



Divider_2 Core User Guide

UG-CORE-DIVIDER_2-v1.0

March 2026

www.efinixinc.com



Contents

| | |
|--|-----------|
| Introduction..... | 3 |
| Features..... | 3 |
| Device Support..... | 3 |
| Resource Utilization and Performance..... | 4 |
| Release Notes..... | 5 |
| Functional Description..... | 6 |
| Ports..... | 8 |
| Latency..... | 9 |
| Divider_2 Operations..... | 10 |
| IP Manager..... | 12 |
| Customizing the Divider_2..... | 13 |
| Divider_2 Testbench..... | 14 |
| Revision History..... | 14 |

Introduction

The Divider_2 core creates a circuit for integer division based on Radix-2 restoring division, which enables a fractional or integer remainder to be generated.

Use the IP Manager to select IP, customize it, and generate files. The Divider_2 core has an interactive wizard to help you set parameters. The wizard also has an option to create a testbench targeting an Efinix® development board.

Features

The Divider_2 core includes the following features:

- Supports quotient output with integer or fractional remainder
- Supports numerator data widths of 1 to 64 bits



Note: To reduce clock latency by one clock cycle, the unsigned reciprocal (1/X) function only support a data width of 1 bit.

- Supports denominator data widths of 2 to 64 bits
- Supports configurable pipeline latency to trade off FPGA resources versus throughput
- Supports signed and unsigned data representation format for both the numerator and denominator:
 - Two's complement signed format
- Supports signed and unsigned reciprocal (1/X) function
- Supports independent numerator, denominator, and fractional bit widths
- Fully registered outputs

Device Support

Table 1: Divider_2 Core Device Support

| FPGA Family | Supported Devices |
|-------------|-------------------|
| Trion | All |
| Titanium | All |
| Topaz | All |

Resource Utilization and Performance



Note: The resources and performance values provided are based on some of the supported FPGAs. These values are just guidance and may change depending on the device resource utilization, design congestion, and user design.

Table 2: Titanium Resource Utilization and Performance

| FPGA | Mode / Width (bit) / Latency_div PIPELINE mode | Rem | LE | FF | RAM | DSP | f _{MAX} (MHz) ⁽¹⁾ |
|--------------|---|------------|-------|-------|-----|-----|---------------------------------------|
| Ti60 F225 C4 | Signed / 8 / 8 | Int | 162 | 157 | 0 | 0 | 655 |
| | Unsigned / 8 / 8 | Int | 126 | 155 | 0 | 0 | 733 |
| | Signed / 32 / 32 | Int | 1,882 | 1,890 | 0 | 0 | 511 |
| | Unsigned / 32 / 32 | Int | 1,675 | 1,850 | 0 | 0 | 549 |
| | Signed / 8 / 4 | Int | 162 | 116 | 0 | 0 | 302 |
| | Unsigned / 8 / 4 | Int | 118 | 109 | 0 | 0 | 455 |
| | Signed / 32 / 16 | Int | 2,300 | 995 | 0 | 0 | 41 |
| | Unsigned / 32 / 16 | Int | 2,016 | 939 | 0 | 0 | 47 |
| | Unsigned / 16 / 26 | Frac (10b) | 884 | 909 | 0 | 0 | 621 |
| | Signed / 32 / 41 | Frac (10b) | 2,476 | 2,517 | 0 | 0 | 511 |
| | Unsigned / 32 / 42 | Frac (10b) | 2,414 | 2,552 | 0 | 0 | 535 |

