



VLSI Design & Consultancy

DATASHEET

Floating Converters

Version 1.0

1 Overview

The Think-Silicon *Floating Point Converters* is a web configurable generator for standard IEEE754 compliant single precision Floating Point to Integer Converters. The generated units support proper round to nearest.

2 Floating Point Numbers

The format of the IEEE 754 single precision floating point numbers is shown in the Table 2-1.

Table 2-1 IEEE single precision floating point numbers format

BITS	NAME	DESCRIPTION
31	sign	This is the sign bit. 0 signifies a positive number and a 1 a negative number
30:23	exponent	This number defines the power of 2 that is multiplied by the fractional part. It also defines if the number is normal.
22:0	fraction	These are fractional bits of the number.

According to IEEE754 standard a floating point single precision number is defined as follows:

$$number = -1^{sign} \times 2^{(127-exponent)} \times 1.(fraction)$$

There are also special bit patterns that define +ve and -ve ∞ (infinity) as well as positive and negative \emptyset (zero). Those patterns as are also supported and handled according to the standard.

3 Signed Integer to floating point (sint2fp)

3.1 sint2fp Block Diagram

The *sint2fp* module (shown in Figure 3-1) converts signed 32-bit integer to single precision floating point numbers. The module has one input port and an output port and is fully combinatorial.

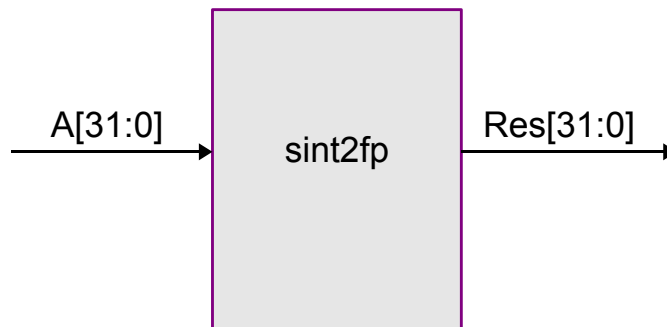


Figure 3-1 The *sint2fp* block

3.2 sint2fp Port Interface

The ports of the module are listed in Table 3-1.

Table 3-1 *sint2fp* module port list

PORT	TYPE	DESCRIPTION
A[31:0]	input	Signed 32bit integer in 2's complement
RES[31:0]	output	IEEE754 Single precision floating point number

3.3 Interfacing sint2fp

The design is fully combinatorial, thus should you require to pipeline the unit, you are advised to do so at the synthesis level, using a balanced registers technique that will provide better results for your target technology.

3.4 sint2fp Example Vectors

Table 3-2 shows *sint2fp* example vectors

Table 3-2 *sint2fp* example vectors

00000000	00000000 (0.0)	Zero returns a positive 0 fp
00000001	3f800000 (1.0)	
00000002	40000000 (2.0)	
ffffffff	bf800000 (-1.0)	
ffffffffffe	C0000000 (-2.0)	
7fffffff	4f000000 (2147483648)	Rounded
80000000	cf000000 (-2147483648)	

4 Unsigned Integer to floating point (uint2fp)

4.1 uint2fp Block Diagram

The *uint2fp* module (Figure 4-1) converts unsigned 32-bit integer to single precision floating point numbers. The module has one input port and an output port and is fully combinatorial.

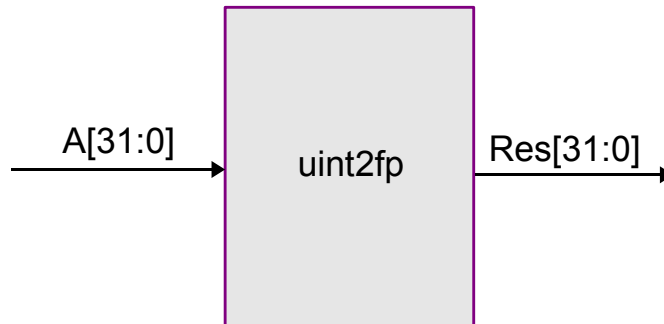


Figure 4-1 *uint2fp* Block Diagram

4.2 uint2fp Port Interface

The ports of the module are listed in Table 4-1.

