data Feed to drive a simple stream in hardware).	out: Bit width. out: Value to be driven.	None.	None.
	TIE-LOW Driver	●	●	●	●	●	Hardware Software	NA	NA	out	out = Tie-Low (value = 0) driven for either data bus or signal network input.	None - Automatically adjusts to the properties of the connected network.	None.	None.
	TIE-HIGH Driver	●	●	●	●	●	Hardware Software	NA	NA	out	out = Tie-High (value = 1) driven for either data bus or signal network input.	None - Automatically adjusts to the properties of the connected network.	None.	None.
I/O Components	Clever				●	●	Hardware	NA	arc trigger	out	out = A binary waveform corresponding to the programmed parameters.	None.	Output waveform type - Periodic or pulse mode - High and low durations in clock counts - Optional waveform output - Optional second delay and pulse width - Number of pulses in burst of pulse mode - Optional trigger bit asserted in case of pulse burst mode - External trigger bound or software triggered in case of pulse mode - Inverse waveform high or low - Waveform enable or abort command.	The trigger input can be optionally driven from a Memory Mapped Register bit. If not driven by any source, this input need be tied low.
	Sample Gate	●	●	●	●	●	Hardware	Sample Streams	in enable reset trigger count	out	out = A specified (gate count) number of in samples passed through, each time the gate is triggered	in: Bit width (= out Bit width).	None.	The Gate Function input (enable), Gate Input (arm), Trigger input (trigger) and Reset input (reset) can be driven by the software from an MMIO bit. The Count input can also be driven by constant values.
	MMIO Write - [Read/Write]		●	●	●	●	Hardware	NA	NA	out	out = (A value programmed using AXI4-Lite interface on a data bus of specified bit width. bit-width = (start_bit - end_bit + 1	out: Start bit index. out: End bit index. MMIO offset address.	This component represents an AXI4-Lite Memory-Mapped Register.	Value to be driven by the MMIO.
	MMIO Read - [Read Only]		●	●	●	●	Hardware	NA	in valid	NA	The data bus value at the (in) pin is captured in the MMIO when ever the valid (pin) is 'high.	in/Bit width. MMIO offset address.	This component represents an AXI4-Lite Memory-Mapped Register.	Value can be read from the MMIO. Valid (in) pin can be driven by the MMIO. Note: It can also be tied high using a 1'b1 HDL component, if the input value need be captured continuously.
	AXI4-Lite Switch					●	Hardware	AXI4-Lite Programming Interface (+ (AXI4-Lite))	sel0 sel1 selN (AXI4-Lite)	out0 out1 outN	This AXI4-Lite 'slave' input is routed to one of the specified number of 'out' AXI4-Lite master ports, depending upon the value of the specified range (sel_msb:sel_lsb) of the decoded byte address.	Number of output master ports. Select address range MSB (sel_msb). Select address range LSB (sel_lsb).	None.	None.
	ASYNC FIFO					●	Hardware	Sample Streams	in	out	out = (a taken through a max 16-deep FIFO. This FIFO can be configured as either synchronous or asynchronous (with the 'out' clock connected to a different clock domain. Macro's used for clock conversion.	in: Bit width (= out Bit width). FIFO Depth (2^N+1 -1). Input side clock (cpu_clk or ser_clk).	None.	None.
Board Support Components	PMOD Input				●	●	Hardware	NA	NA	pmodA pmodB pmodC	The selected number (up to 8) of PMOD inputs are routed to the Zynq PS. Related interrupt control functions and ISR templates are also generated.	Number of PMOD input pins to be accessed.	None.	None.
	PMOD Output				●	●	Hardware	NA	pmodA pmodB pmodC	NA	The selected number (up to 8) of PMOD outputs are routed to connector pins J1[1, 2, 3, 4], J2[1, 2, 3, 4], J3[1, 2, 3, 4], and J4[1,2,3,4] of the ZedBoard respectively. The required IO map connections are also generated.	Number of PMOD output pins to be driven.	None.	The pmod values can be optionally driven from Memory Mapped Register bits, if the XFarmos version supports MMR registers.
	LED Output	●	●	●	●	●	Hardware	NA	led0 led1 led2	NA	The selected number (up to 8) of LED driven pins are routed to the ZedBoard LEDs. The required IO map connections are also generated.	Number of LEDs to be driven.	None.	The LED values can be optionally driven from Memory Mapped Register bits, or the XFarmos version supports MMR registers.
	SWITCH Input	●	●	●	●	●	Hardware	NA	NA	switch0 switch1 switch2	The selected number (up to 8) of Switch inputs are routed from the ZedBoard switches. The required IO map connections are also generated.	Number of Dx3 switches to be accessed.	None.	None.
