Using the Floating-Point Converter Example

AN039-v1.0
February 2022
www.efinixinc.com

Contents

Introduction

Binary floating-point arithmetic is commonly used to represent real numbers on computers because it can represent a wide range of numbers. When implemented in FPGAs, however, floating-point can use considerable logic resources. Therefore, designers use fixed-point because of its better performance and lesser resource utilization. Essentially, a floating-point number is converted to a fixed-point before an operation. The result is then converted back to a floating-point for compatibility with other systems.

Efinix provides example design that demonstrates the conversion between floating-point and fixed-point numbers. The design targets the Titanium Ti60 F225 Development Board and Trion® T120 BGA576 Development Board. The example design includes Efinity Virtual I/O Debugger core for customizing and monitoring the design.

Features

- Floating-point to fixed-point conversion
- Fixed-point to floating-point conversion
- Binary floating-point complies with IEEE 754 standard
- Supports floating-point of single-precision (32-bit) and double-precision (64-bit)
- Supports all floating-point numbers
- Supports a fixed-point width of 3 to 64 bits
- Optional asynchronous reset and clock enable ports for both float-to-fixed and fixed-to-float converters
- Optional output ports indicating overflow, underflow, not-a-number (NaN), infinity, zero, and denormalized number in float-to-fixed converter
- Fully-pipelined design with six pipeline stages
- RTL design in Verilog HDL
- Round to nearest, ties away from zero rounding rule

Trion Performance and Resource Utilization

FPGA	Mode / Width	Logic Utilization	Memory Blocks	Multipliers	f_{MAX} (Mhz)	Efinity Version
T120 F576 C4	Double-precision to 32.32 fixed-point	662	0	0	207.4	2021.2
	Double-precision to 16.16 fixed-point	358	0	0	254.4	
	Single-precision to 32.32 fixed-point	518	0	0	230.5	
	Single-precision to 16.16 fixed-point	330	0	0	266.6	
	32.32 fixed-point to double-precision	660	0	0	204.1	
	16.16 fixed-point to double-precision	313	0	0	245.0	
	32.32 fixed-point to single-precision	468	0	0	210.3	
	16.16 fixed-point to single-precision	295	0	0	220.8	

Titanium Performance and Resource Utilization

FPGA	Mode / Width	Logic and Adders	Memory Blocks	DSP48 Blocks	f_{MAX} (Mhz)	Efinity Version
Ti60 F225 C4	Double-precision to 32.32 fixed-point	662	0	0	537.9	2021.2
	Double-precision to 16.16 fixed-point	358	0	0	712.0	
	Single-precision to 32.32 fixed-point	518	0	0	680.1	
	Single-precision to 16.16 fixed-point	330	0	0	827.1	
	32.32 fixed-point to double-precision	660	0	0	547.5	
	16.16 fixed-point to double-precision	313	0	0	690.2	
	32.32 fixed-point to single-precision	468	0	0	550.5	
	16.16 fixed-point to single-precision	295	0	0	730.6	

Functional Description

The reference design contains two modules, a floating-point to fixed-point converter and a fixed-point to floating-point converter.

Float-to-Fixed Conversion

The float-to-fixed converter converts a floating-point number into a fixed-point number. The converter accepts all floating-point numbers as input. The converter indicates special cases using output signals as listed in **Table 1: Float-to-Fixed Converter Ports** on page 7.
The rounding behavior of the converter follows the "round to nearest, ties away from zero" rule. The source file for the float-to-fixed converter, **float_to_fixed.v**, is in the **floating-point-converter-<version>/rtl/** directory.

Fixed-to-Float Conversion

The fixed-to-float converter converts a fixed-point number to a floating-point number. The converter uses the "round to nearest, ties away from zero" rounding rule. The source file for the float-to-fixed converter, **fixed_to_float.v**, is in the **floating-point-converter-<version>/rtl/** directory.

Number Formats

Floating-Point Format

The example design supports single-precision and double-precision floating-point formats defined by the IEEE 754 standard. The data width of the single-precision format is 32-bit, including 1 sign bit, an 8-bit exponent, and a 23-bit mantissa. For double-precision, the data width of the exponent and mantissa is extended to 11 bits and 52 bits, respectively.

Fixed-Point Format

A fixed-point number comprises an integer and a fractional part. The integer uses two's complement representation, but the fractional part is always unsigned. If a 48-bit fixed-point number has 16 bits in the integer part and 32 bits in the fractional part, the number format can be specified using 16.32. The supported total number of bits in this reference design is 3 to 64.