Table 4-1 *uint2fp* Port Interface

PORT	TYPE	DESCRIPTION
A[31:0]	Input	Unsigned 32bit integer
RES[31:0]	Output	IEEE 754 Single precision floating point number

4.3 Interfacing uint2fp

The design is fully combinatorial, thus should you require to pipeline the unit, you are advised to do so at the synthesis level, using a balanced registers technique that will provide better results for your target technology.

4.4 uint2fp Example Vectors

Table 4-2 shows *uint2fp* example vectors.

Table 4-2 *uint2fp* example vectors

SIGNED INTEGER	RESULT	COMMENTS
00000000	00000000 (0.0)	Zero returns a positive 0 fp
00000001	3f800000 (1.0)	
00000002	40000000 (2.0)	
ffffffff	4f800000 (4294967296)	Rounded

5 Floating point to Signed integer (fp2sint)

5.1 fp2sint Block Diagram

The *fp2sint* module (Figure 5-1) converts single precision floating point numbers to signed 32-bit integers. The module has one input port and an output port and is fully combinatorial.

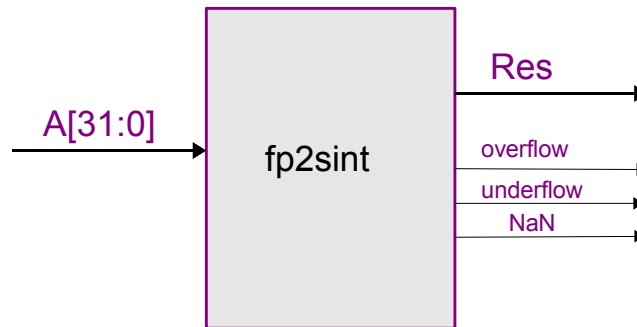


Figure 5-1 *fp2sint* Block Diagram

5.2 fp2sint Port Interface

The ports of the module are listed in Table 5-1.

Table 5-1 *fp2sint* Port Interface

PORT	TYPE	DESCRIPTION
A[31:0]	Input	Signed 2's complement 32bit integer
RES[31:0]	Output	IEEE754 Single precision floating point number
overflow	Output	Signifies that the number is larger than what can be represented
underflow	Output	Signifies that the number is smaller than what can be represented
NaN	Output	Not-a-number. Set when the input is a NaN or infinity.

5.3 Interfacing fp2sint

The design is fully combinatorial, thus should you require to pipeline the unit, you are advised to do so at the synthesis level, using a balanced registers technique that will provide better results for your target technology. Note that NaN will also be set for $-/+ \infty$ infinity, but this will also set the overflow (for positive infinity $+\infty$) or underflow (for negative infinity $-\infty$).

5.4 fp2sint Example Vectors

The Table 5-2 shows *fp2sint* example vectors.

Table 5-2 *fp2sint* example vectors.

FLOATING POINT	INTEGER	COMMENTS
7f800000 (+inf)	7fffffff	Overflow & NaN
FF800000 (-inf)	80000000	Underflow & NaN

FFC00000 (NaN)	00000000	NaN
00000000 (+0)	00000000	
80000000 (-0)	00000000	
3F800000 (+1)	00000001	
BF800000 (-1)	ffffffff	
40000000 (+2)	00000002	
C0000000 (-2)	fffffff0	
3e800000 (0.25)	00000000	rounded
BE800000 (-0.25)	00000000	rounded
3F000000 (+0.5)	00000001	rounded
BF000000 (-0.5)	ffffffff	rounded
3F400000 (+0.75)	00000001	rounded
BF400000 (-0.75)	ffffffff	rounded
7F7FFFFFFF (max+ve)	7fffffff	Overflow set
FF7FFFFFFF (max-ve)	80000000	Underflow set
00000001 (min+ve)	00000000	Denormal to zero
80000001 (min-ve)	00000000	Denormal to zero

6 Floating point to Unsigned integer (fp2uint)

6.1 fp2uint Block Diagram

The *fp2uint* module (Figure 6-1) converts single precision floating point to unsigned 32-bit integer numbers. The module has one input port and an output port and is fully combinatorial.

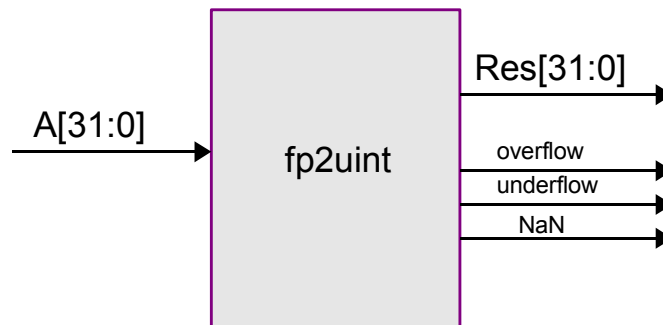


Figure 6-1 *fp2uint* Block Diagram

6.2 fp2uint Port Interface

The ports of the *fp2uint* module are listed in Table 6-1.