	IRQ Input				●	●	Hardware	NA	NA	irq0 irq1 irq2	The selected number (up to 8) of interrupts are routed to the Zynq PS. Related interrupt control functions and ISR templates are also generated.	Number of IRQ pins needed.	None.	None.
	Zed Platform Integration	●	●	●	●	●	Hardware	NA	NA	NA	Simulation of Zed Platform during the system simulation. This includes the Zynq PS simulated and Zynq specific infrastructure components like the DMAs, IRQ connectivity, Clock/Reset modules, etc. Also generates the Zynq PS configuration executable TCL/Command line and C++ Drivers/Initialization code for bit-stream and off-generation.	None.	xil/AXI GMA Parameters.	The XFarmos generated IP, can include the Zynq PS configuration TCL script, to be used to drive external memory by connecting them in the 'Xilinx Block Design' environment. This is to be used in conjunction with the Zynq PS configuration TCL, explicitly for versions that support MMR registers.
	AD9467 ADC Interface	●	●	●	●	●	Hardware	NA	NA	NA	Instantiation of this component in the schematic results in integration of the ADC Interface IPs, Clocking and Reset modules, Zynq PS configuration, along with the required intermediate TCL/Command files, and C++ Drivers/Initialization code for bit-stream and off-generation.	None.	AD ADC Parameters.	None.
External Interface Components	Multi-Ch AXIS Out Interface					●	Hardware	Sample Streams	in sof eol vld ... (Internal Stream Signals) headerA headerB headerC (All AXIS) data (For AXIS)	out (AXIS) m_axi	out = (A (single channel mode) or AP (multi-channel mode) connected to AXIS Stream format, with the 'in' input connected to a source that supports the type of Frame, the sample blocks, interleaved according to the specified Frame format and the Frame header, are output to the AXIS Master port. The Frame format is defined by the headers. In multi-channel mode, the eA_sof output is used to indicate the start of channel 'eA's active for interleaving.	in: Bit width. out: AXIS Bit width.	None.	Headers input (headerA), Headers input (headerB), AXIS TID value (dest) and AXIS TDEST value (data) can be driven from AXIMLite/ the XFarmos service supports MMR registers/ or constants.
	Single-Ch AXIS Out Interface	●	●	●	●	●	Hardware	Sample Streams	in (AXI)	out (AXI)	out = (A sample packed, synchronized to the internal clock and converted to AXIS Stream format. to be driven to external/ modules.	in: sample Bit width. out: AXIS data Bit width. Number of Samples to be packed per output data word. Input side and Output side clock selection. AXIS TID and TDEST to be driven.	None.	None.
	Software Frame In Interface	●	●	●	●	●	Hardware	Sample Streams	in	out (Sample Buffer)	out = (A sample input placed in a software 'AXI4' buffer.	in: sample Bit width.	None.	None.
	Hardware to Software Interface	●	●	●	●	●	Interface	Sample Streams	in (AXI)	out (Sample Buffer)	out = (A sample packed, synchronized to the external clock, connected to AXIS Stream format. to be driven to software/ input buffer using an AXI Master interface with burst support.	in: sample Bit width. out: AXIS data Bit width. Number of Samples to be packed per output data word. Input side and Output side clock selection. AXIS Block Size. AXIS Burst Type. AXIS Burst Length.	None.	None.
	AXIS to I2S connector (I2s)	●	●	●	●	●	Hardware	AXIS Stream	in (AXIS)	out (I2S)	out = (A connected to Logicnet Semiconductors proprietary internal Stream format (I2S).	out: Bit width. AXIMLI interface or not (if AXIMLI is not selected, the I2S Clocks are automatically inverted to remove the clock from AXIS should be I2S based).	None.	None.
	I2S to AXIS connector (I2s)	●	●	●	●	●	Hardware	AXIS Stream	in (I2S)	out (AXIS)	out = (A I2S connected to a 32b AXIS stream.	in/bit width.	None.	None.
	I2S to AXIS connector (I2s)	●	●	●	●	●	Hardware	AXIS Stream	in (I2S)	out (AXIS)	out = (A I2S connected to a 32b AXIS stream.	in/bit width.	None.	None.
	AXIS Frame In Interface					●	Hardware	AXI4	in (AXIS) sof (AXIS Line)	out out0 out1 (external Stream Signals)	out = (A connected to Logicnet Semiconductors proprietary internal Stream format (I2S). to be driven to external/ modules. sof = (a be multiplexed to multiple I2S streams, in case of multi-channel mode.	in (AXIS) Bit width. out: I2S Bit width.	Various parameters of the sof (start of frame) can be programmed directly/ thru an AXI4-Lite interface.	None.

	Software Frame Out Interface					■	Software	Sample Buffer	In	**out**	**out :** An sample output buffer from software partition placed in the output buffers with the same function for transmit to the hardware partition.	**in** sample bits width.	None.	None.