The following formula gives the value of a fixed-point number:

$$value = integer + fraction/(2^{width\ of\ fraction})$$

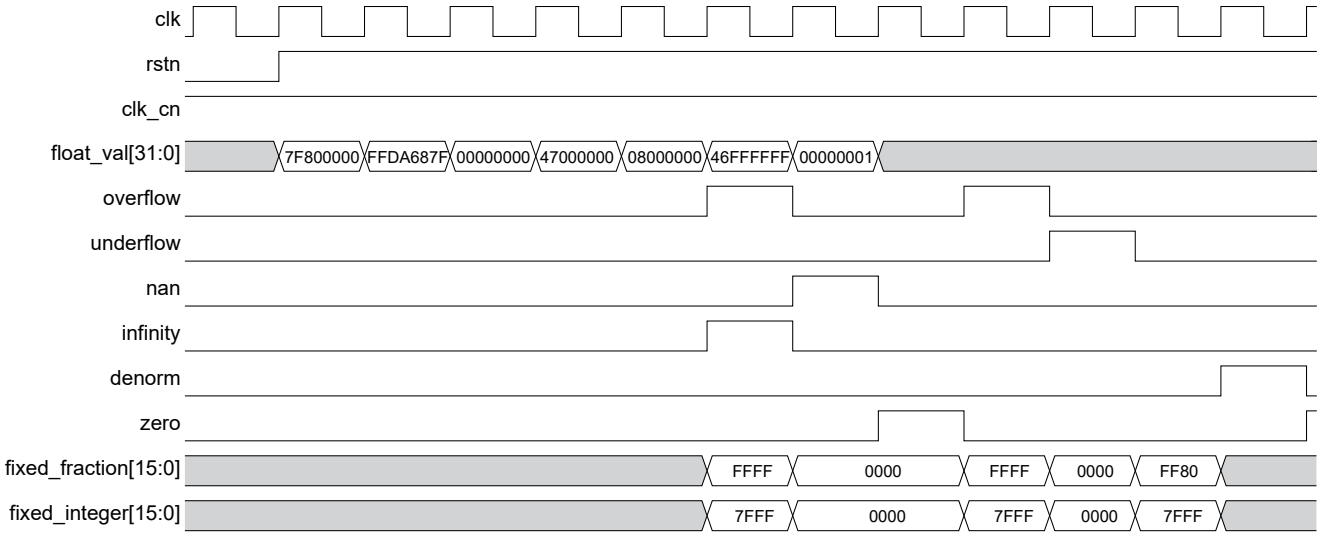
For example, for a 16.32 fixed-point number, if the data of the integer part and the fractional part are 0xFFE2 and 0x4CCCCCCD, respectively, the value of this fixed-point number is -29.7.

Operations

Float-to-Fixed Converter Operation Example

The float-to-fixed converter is a fully pipelined design that takes six cycles to convert a number. In the following figure, `0x7f800000` is assigned to input port `float_val` using a non-blocking assignment. The data appears at the input port after one clock cycle. The conversion results are available at output ports after 6 clock cycles. Because the data `0x7f800000` represents a positive infinity in a single-precision format, the converter asserts the `overflow` and `infinity` signals, and the fixed-point output is set to the maximum.

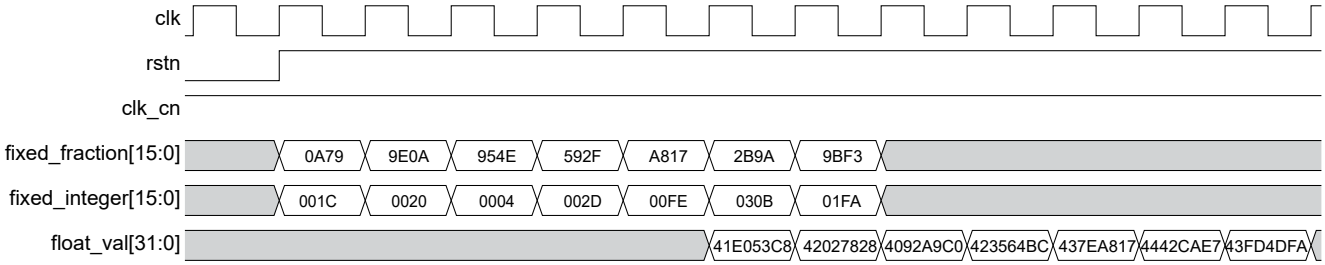
Figure 1: Float-to-Fixed Conversion Timing Diagram Example



Fixed-to-Float Converter Operation Example

The fixed-point to floating-point converter is a fully pipelined design with six pipeline stages. In the following figure, a fixed-point number is assigned to the input ports using a non-blocking assignment. Six cycles later, the result, `0x41e053C8`, is available at the output port. The converter accepts a fixed-point number every cycle.