Table 3: Trion Resource Utilization and Performance

| FPGA | Mode / Width (bit) / Latency_div PIPELINE mode | Rem | LE | FF | RAM | DSP | f _{MAX} (MHz) ⁽¹⁾ |
|---------------|--|------------|-------|-------|-----|-----|---------------------------------------|
| T20 BGA256 C4 | Signed / 8 / 8 | Int | 154 | 208 | 0 | 0 | 218 |
| | Unsigned / 8 / 8 | Int | 117 | 190 | 0 | 0 | 294 |
| | Signed / 32 / 32 | Int | 1,822 | 2,741 | 0 | 0 | 151 |
| | Unsigned / 32 / 32 | Int | 1,618 | 2,674 | 0 | 0 | 199 |
| | Signed / 8 / 4 | Int | 160 | 128 | 0 | 0 | 103 |
| | Unsigned / 8 / 4 | Int | 117 | 108 | 0 | 0 | 144 |
| | Signed / 32 / 16 | Int | 2,175 | 1,385 | 0 | 0 | 15 |
| | Unsigned / 32 / 16 | Int | 1,901 | 1,336 | 0 | 0 | 17 |
| | Unsigned / 16 / 26 | Frac (10b) | 1,276 | 849 | 0 | 0 | 235 |
| | Signed / 32 / 41 | Frac (10b) | 2,405 | 3,626 | 0 | 0 | 146 |
| | Unsigned / 32 / 42 | Frac (10b) | 2,346 | 3,700 | 0 | 0 | 197 |

Release Notes

You can refer to the IP Core Release Notes for more information about the IP core changes. The IP Core Release Notes are available on the [Efinity Downloads](#) page under each Efinity software release version.



Note: You must be logged in to the Support Center to view the IP Core Release Notes.

⁽¹⁾ These speeds are average results using a seed of 10 in the Efinity **Project Editor** > **Place and Route** tab. All other settings are default.

Functional Description

The Radix-2 Divider_2 IP core is represented by the following equations:

Integer remainder

$$\text{Numerator} = \text{Quotient} * \text{Denominator} + \text{IntRemainder}$$

For signed mode with integer remainder:

- $3/-2 = -1$ Remainder 1 or
- $-3/2 = -1$ Remainder -1

Fractional remainder

$$\left(\frac{\text{Numerator}}{\text{Denominator}} \right) = \text{Quotient} + \left(\frac{\text{IntRemain}}{\text{Denominator}} \right), \text{ where the fractional remainder} = \left(\frac{\text{IntRemain}}{\text{Denominator}} \right)$$

In the fractional case, the result is a quotient at WIDTHD bit width with a fractional remainder at WIDTHF bit width. In signed operation, all operands and results are in two's complement form. The MSB is the sign bit and the rest of the bits are quotient or fractional remainder.

For example, a five-bit dividend, divisor, and fractional output:

$$-15/2 = 15/-2 = -7.0 + (-0.5)$$

Result:

$$\text{Quotient} = \mathbf{11001} (-7.0)$$

$$\text{Remainder} = \mathbf{11000} (-0.5)$$

The bold numbers above are the sign bits, in this case it is 1.



Note: The quotient, remainder, and fractional results of division by zero are undefined. You can monitor the `div_by_zero` port to watch for divide-by-zero cases.

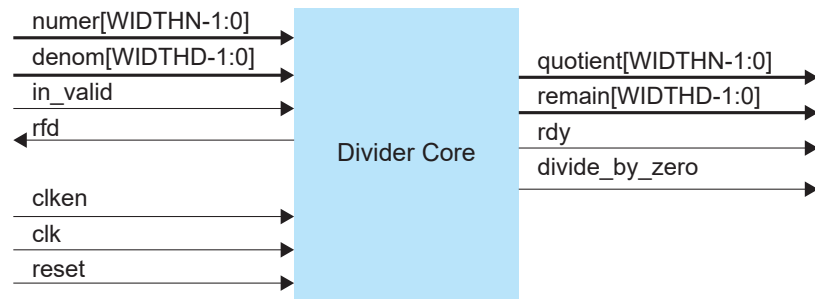
You can set the numerator and denominator bit widths independently. The bit width of the quotient is equal to the bit width of the numerator, WIDTHN, and the bit width of the remainder is equal to the width of the denominator, WIDTHD.

To implement the fractional reciprocal 1/X function:

- For both the numerator and denominator signed operation: the numerator bit width is set to 2b and tied to 2'b01 (1) or 2'b11 (-1) and the fractional bit width is selected.
- For both the numerator and denominator unsigned operation: the numerator bit width is set to 1b and tied to 1'b1 (1) and the fractional bit width is selected. This setting reduces latency by one clock cycle.
- For the signed numerator (-1) and unsigned denominator operation: the numerator bit width is set to 1b and tied to 1 (-1) and the fractional bit width is selected. This reduces latency by one clock cycle.