Topology Components	**AXI4-Lite Splitter** (32b)	■	■	■	■	■	Hardware	AXI4 Stream	In (AXI4-Lite)	**out1** (AXI4-Lite) **out2** (AXI4-Lite)	**out1 - out2 :** Memorymapped interface organizes AXI4-Lite Master. Typically used to program slave components with exactly the same behavior.	None.	None.	None.
	I2S Splitter	■	■	■	■	■	Hardware	I2S (Legacyand Xilinxinterfaces, Proprietary, Secure Formats)	In (I2S)	**out1** (I2S) **out2** (I2S)	**out1 - out2 :** An I2S sp32 into two I2S channels with 32bit resolution. Typically used to program slave. FIFOs are provided on each output to avoid input stall in case of short non-stalls(Valid) sendStream at the outputs.	None.	None.	None.
	I2S Fractional Stream Splitter					■	Hardware	I2S (Legacyand Xilinxinterfaces, Proprietary, Secure Formats)	In (I2S: Integer+ Fraction)	**out1** (I2S Integer+ Fraction) **out2** (I2S Integer+ Fraction)	**out1 - out2 :** An I2S IntegerFraction stream sp1n into two I2S IntegerFraction streams with full throttling support on each. A small synchronous FIFO is provided on each output to avoid input stall in case of short non-stalls(Valid) sendStream at the outputs.	None.	None.	None.
	I2S Data Feed	■	■	■	■	■	Hardware	Data files	**in** (Data files) **valid**	**out** (I2S)	**in :** the data files converted to the Jupyter's memory space using Pynq's allocate API. The I2S, **out** will be driven high when ever the **valid** input signal is held high.	**in** Bits width (= **out** Bits width).	None.	The **valid** input can be optionally driven from a Memory Mapped Register bit. If **valid** is held high, the I2S, **out** will be held high as above. The I2S, **out** will continuously
	I2S Data Tap	■	■	■	■	■	Hardware	I2S (Legacyand Xilinxinterfaces, Proprietary, Secure Formats)	In (I2S)	**out** (Data files) **valid**	**out :** The data will be driven on the I2S input transfer stream. The **valid** output signal is driven high when ever the I2S input will is high.	**in** Bits width (= **out** Bits width).	None.	The **valid** can be read using a read only Memory Mapped Register, and then used as the **valid** output signal of this component.
	Integer to Fraction converter					■	Hardware	I2S (Legacyand Xilinxinterfaces, Proprietary, Secure Formats)	In (Integer)	**out** (Integer + Fraction)	**out :** An Integer only stream converted (up) to an Integer+Fraction pair stream of specified sfp format. The resultant **out** stream can be fed to the Fixed- sP Functions' AXI4stream operands.	**in** Bits width. **out** Integer part Bits width. **out** Fractional part Bits width.	None.	None.
	Fraction to Integer converter					■	Hardware	I2S (Legacyand Xilinxinterfaces, Proprietary, Secure Formats)	In (Integer + Fraction)	**out** (Integer)	**out :** An Integer + Fraction stream of specified sfp format. The resultant **out** stream can be fed to the inputs of Integer only Arithmetic operators.	**in** Integer part Bits width **in** Fractional part Bits width. **out** Bits width	None.	None.
	Integer-Fraction Separator					■	Hardware	I2S (Legacyand Xilinxinterfaces, Proprietary, Secure Formats)	In (Integer + Fraction)	**in_int** (Integer) **frac_out** (Integer)	**in_int :** Integer part of In. **frac_out :** Fractional part of In.	**in** Integer part Bits width (= **in_int** Bit width) **in** Fraction part Bits width (= **frac_out** Bit width).	None.	None.
	Integer-Fraction Merge					■	Hardware	I2S (Legacyand Xilinxinterfaces, Proprietary, Secure Formats)	**in_in** (Integer) **frac_in** (Fraction)	**out** (Integer + Fraction)	**out :** Integer + Fraction stream with **in_in** as the Integer part and **frac_in** as the fractional part.	**in** Integer part Bits width (= **in_int** Bit width) **in** Fractional part Bits width (= **frac_out** Bit width).	None.	None.
Host Interface	**Xsa Connect Terminator**	■	■	■	■	■	Hardware Software	Xsa	NA	NA	Unmerit unused outputs to this component.	None.	None.	None.
	Schematic Input Pin	■	■	■	■	■	Hardware Software	Xsa	NA	NA	Hardware of Software Schematic Input Pin	None.	None.	None.
	Schematic Output Pin	■	■	■	■	■	Hardware Software	Xsa	NA	NA	Hardware of Software Schematic Output Pin	None.	None.	None.