Figure 2: Fixed-to-Float Conversion Timing Diagram Example



Ports

Table 1: Float-to-Fixed Converter Ports

Port	Direction	Description
clk	Input	System clock.
rstn	Input	Asynchronous active low reset.
clk_en	Input	Active high clock enable.
float_val	Input	Floating-point number input.
overflow	Output	Overflow flag. Indicates that the floating-point input exceeds the range of fixed-point number output. This signal goes high when: • Input value is greater than or equals to 2^{INT_WID-1} • Input value is smaller than -2^{INT_WID-1} • Input number is infinity. Depending on the sign bit, the value of fixed-point output becomes either maximum or minimum when the overflow flag is set.
underflow	Output	Underflow flag. Indicates that the floating-point input is too small and cannot be represented by the fixed-point output. This signal goes high when: BIAS minus FRA_WID is higher than or equal to the exponent part of the floating-point input. BIAS is an exponent bias defined in IEEE 754. The bias in single precision is +127, and +1023 for double precision. The fixed-point output becomes all zero when underflow flag is set.
nan	Output	This signal goes high when the floating-point input is NaN. The fixed-point output become all zero when this is set.
infinity	Output	This signal goes high when the floating-point input is positive infinity or negative infinity. The overflow flag is also set at the same time. Depending on the sign bit, value of the fixed-point output become either the maximum or minimum when this signal is set.
denorm	Output	This signal goes high when the floating-point input is a denormalized number. Fixed-point output become all zero when this is set.
zero	Output	This signal goes high when the floating-point input is exactly zero.
fixed_fraction	Output	Fractional part of the fixed-point output. The data is unsigned.
fixed_integer	Output	Integer part of the fixed-point output. Two's complement representation is used.

Table 2: Fixed-to-Float Converter Ports

Port	Direction	Description
clk	Input	System clock.
rstn	Input	Asynchronous active low reset.
clk_en	Input	Active high clock enable.
float_val	Input	Floating-point number output.
fixed_fraction	Input	Fractional part of the fixed-point input. The data is unsigned.
fixed_integer	Output	Integer part of the fixed-point input. Two's complement representation is used.

Customizing the Floating-Point Converter

You can further customize the modules using the following parameters in the source files.

Table 3: Float-to-Fixed Converter Parameters

Parameter	Type	Options	Description
FLOAT_FMT	String	float, double	Sets the floating-point format.
INT_WID	Integer	1 - 63	The width of integer part of fixed-point output. The total width of integer and fractional part should greater than 2.
FRA_WID	Integer	1 - 63	The width of fractional part of fixed-point output. The total width of integer and fractional part should be greater than 2.

Table 4: Fixed-to-Float Converter Parameters

Parameter	Type	Options	Description
FLOAT_FMT	String	float, double	Sets the floating-point format.
INT_WID	Integer	1 - 63	The width of integer part of fixed-point input. The total width of integer and fractional part should greater than 2.
FRA_WID	Integer	1 - 63	The width of fractional part of fixed-point input. The total width of integer and fractional part should be greater than 2.

Required Hardware

The example designs use the following hardware:

- Titanium Ti60 F225 Development Board or Trion® T120 BGA576 Development Board
- 1 USB type-C cable (Ti60 F225 board) or USB micro type-B cable (T120 BGA576 board)
- Universal AC to DC power adapter

Set Up the Hardware

The example design does not require additional hardware to be set up with the development board other than the power supply and the USB cable connection to the computer. However, Efinix recommends that you use the default jumper settings for the development boards.