Table 6-1 *fp2uint* Port Interface

PORT	DIRECTION	DESCRIPTION
A[31:0]	Input	Signed 2's complement 32bit integer
RES[31:0]	Output	IEEE754 Single precision floating point number
overflow	Output	Signifies that the number is larger than what can be represented
underflow	Output	Signifies that the number is smaller than what can be represented
NaN	Output	Not-a-number. Set when the input is a NaN or infinity.

6.3 Interfacing fp2uint

The design is fully combinatorial, thus should you require to pipeline the unit, you are advised to do so at the synthesis level, using a balanced registers technique that will provide better results for your target technology. Note that NaN will also be set for $-/+ \infty$ infinity, but this will also set the overflow (for positive infinity $+\infty$) or underflow (for negative infinity $-\infty$).

6.4 fp2uint Example Vectors

Table 6-2 shows *fp2uint* example vectors

Table 6-1 *fp2uint* example vectors

FLOATING POINT	INTEGER	COMMENTS
7f800000 (+inf)	ffffffff	Overflow & NaN
FF800000 (-inf)	00000000	Underflow & NaN

FFC00000 (NaN)	00000000	NaN
00000000 (+0)	00000000	
80000000 (-0)	00000000	
3F800000 (+1)	00000001	
BF800000 (-1)	00000000	Underflow set
40000000 (+2)	00000002	
C0000000 (-2)	00000000	Underflow set
3e800000 (0.25)	00000000	Rounded
BE800000 (-0.25)	00000000	Rounded (underflow not set, this is a valid rounding to zero)
3F000000 (+0.5)	00000001	rounded
BF000000 (-0.5)	00000000	Underflow set
3F400000 (+0.75)	00000001	Rounded
BF400000 (-0.75)	00000000	Underflow set
7F7FFFFFFF (max+ve)	ffffffff	Overflow set
FF7FFFFFFF (max-ve)	00000000	Underflow set
00000001 (min+ve)	00000000	Denormal to zero
80000001 (min-ve)	00000000	Denormal to zero

7 Generator Usage

The *Floating Point Converters* generator employs a graphical web user interface (GUI) for configuring the arithmetic converter modules. In order to use the GUI you must sign-in Think Silicon Ltd web site. If already registered, click on *Sign-in* link in the upper, right side of the web page. Otherwise click on the *Register* link first and follow the instructions. As soon as you sign-in the second GUI page (Figure 7-1) appears.

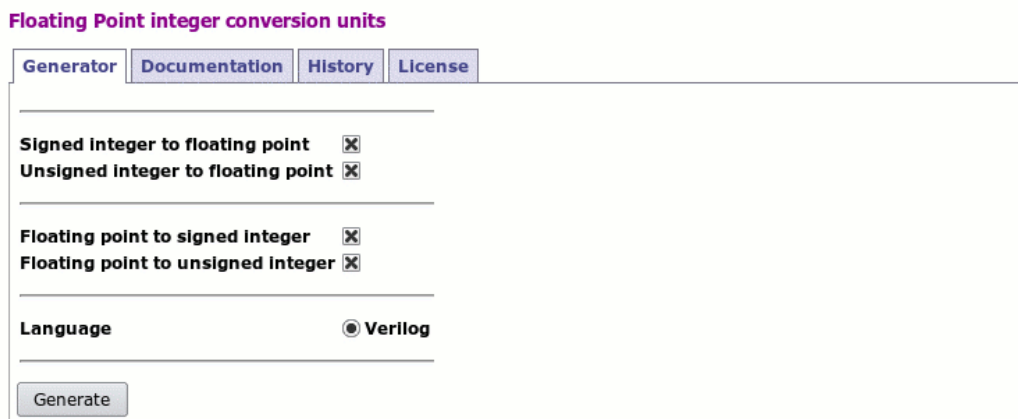


Figure 7-1 *Floating Point Converters* Generators GUI

The user is able to tick in the boxes in order choose which arithmetic conversion modules will be generated. Press the *Generate* button in order to generate the arithmetic conversion modules of your choice.