The `denom[WIDTHN-1:0]` input is the X value and the `remain[WIDTHN-1:0]` output is the result.

Figure 1: Divider_2 Core Block Diagram



Ports

Table 4: Divider_2 Ports

| Port | Direction | Clock | Description |
|---|-----------|-------|---|
| numer [WIDTHN-1:0] | Input | clk | Numerator input to the Divider_2 core for calculation. Allowed bit widths: 1 ⁽²⁾ - 64 |
| denom [WIDTHD-1:0] | Input | clk | Denominator input to the Divider_2 core for calculation. Allowed bit widths: 2 - 64 |
| quotient [WIDTHN-1:0] | Output | clk | Quotient output from the Divider_2 core after calculation. |
| remain [WIDTHD-1:0] or remain [WIDTHF-1:0] | Output | clk | Remainder output from the Divider_2 core after calculation. <ul style="list-style-type: none"> Integer remainder: Result data bit width determined by divisor width. Fractional remainder: Result data bit width determined by fractional width. For signed operation (either <code>numer</code> or <code>denom</code>), the output is in two's complement form. |
| clk | Input | N/A | System clock. |
| clken (optional) | Input | clk | Clock enable. When deasserted to low, all the inputs are ignored and the core remains in its current state. Tie to 1 if unused. 0: Idle 1: In progress |
| in_valid (optional) | Input | clk | Input data valid. Tie to 1 if unused. |
| rfd | Output | clk | Ready for data signal. When latency is set to 0, this signal is constantly high. 0: Calculation is in progress. 1: Divider_2 core calculation is completed and ready for the next input data. This port is available only when PIPELINE = 0. |
| reset (optional) | Input | clk | Synchronous active high reset. Tie 0 if unused |
| rdy (optional) | Output | clk | Output signal ready. When LATENCY_DIV is set to 0, this pin is tied high. You can choose to ignore this pin. |
| div_by_zero (optional) | Output | clk | Detection of division by zero. |



Note: Because the two's complement function is not symmetrical (the positive range is $(2^{\text{WIDTHN}}-1)$ while the negative range is (-2^{WIDTHN})), the numerator and quotient must be large to support the maximum possible quotient in signed operations. To handle the difference, the dividend and quotient widths must be extended by 1 bit.

⁽²⁾ 1b is only used for the reciprocal (1/X) fractional function under certain conditions. Refer to the [Functional Description](#) on page 6 for further information.

Latency

Latency is measured when `clken` goes high and is defined as $LATENCY_DIV + 1$, based on the following rules:

- For integer remainder dividers: latency $\leq WIDTHN + 1$. In a fully pipelined configuration, $LATENCY_DIV = WIDTHN$.
- For fractional remainder dividers: latency $\leq WIDTHN + WIDTHF + 1$. In a fully pipelined configuration, $LATENCY_DIV = WIDTHN + WIDTHF$

Table 5: Divider_2 Core Latency

| Signed | Fractional | Latency |
|--------|------------|--------------------------------------|
| X | X | $\leq WIDTHN + 1$ |
| X | ✓ | $\leq WIDTHN + WIDTHF + 1$ |
| ✓ | X | $\leq WIDTHN + 1$ |
| ✓ | ✓ | $\leq WIDTHN + (WIDTHF^*)^{(3)} + 1$ |



Note: You can trade off latency for f_{MAX} to reduce the resource usage. For configurable $LATENCY_DIV$, you need to be aware that the actual contribution of $WIDTHF^*$ to the latency output is $(WIDTHF - 1)$.

⁽³⁾ In the Fractional signed operation, the $WIDTHF^*$ is $(WIDTHF - 1)$ since the MSB of the Remainder is used for the signed indicator.

