# Speedster7t Soft IP User Guide (UG103)

*Speedster FPGAs*

Achronix
Data Acceleration

# Copyrights, Trademarks and Disclaimers



## Notice of Disclaimer



## Achronix Semiconductor Corporation

2903 Bunker Hill Lane
Santa Clara, CA 95054
USA

Website: www.achronix.com
E-mail : info@achronix.com

# Table of Contents

# Chapter 1 : Overview

There are a number of available soft IP cores for the Speedster®7t family of FPGAs. Each of these cores has an IP configurator within ACE that allows configuration of the soft IP core. When configured, the generated wrapper for the core can be instantiated within a user project enabling both synthesis and simulation of the design.

This document describes the available soft IP cores and the methods for configuration and instantiation of each. Soft IP cores are primarily implemented using the components present in the FPGA programmable fabric. For details of these components, refer to the ***Speedster7t IP Component Library User Guide*** (UG086)[1].

## Instructions

Within ACE, all IP cores are accessed using the IP perspective (see the "Perspectives" chapter in the ***ACE User Guide*** (UG070)[2]. The flow and method for generating user IP cores is fully detailed in the "Creating an IP Configuration" chapter in the ACE User Guide. Unless directed otherwise in the following sections, follow the instructions in the ACE User Guide.

## I/O Ring and Core

Within the Speedster7t FPGA there are two categories of IP core. These are listed within the IP Libraries pane as **I/O Ring** and **Core**. For the Speedster7t family of FPGAs, the following soft IP cores are available:

1 https://www.achronix.com/documentation/speedster7t-ip-component-library-user-guide-ug086
2 https://www.achronix.com/documentation/ace-user-guide-ug070