Figure 3: Titanium Ti60 F225 Development Board Hardware Setup

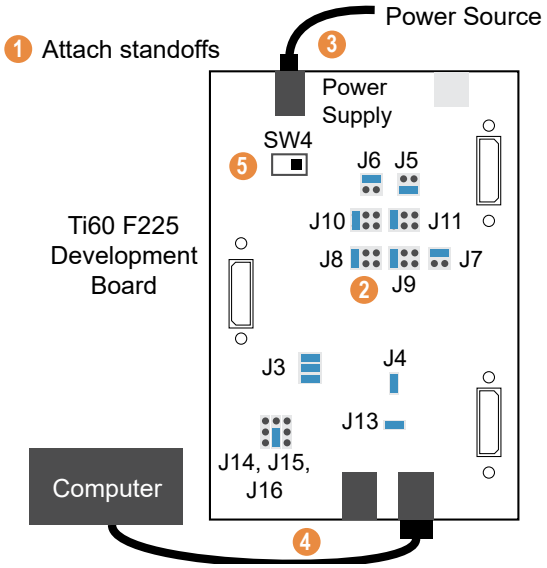
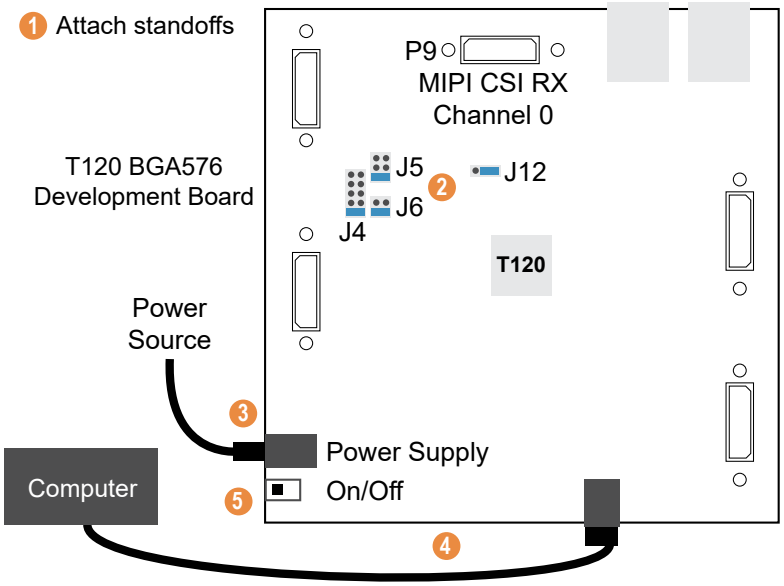


Figure 4: Trion® T120 BGA576 Development Board Hardware Setup



1. Attach standoffs to the board if you have not already done so.
2. Depending your board, connect the following jumpers:

Table 5: Titanium Ti60 F225 Development Board Jumper Settings

Header	Short Pins
J3	1 and 2, 3 and 4, 5 and 6
J4	1 and 2
J5	1 and 2
J6	3 and 4
J7	3 and 4
J8	5 and 6
J9	5 and 6
J10	5 and 6
J11	5 and 6
J13	1 and 2
J14	Unconnected
J15	1 and 2
J16	Unconnected

Table 6: Trion® T120 BGA576 Development Board Jumper Settings

Header	Short Pins
J4	9 and 10
J5	6 and 6
J6	3 and 4
J12	2 and 3

3. Ensure the board power switch is turned off, then connect the 12 V power cable to the board connector and to a power source.
4. Connect the following USB header to the USB port of your computer.
   - *J12*—Titanium Ti60 F225 Development Board
   - *J20*—Trion® T120 BGA576 Development Board
5. Turn on the board's power switch.

Program the Development Board

The Efinix development boards are pre-loaded with a demonstration design. To use the Floating-Point Converter example design, you must program the design into the board.

1. Turn on the development board.
2. Download the example design file, **floating_point_converter-v<version>.zip** from the Support Center.
3. Unzip the file into your working directory.
4. Open the project in the Efinity software and review it.
5. Use the Efinity® Debugger to download the bitstream file to your board. The example includes bitstream files for:

Bitstream File	Module	Development Board
../fixed2float_ti/fixed2float_ti.bit	Fixed-to-float	Titanium Ti60 F225 Development Board
../float2fixed_ti/float2fixed_ti.bit	Float-to-fixed	Titanium Ti60 F225 Development Board
../fixed2float_ti/fixed2float.bit	Fixed-to-float	Trion® T120 BGA576 Development Board
../float2fixed_ti/float2fixed.bit	Float-to-fixed	Trion® T120 BGA576 Development Board