Divider_2 Operations

The following waveforms show the integer and fractional divider using the following settings:

- Both `numer` and `denom` are 5b
- Fractional remainder is 5b
- All unsigned operations

Figure 2: Waveform when PIPELINE = 1

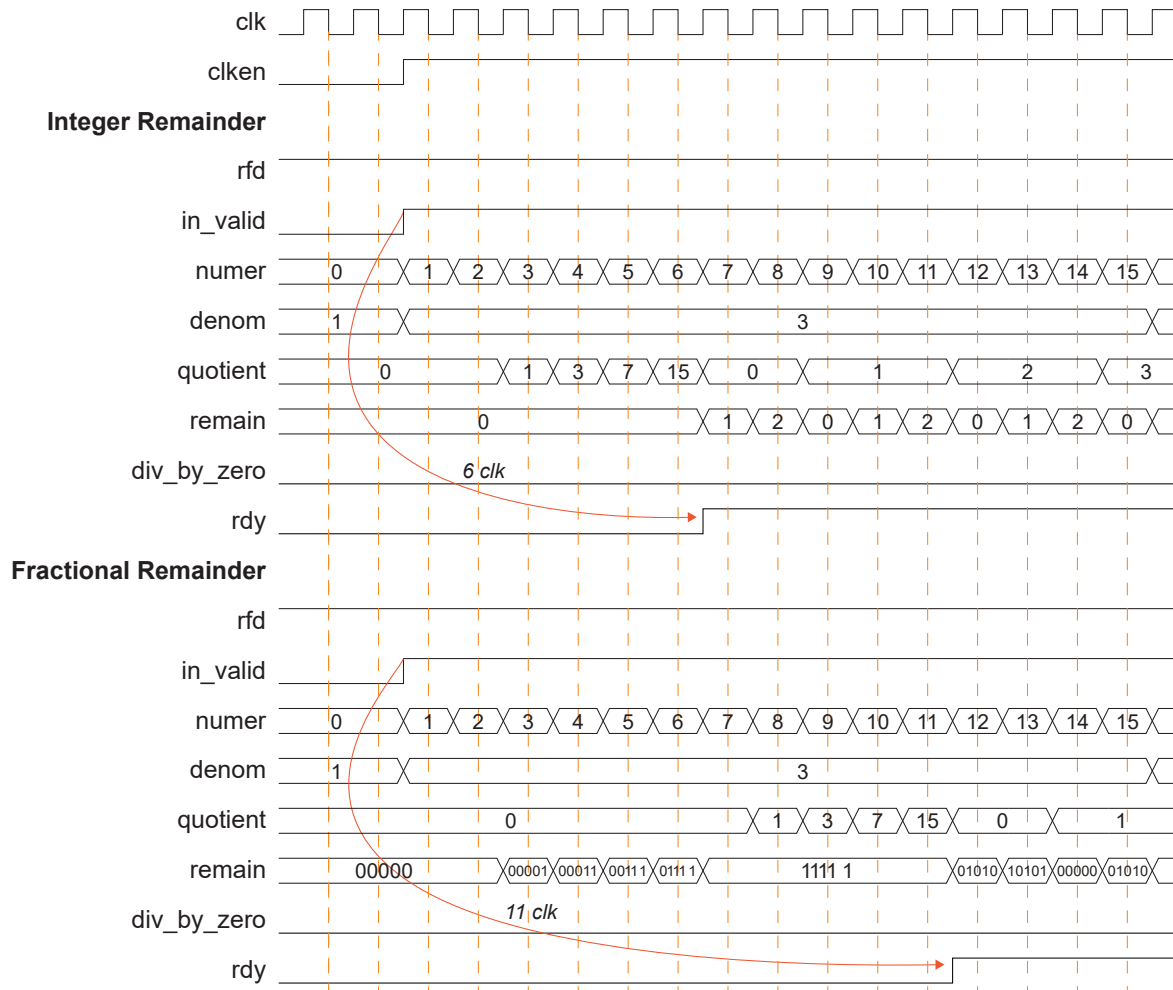
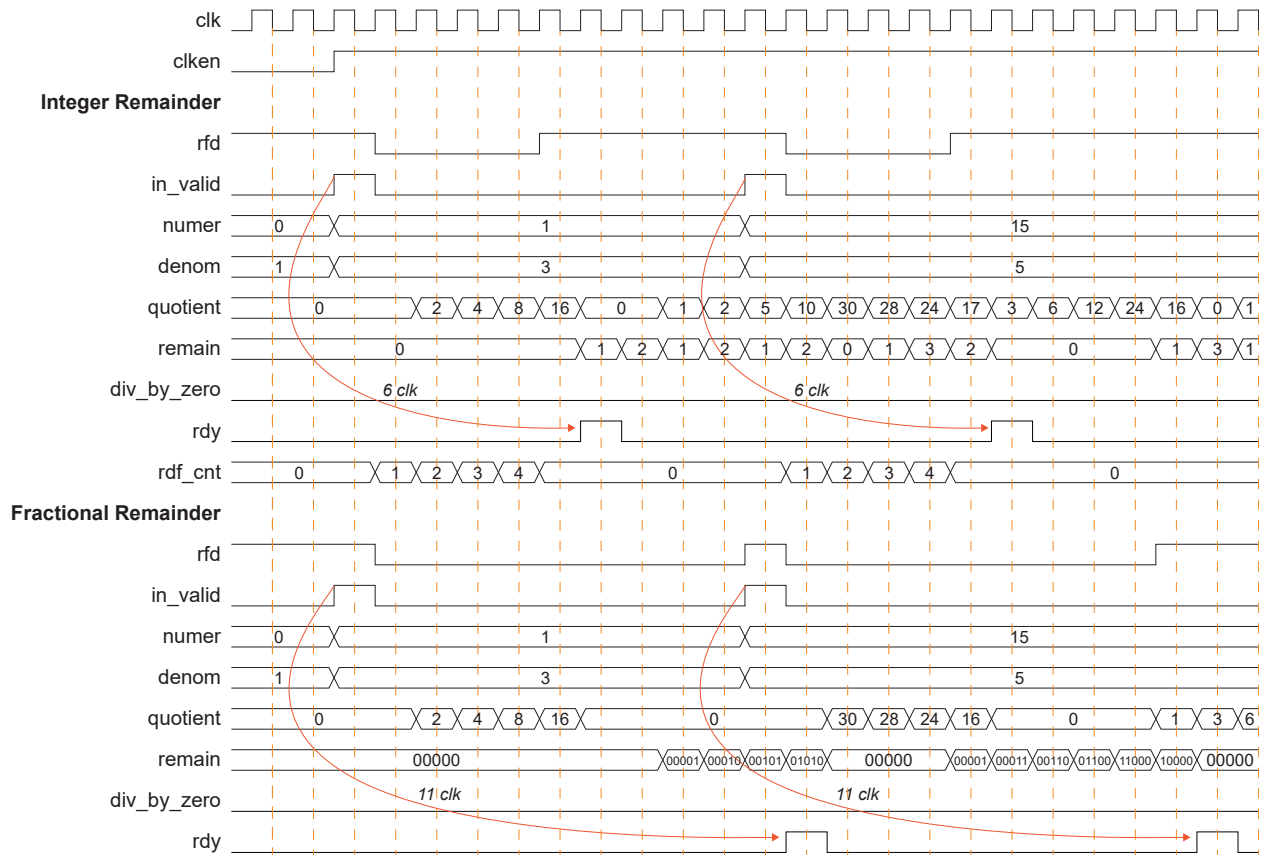