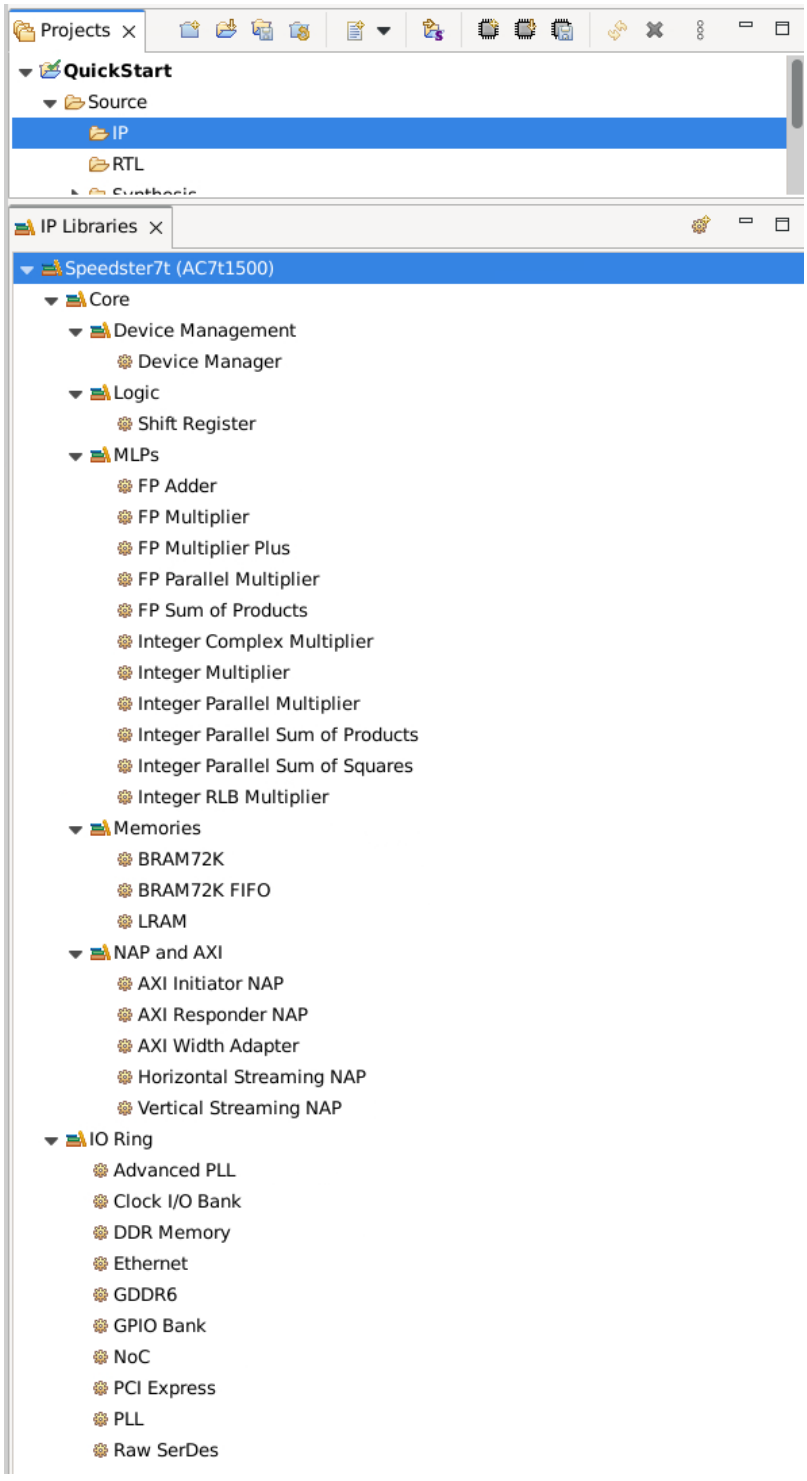


***Figure 1 • Speedster7t IP Libraries View***

### I/O Ring

The I/O ring contains the configuration for each of the IP cores located in the I/O ring of the FPGA such as the clock banks, PLLs, GDDR, DDR, GPIO, PCIe, 2D NoC, SerDes and Ethernet. The configuration of each of these IP cores is detailed in its respective User Guide.

### Core

The **Core** view contains the available IP configurators for the selected target FPGA used in the project. For the Speedster7t FPGAs, the available soft cores are **shown above** (see figure 1), and listed in the **Available Configurators** (page 3) chapter. The details of how to configure each of these cores are given in the appropriate chapter.

# Available Configurators

## Memories

- **BRAM72K Soft IP** (page 5) – for creating large block RAM memory arrays
- **LRAM Soft IP** (page 9) – for creating large local RAM memory arrays

## MLPs

- **Integer Complex Multiplier Soft IP** (page 13) – for creating a complex multiplier with a single machine learning processing block.
- **Integer Multiplier Soft IP** (page 18) – for creating a single multiplier of up to 32 × 32
- **Integer Parallel Multiplier Soft IP** (page 23) – for creating parallel multipliers of up to 32 × 32
- **Integer Parallel Sum of Products Soft IP** (page 29) – for integer sum of products from up to 24 multipliers
- **Integer Parallel Sum of Squares Soft IP** (page 35) – for integer sum of squares inputs
- **Integer RLB Multiplier Soft IP** (page 40) – for creating a single multiplier using RLBs in the fabric logic
- **Floating Point Adder** (page 45) – for two-input floating-point adder with optional accumulation
- **Floating Point Multiplier** (page 50) – for two-input floating-point multiplier with optional accumulation
- **Floating Point Multiplier Plus** (page 55) – for a two-input floating-point multiplier with a third adder and optional accumulation
- **Floating Point Parallel Multiplier** (page 60) – for two parallel floating-point multiplies with optional accumulation
- **Floating Point Sum of Products** (page 66) – for floating-point addition of two products

## Logic

- **Shift Register Soft IP** (page 71) – for creating DFF-based shift registers

## NAP and AXI

- **AXI Initiator NAP Soft IP** (page 74) – for creating a NAP that initiates AXI traffic

- **AXI Responder NAP Soft IP** (page 78) – for creating a NAP that responds to AXI traffic
- **AXI Width Adapter Soft IP** (page 82) – for creating a width adapter to connect to the initiator or responder NAPs

# Chapter 2 : BRAM72K Soft IP

## Description

The Speedster7t BRAM72K soft IP core creates an arbitrary sized memory array, comprised of ACX_BRAM72K primitives. The macro employs the embedded data and address cascade paths between ACX_BRAM72K primitives enabling fast connections for both address and data paths.

If only a single ACX_BRAM72K is required, this primitive can be inferred or instantiated in the code directly. However, if a memory array comprising multiple BRAM72K blocks is required, it is recommended to use the soft IP configuration to enable the optimum architecture.

## Configuration

The user macro has the following configuration options:

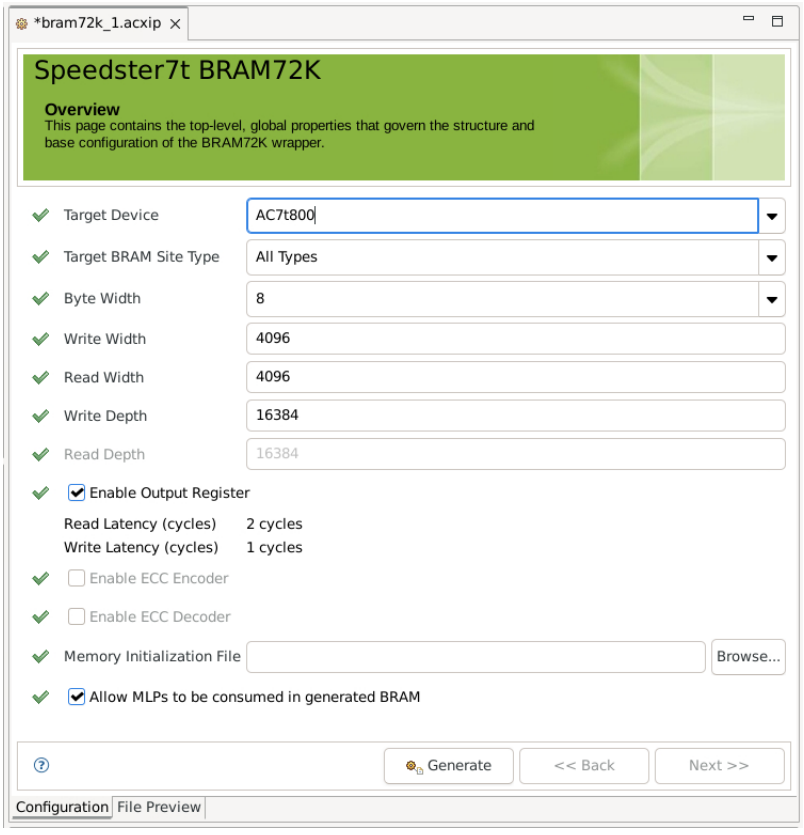


***Figure 2 • BRAM72K Soft IP Configuration***

### Table 1 • Configuration Options

| Name | Default | Range | Description |
|---|---|---|---|
| Target Device | Selected target device | – | Set to match the target device of the project. |
| Byte Width | 9 | 8, 9 | Determines whether fields should be set to 8-bit or 9-bit. |
| Write Width | 72 | 1 to 9216 | Write port data width. Values greater than 144 limit the write depth to 16K words. |
| Read Width | 72 | 1 to 9216 | Read port data width. Values greater than 144 limit the write depth to 16K words. |
| Write Depth | 1024 | 512 to 1048576 | Write port address depth. The maximum value is limited by the number of BRAM72K blocks in a column in the target device. The maximum value is also dependent on the write width as detailed in the table, Write and Read Depths versus Data Width (page 7) |
| Read Depth | 1024 | 512 to 1048576 | Read port address depth. Currently set to match the write depth. This value cannot be changed by the user. Future releases of this user macro are planned to allow configuring different write and read depths. |
| Enable Output Register | Off | On, Off | Determines whether the output register in each of the BRAM72K primitives is enabled. Adds an additional cycle of latency to any read operation. |
| Enable ECC Encoder | Off | On, Off | Determines whether the ECC encoder is enabled for writes to the memory array. This option is currently disabled and cannot be set by the user. |
| Enable ECC Decoder | Off | On, Off | Determines whether the ECC encoder is enabled for reads from the memory array. This option is currently disabled and cannot be set by the user. |
| Use Memory Initialization File (1) | Off | On, Off | Determines whether a memory initialization file is used to initialize the memory contents. This initialization occurs for both synthesis and simulation. When this option is enabled, entry of the file location in the associated file browser dialog is permitted. |
| Allow MLPs to be Consumed in Generated BRAM | On | On, Off | If enabled, use the I/O from MLPs in the same time in certain modes, thus consuming the MLP and saving on BRAM utilization. |

| Name | Default | Range | Description |
|---|---|---|---|

**Table Notes**

1. If relative paths are used for the memory initialization file location, the same relative paths must be valid from both the ACE project directory and the simulation directory. It is recommended to locate both of these directories at the same relative depth in the project tree, and to use relative paths that navigate up the tree to the first common directory, before descending the tree to the location of the files.

   For example: The Achronix reference designs locate the ACE project in `<project root>/src/ace`. The simulation directories are located in `<project root>/sim/vcs` or `<project root>/sim/questa`. The memory initialization files are located in `<project root>/src/mem_init_files`. A relative path correct for both simulation and ACE is `"../../src/mem_init_files/filename.txt"`.

# Write and Read Depth

## Absolute Limits

The write and read depths are related to the write and read widths. The absolute limit on these values is detailed in the following table:

***Table 2 • Write and Read Depths Versus Data Width***

| Memory Width | Maximum Memory Depth |
|---|---|
| 1 to 144 | 1048576 |
| 145 to 9216 | 16384 |

## Device Specific Limits

Within a device, the largest memory that can be generated is limited by the number of ACX_BRAM72K primitives in a column. For example, if there are 64 ACX_BRAM72K primitives in a column, then the maximum memory size is 64 × 72K bits = 4,718,592 bits. This would support a configuration of 72-bits × 64K depth, or 36-bits × 128K depth.