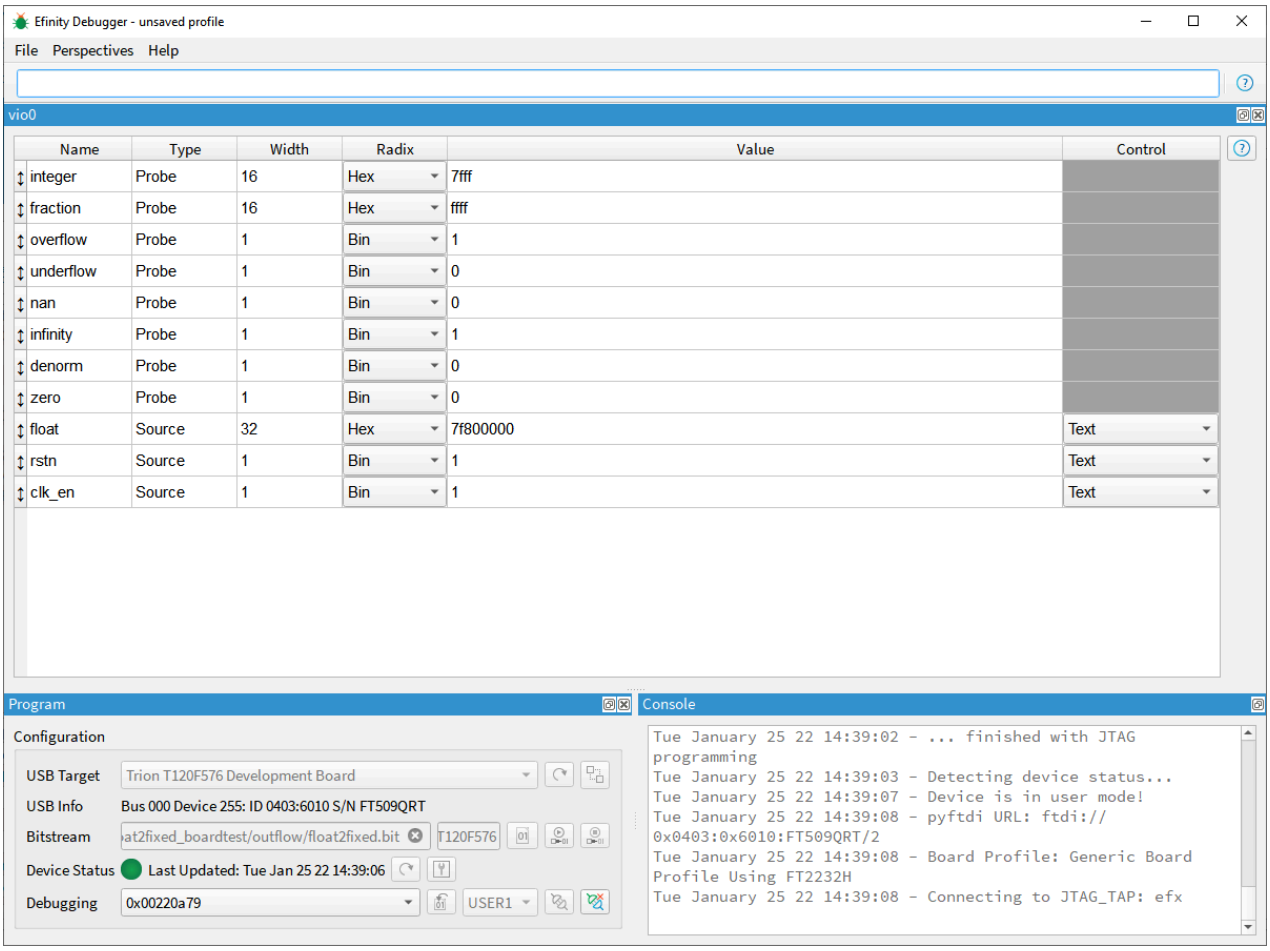
Learn more: Instructions on how to use the Efinity® software is available in the Support Center.