Figure 3: Waveform when PIPELINE = 0



IP Manager

The Efinity® IP Manager is an interactive wizard that helps you customize and generate Efinity® IP cores. The IP Manager performs validation checks on the parameters you set to ensure that your selections are valid. When you generate the IP core, you can optionally generate an example design targeting an Efinity development board and/or a testbench. This wizard is helpful when you use several IP cores, multiple instances of an IP core with different parameters, or the same IP core across different projects.



Note: Not all Efinity IP cores include an example design or a testbench.

Generating the Divider_2 Core with the IP Manager

The following steps explain how to customize an IP core with the IP Configuration wizard.

1. Open the IP Catalog.
2. Choose **Arithmetic** > **Divider_2** core and click **Next**. The **IP Configuration** wizard opens.
3. Enter the module name in the **Module Name** box.



Note: You cannot generate the core without a module name.

4. Customize the IP core using the options shown in the wizard. For detailed information on the options, refer to the *Customizing the Divider_2* section.
5. (Optional) In the **Deliverables** tab, specify whether to generate an IP core example design targeting an Efinity® development board and/or testbench. These options are turned on by default.
6. (Optional) In the **Summary** tab, review your selections.
7. Click **Generate** to generate the IP core and other selected deliverables.
8. In the **Review configuration generation** dialog box, click **Generate**. The Console in the **Summary** tab shows the generation status.



Note: You can disable the **Review configuration generation** dialog box by turning off the **Show Confirmation Box** option in the wizard.

9. When generation finishes, the wizard displays the **Generation Success** dialog box. Click **OK** to close the wizard.

The wizard adds the IP to your project and displays it under **IP** in the Project pane.

Generated Files

The IP Manager generates these files and directories:

- **<module name>_define.vh**—Contains the customized parameters.
- **<module name>.v**—IP source code.
- **settings.json**—Configuration file.
- **Testbench**—Contains generated RTL and testbench files.

Customizing the Divider_2

The core has parameters so you can customize its function. You set the parameters in the **General** tab of the core's IP Configuration window.

Table 6: Divider_2 Core Parameters

| Parameters | Option | Description |
|-----------------|--------|--|
| WIDTHN | 1 - 64 | Defines the numerator and quotient data width. Default: 16 |
| WIDTHD | 2 - 64 | Defines the denominator and remainder data width. Default: 16 |
| WIDTHF | 1 - 64 | Fractional remainder bit width when REMAIN_FMT is set to 1. In signed operation, the MSB of the Frac Remain is for signed representation, contributing to the total bit width used for fractional division is (FRAC_NUM - 1). Default: 10 |
| NREPRESENTATION | 0, 1 | Sign representation of numerator input. When this parameter is set to SIGNED , the Divider_2 core interprets the numer input as signed two's complement. 0: Unsigned 1: Signed Default: Unsigned |
| DREPRESENTATION | 0, 1 | Sign representation of denominator input. When this parameter is set to SIGNED , the Divider_2 core interprets the denom input as signed two's complement. 0: Unsigned 1: Signed Default: Unsigned |
| PIPELINE | 0, 1 | Enables or disables the Divider_2 core pipeline. 0: Disable 1: Enable Default: Enable |
| LATENCY_DIV | - | The clock latency per division. Integer remainder: $\leq \text{WIDTHN} + 1$ Fractional remainder: $\leq \text{WIDTHN} + \text{WIDTHF}^{(4)} + 1$ Default: 16 |
| REMAIN_FMT | 0, 1 | Remainder is in integer or fractional mode. 0: Integer 1: Fractional Default: 0 |

⁽⁴⁾ Refer to Latency for more information on Fractional WIDTHF.

| Parameters | Option | Description |
|------------|--------|--|
| REMAIN_POS | 0, 1 | The value of the remainder must be greater than or equal to zero. 0: Value can be +ve/-ve or zero 1: Value is always +ve or zero Default: 0 |

Divider_2 Testbench

When you generate the Divider IP in the IP Manager Configuration window, the IP wizard generates a testbench automatically.

The wizard generate the following files in the `<project>/ip/<instances_name>/Testbench/example_design/modelsim` directory:

1. `divider_rem_frac_efx_ed.sv` (Remainder in fractional mode)
2. `divider_rem_int_efx_ed.sv` (Remainder in integer mode)
3. `divider_tb.sv`
4. `Makefile`
5. `README`

The included `README` file explains how to use the `divider_rem_frac_efx_ed.sv` and `divider_rem_int_efx_ed.sv` files for simulation.

To run the simulation, type `make all`. When simulation completes successfully, the simulator shows a passing result:

```
Simulation PASSED
```

Revision History

Table 7: Revision History

| Date | Document Version | IP Version | Description |
|------------|------------------|------------|------------------|
| March 2026 | 1.0 | 1.0 | Initial release. |