# Examples

The following figure shows the macro configured for a 4096-bit by 16,384 entry memory with the memory output register enabled:

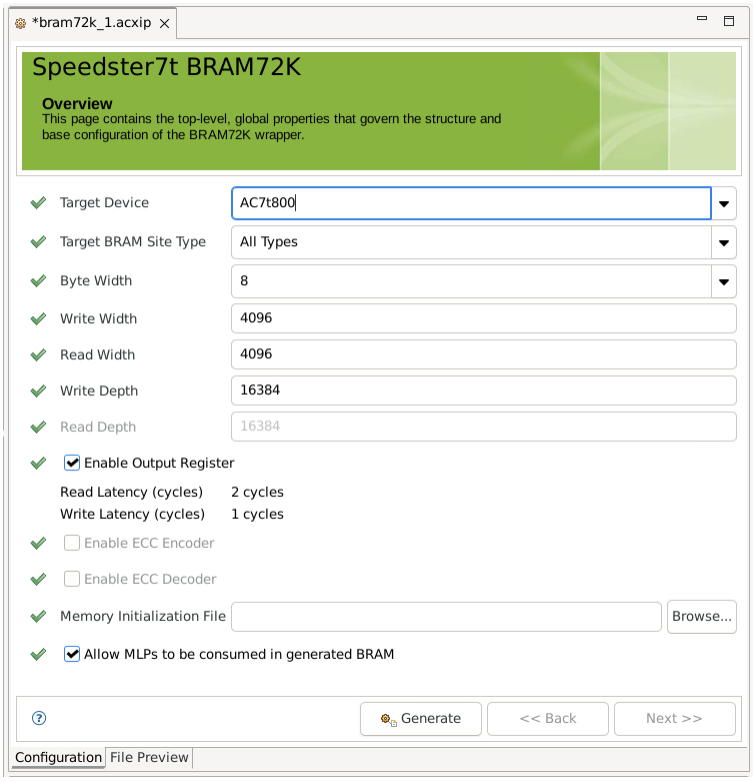


***Figure 3 • 4096 × 16K Memory Configuration***

The following figure shows the IP diagram for the above configuration:

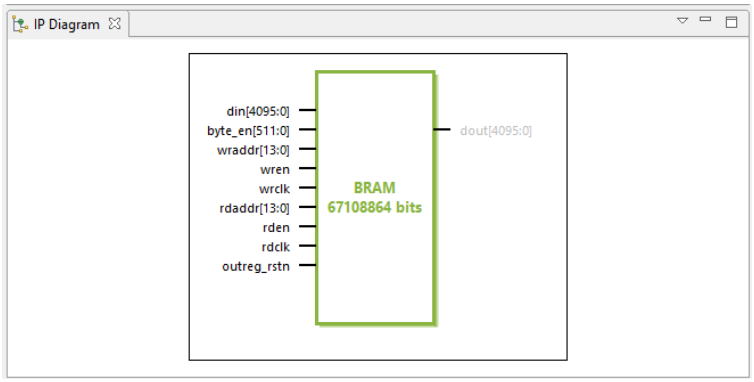


***Figure 4 • 4096 × 16K Memory IP Diagram***

# Chapter 3 : LRAM Soft IP

## Description

The LRAM soft IP core creates an arbitrary sized memory array comprised of LRAM primitives.

If only a single LRAM is required, this primitive can be inferred or instantiated into the code directly. However, if a memory array consisting of multiple LRAM primitives is required, it is recommended to use the soft IP configurator to achieve the optimum architecture.

## Utilization

> **Note**
>
> Within the Speedster7t family, the LRAM and MLP primitives share a site. Therefore if a site is allocated for use as an LRAM primitive, the MLP on that same site cannot be used.

## Configuration

The LRAM2K soft IP configurator has the following configuration options:

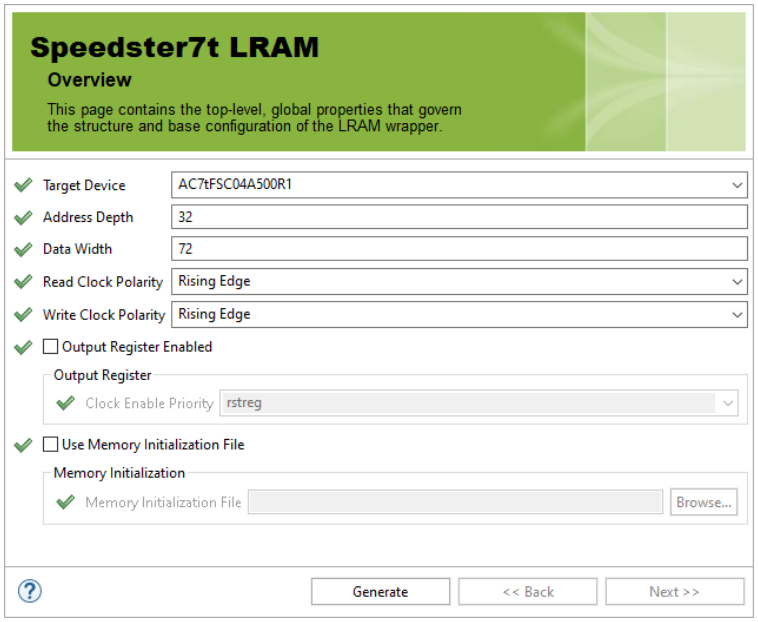


***Figure 5 • LRAM Soft IP Configurator***

### Table 3 • Configuration Options

| Name | Default | Range | Description |
|---|---|---|---|
| Target Device | Selected target device | - | Set to match the target device of the project. |
| Address Depth | 32 | 4 to 4096 | Address depth of the memory array in words. The depth imposes limitations on the maximum data width. The limits are detailed in Read and Write Depths Versus Data Widths (page 11). |
| Data Width | 72 | 1 to 184320 | Data port width in bits of both `din` and `dout`. |
| Read Clock Polarity | Rising Edge | Falling or Rising Edge | The `rdclk` active edge on which all read transactions occur. |
| Write Clock Polarity | Rising Edge | Falling or Rising Edge | The `wrclk` active edge on which all write transactions occur. |
| Output Register Enabled | Off | On, Off | Determines whether the output register in each of the LRAM primitives is enabled. This adds an additional cycle of latency to any read operation. If the output register is disabled, the memory array is combinatorial. The output changes when the input address changes. |
| Output Register Clock Enable Priority | rstreg | rstreg, rstce | Controls the clock enable input of the output register.<br>• rstreg – the `outregce` input is ignored when `rstregn` = `1'b0`. The output register is reset on the next active `rdclk` edge.<br>• rstce – `outregce` must be equal to `1'b1` and `rstregn` = `1'b0` for the output register to be reset on the next active `rdclk` edge. |
| Use Memory Initialization File (1) | Off | On, Off | Determines whether a memory initialization file is used to initialize the memory contents. This initialization occurs for both synthesis and simulation. When this option is enabled, entry of the file location in the associated file browser dialog is permitted. |

**Table Notes**

1. If relative paths are used for the memory initialization file location, the same relative paths must be valid from both the ACE project directory and the simulation directory. It is recommended to locate both these directories at the same relative depth in the project tree, and to use relative paths that navigate up the tree to the first common directory, before descending the tree to the location of the files.

   For example, the Achronix reference designs locate the ACE project in `<project root>/src/ace`. The simulation directories are located in `<project root>/sim/vcs` or `<project root>/sim/questa`. The memory initialization files are located in `<project root>/src/mem_init_files`. A relative path correct for both simulation and ACE is `"../../src/mem_init_files/filename.txt"`.

# Write and Read Depth

## Absolute Limits

The write and read depths are related to the write and read widths. The absolute limit on these values is detailed in the following table:

***Table 4 • Write and Read Depths versus Data Width***

| Memory Width | Maximum Memory Depth |
|---|---|
| 4 to 32 | 184320 |
| 33 to 64 | 92160 |
| 65 to 96 | 61416 |
| 97 to 128 | 46080 |
| 129 to 144 | 36864 |

## Device Specific Limits

Within a device, the largest memory that can be generated is limited by the number of ACX_LRAM primitives in a column. For example, if there are 64 ACX_LRAM2K primitives in a column, the maximum memory size is 64 × 2K bits = 131,027 bits. This would support a configuration of 64-bits x 2K depth, or 32-bits × 4K depth. Alternatively, if there are 64 ACX_LRAM4K in a column, the maximum memory size is 64 × 4K bits = 262,144 bits. This would support a configuration of 64-bits × 4K depth, or 32-bits × 8K depth.

The type of LRAM used by the Speedster7t LRAM configurator is dependent upon the device chosen and the available LRAM types within the fabric.

# Examples

The following figure shows the soft IP configured for a 128-bit × 4096-word LRAM memory with the memory output register enabled:

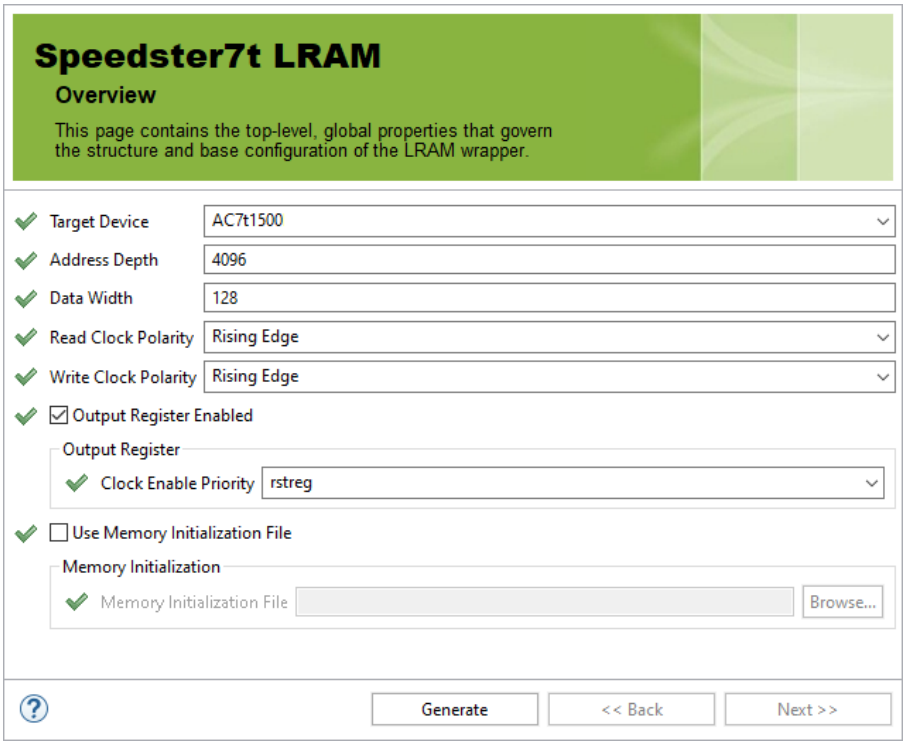


***Figure 6 • 128-Bit x 4096-Word LRAM Memory Configuration***

The following figure shows the IP diagram for the above configuration:

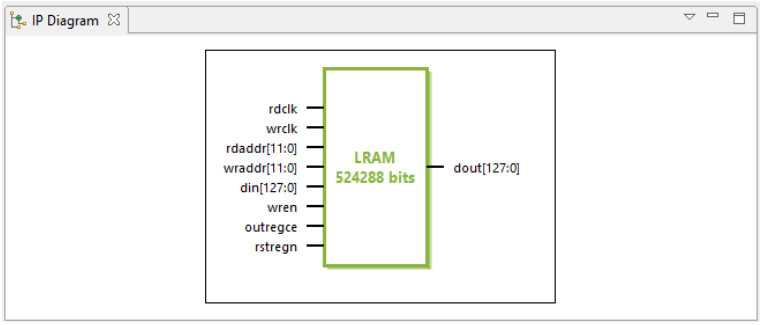


***Figure 7 • 128-Bit x 4096-Word LRAM Memory IP Diagram***

# Chapter 4 : Integer Complex Multiplier Soft IP

## Description

The Integer Complex Multiplier Soft IP implements a complex multiplication with a single machine learning processing block.

## Configuration

The integer complex multiplier soft IP configurator has the following options:

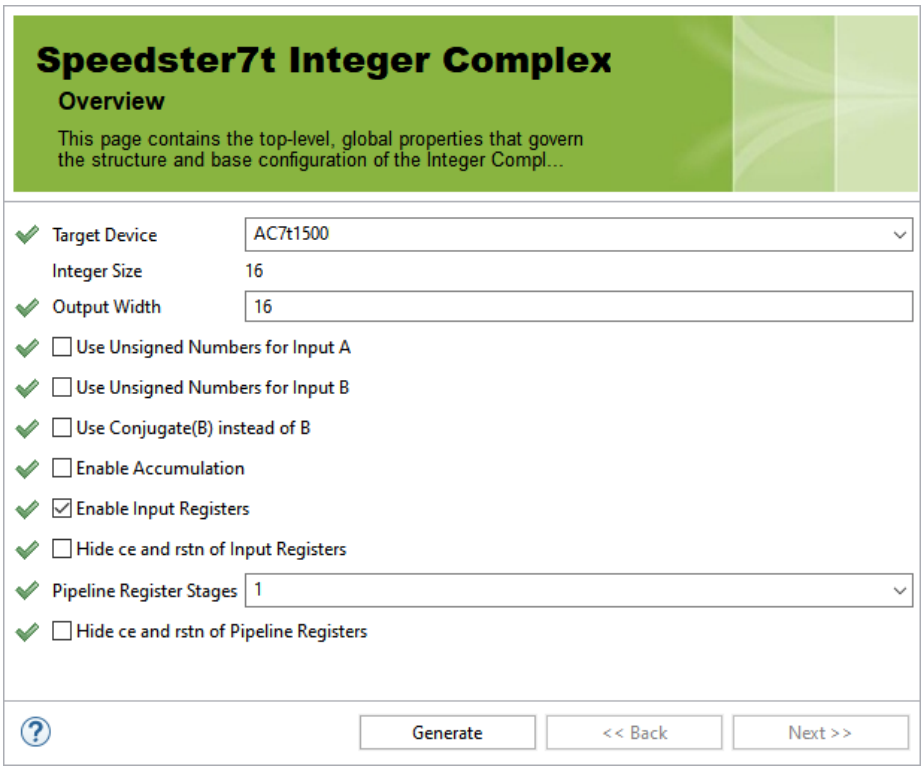


***Figure 8 • Complex Integer Multiplier Soft IP Configurator***

### Table 5 • Configuration Options

| Name | Default | Range | Description |
|---|---|---|---|
| Target Device | AC7t1500 | All Speedster7t devices | Set to match the target device of the project. |
| Integer Size | 16 | 16 | Number of bits of each input coefficient. |
| Output Width | 16 | 1-36 | Width of the o_dout_re and o_dout_im outputs.(1) |
| Use Unsigned Numbers for Input A | No | Yes, No | 0 – i_din_a_re and i_din_a_im are signed (two's complement).<br>1 – i_din_a_re and i_din_a_im are unsigned. |
| Use Unsigned Numbers for Input B | No | Yes, No | 0 – i_din_b_re and i_din_b_im are signed (two's complement).<br>1 – i_din_b_re and i_din_b_im are unsigned. |
| Use Conjugate(B) instead of B | No | Yes, No | Compute A × conjugate(B) instead of A × B. |
| Enable Accumulation | No | Yes, No | Enables accumulation:<br>• No – No accumulation: (o_dout_re + o_dout_im·j) = (i_din_a_re + i_din_a_im·j) × (i_din_b_re + i_din_b_im·j).<br>• Yes – Accumulation: o_dout_re + o_dout_im·j is the accumulated value. The start of accumulation is signaled by asserting i_load = 1.<br>When i_load is high, the previous value of the internal accumulation register is ignored, and the new value is stored. The output is then set to this value. When i_load is low, the old and new values are added, and the sum is stored.<br>The i_load signal is internally pipelined to have the same latency as the input. If a set of inputs start a new accumulation, i_load must be high when those inputs are presented. If accumulation is not enabled, the i_load signal is ignored. |
| Enable Input Registers | No | Yes, No | 0 – No input registers.<br>1 – i_din_a_re, i_din_a_im, i_din_b_re, and i_din_b_im are registered. The input registers are controlled by the i_in_reg_a_ce, i_in_reg_b_ce, and i_in_reg_rstn inputs. Enabling the input register adds one cycle of latency. |
| Hide ce and rstn of Input Registers | No | Yes, No | If selected, the i_in_reg_a_ce, i_in_reg_b_ce, and i_in_reg_rstn inputs are automatically tied high (1'b1). |

| Name | Default | Range | Description |
|---|---|---|---|
| Pipeline Register Stages | 0 | 0, 1, 2 | The number of pipeline registers, not counting the input register. The total latency is `pipeline_regs` + `in_reg_enable`. |
| Hide `ce` and `rstn` of Pipeline Registers | No | Yes, No | If selected, the `i_pipeline_ce` and `i_pipeline_rstn` inputs are automatically tied high (`1'b1`). |

**Table Notes**

1. Ensure that Output Width is sufficient to represent the maximum result that can be accumulated. In the event of overflow, the higher order bits of any result are truncated. When accumulation is enabled, it might be necessary to increase the data output width to account for the growth of the result over multiple accumulation cycles.

   The maximum result width can be calculated as:

   $2 \times \text{Integer Size} + \lceil \log 2 \ (\text{N umber of Accumulation Cycles}) \rceil$

### *Table 6 • Ports*

| Name | Direction | Description |
|---|---|---|
| `i_clk` | Input | Clock input, used for the (optional) registers and accumulator. |
| `i_din_a_re`[(Integer Size – 1):0] | Input | Real coefficient of A data input to multiplier. |
| `i_din_a_im`[(Integer Size – 1):0] | Input | Imaginary coefficient of A data input to multiplier. |
| `i_din_b_re`[(Integer Size – 1):0] | Input | Real coefficient of B data input to multiplier. |
| `i_din_b_im`[(Integer Size – 1):0] | Input | Imaginary coefficient of B data input to multiplier. |
| `i_load` | Input | Resets the accumulator to A × B, ignoring the previous value. This signal is internally pipelined to have the same latency as A × B. |
| `i_in_reg_a_ce` | Input | (Optional) Clock enable for `i_din_a_re` and `i_din_a_im`. Present when **Enable Input Registers** is set to **On**. |
| `i_in_reg_b_ce` | Input | (Optional) Clock enable for `i_din_b_re` and `i_din_b_im`. Present when **Enable Input Registers** is set to **On**. |
| `i_in_reg_rstn` | Input | (Optional) Synchronous active-low reset for input registers. Present when **Enable Input Registers** is set to **On**. |
| `i_pipeline_ce` | Input | (Optional) Clock enable for pipeline and accumulator registers. Present when **Enable Accumulation** is set to **On** or when **Pipeline Register Stages** is greater than 0. |
| `i_pipeline_rstn` | Input | (Optional) Synchronous active-low reset for pipeline and accumulator registers. Present when **Enable Accumulation** is set to **On** or when **Pipeline Register Stages** is greater than 0. |

| Name | Direction | Description |
|---|---|---|
| o_dout_re[(Output Width – 1):0] | Output | Real coefficient of the result of multiplication and accumulation. Always signed (2's complement). |
| o_dout_im[(Output Width – 1):0] | Output | Imaginary coefficient of the result of multiplication and accumulation. Always signed (2's complement). |