8 Deliverables

The package generated with *Floating Point Converters* consists of the present document and source code files in Verilog^{TM 1} HDL and C language. The files are listed in Table 8-1.

Table 8-1 *Floating Point Converters* Deliverables

FILE	DESCRIPTION
./uint2fp/uint2fp.v	The <i>uint2fp</i> module
./uint2fp/tb_vectors.v	The <i>uint2fp</i> testbench
./uint2fp/gen_vectors.c	Test Vector generator for <i>uint2fp</i> module
./uint2fp/Makefile	Makefile for <i>uint2fp</i> simulation scripts
./fp2uint/fp2uint.v	The <i>p2uint</i> module
./fp2uint/tb_vectors.v	The <i>p2uint</i> testbench
./fp2uint/gen_vectors.c	Test Vector generator for <i>p2uint</i> module
./fp2uint/Makefile	Makefile for <i>p2uint</i> simulation scripts

¹ Verilog is a trademark of Cadence Design Automation. (<http://www.cadence.com>)

FILE	DESCRIPTION
./fp2sint/fp2sint.v	The <i>p2sint</i> module
./fp2sint/tb_vectors.v	The <i>p2sint</i> testbench
./fp2sint/gen_vectors.c	Test Vector generator for <i>p2sint</i> module
./fp2sint/Makefile	Makefile for <i>p2sint</i> simulation scripts
./sint2fp/sint2fp.v	The <i>sint2fp</i> module
./sint2fp/tb_vectors.v	The <i>sint2fp</i> testbench
./sint2fp/gen_vectors.c	Test Vector generator for <i>sint2fp</i> module
./sint2fp/Makefile	Makefile for <i>sint2fp</i> simulation scripts
./parameters.txt	<i>Floating Point Converters</i> configuration file
./doc/TSi_fp_conv.pdf	The present document

9 Verification

The Verification Flow Diagram of the Floating Point IP cores is shown in Figure 8-1.

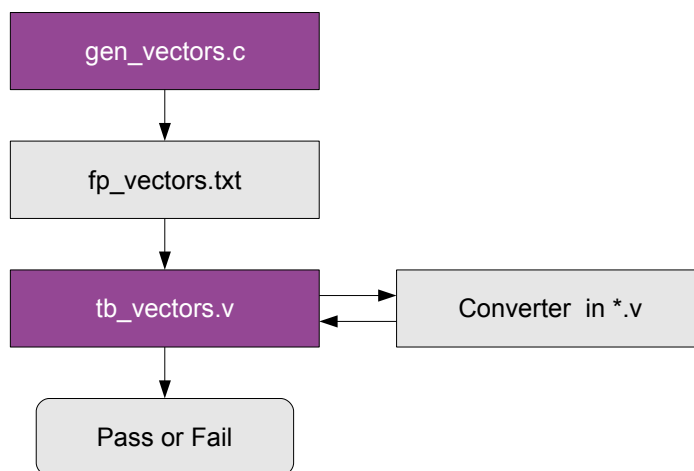


Figure 8-1 Verification Flow Diagram

The testbench is based on gcc and the Icarus Verilog simulator and we assume these are installed in your system. In order to generate test vectors and run the verification tests for the Floating Point IP cores, enter the commands shown in Figure 8-2 in a command line shell.

```
>cd ./src/sint2fp
>make

>cd ./src/uint2fp
>make

>cd ./src/fp2sint
>make

>cd ./src/fp2unit
>make
```

Figure 8-2 Verification commands

This should automatically compile the test vector generator and start the simulator. The simulator finishes with a pass or fail. The test vector generator and the netlist are written in standard C and Verilog^{TM 2}, so other compilers and Simulators, such as NCVerilog^{TM 3} and MTI Modelsim^{TM 4} can be used.

² Verilog is a trademark of Cadence Design Automation, UK (<http://www.cadence.com>)

³ NCVerilog is a trademark of Cadence Design Automation, UK (<http://www.cadence.com>)

⁴ MTI Modelsim is a trademark of Mentor Graphics (<http://www.mentor.com>)

Contact:

Think Silicon Ltd
Suite B12
Patras Science Park
Rion Achaias 26504
Greece

web: <http://www.think-silicon.com>
email: info@think-silicon.com
Tel: +30 2610 911543