View the Design in the Debugger

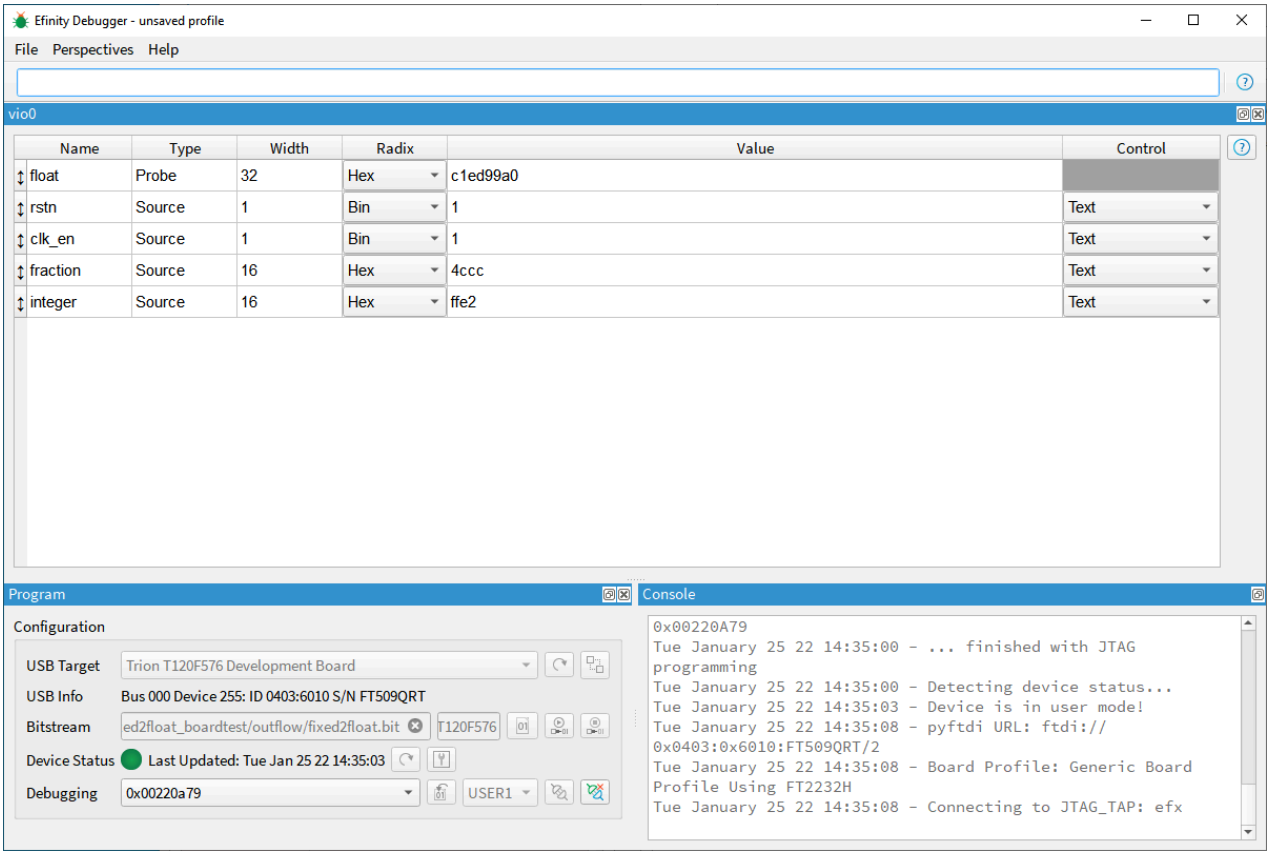
All input and output ports except for the system clock, are connected to a Virtual I/O Debugger. You can enter any number using the Virtual I/O Debugger to validate the results.

Learn more: Refer to the Debug Perspective: Virtual I/O section of the Efinity Software User Guide for more information on how to use the Virtual I/O Debugger.

The following figure illustrates the Virtual I/O Debugger example of a float-to-fixed converter operation. Start the design by asserting the clock enable signal (`clk_en`) and deasserting the reset signal (`rstn`). `0x7f800000` value is added to the `float` source which means a positive infinity in the single-precision floating-point format. The converter asserts the `overflow` and `infinity` signals, and the fixed-point output is set to the maximum.



The following figure illustrates the Virtual I/O Debugger example of a fixed-to-float converter operation. You start the design by asserting the clock enable signal (`clk_en`) and deasserting the reset signal (`rstn`). The value of the fixed-point number input in the figure is $-30 + 19660/(2^{16}) \approx -29.700$. It is converted into a single-precision floating-point number with data 0xc1ed99a0. The decimal value for the floating-point number is approximately -29.7, according to the IEEE 754 standard.



Revision History

Table 7: Revision History

Date	Version	Description
February 2022	1.0	Initial release.