# Examples

The following example shows the integer complex multiplier configured for 16-bit unsigned inputs with input registers, accumulation and two pipeline stages, and 32-bit output:

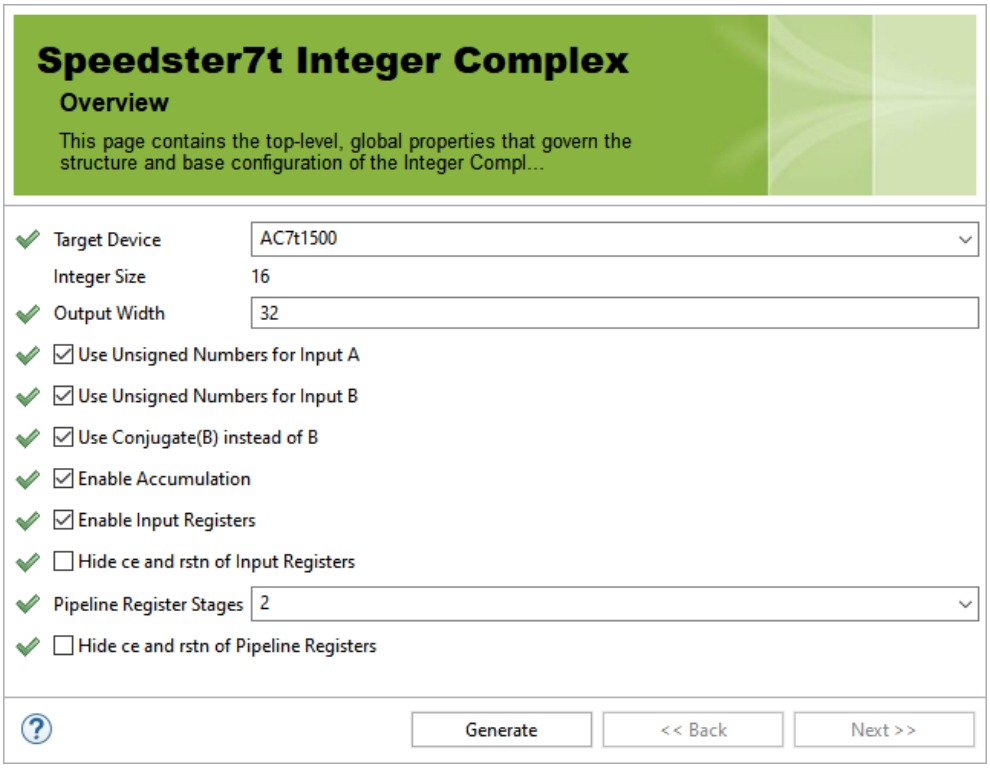


***Figure 9 • Two Pipeline Stage Integer Multiplier Configuration***

The following figure shows the IP diagram for the above configuration:

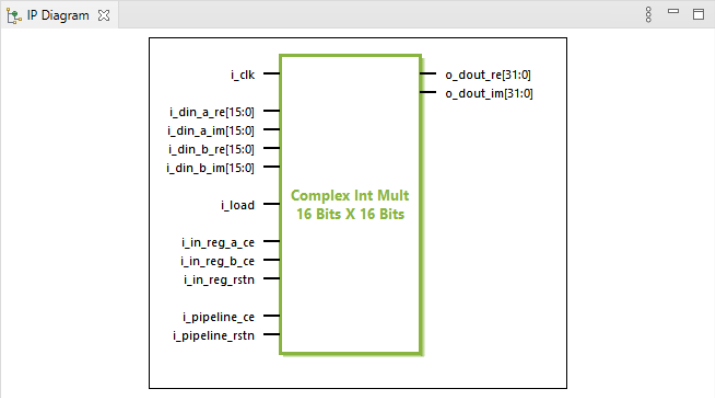


***Figure 10 • Two Pipeline Stage Integer Multiplier I/O***

# Chapter 5 : Integer Multiplier Soft IP

## Description

The Integer Multiplier soft IP core configures a two-input integer multiplier using either RLB (logic) based multipliers or MLP primitives. The multiplier also supports optional result accumulation. The configurator supports sizes of up to 32 × 32 integers, with both signed and unsigned numerical formats.

## Configuration

The Integer Multiplier soft IP configurator has the following options:

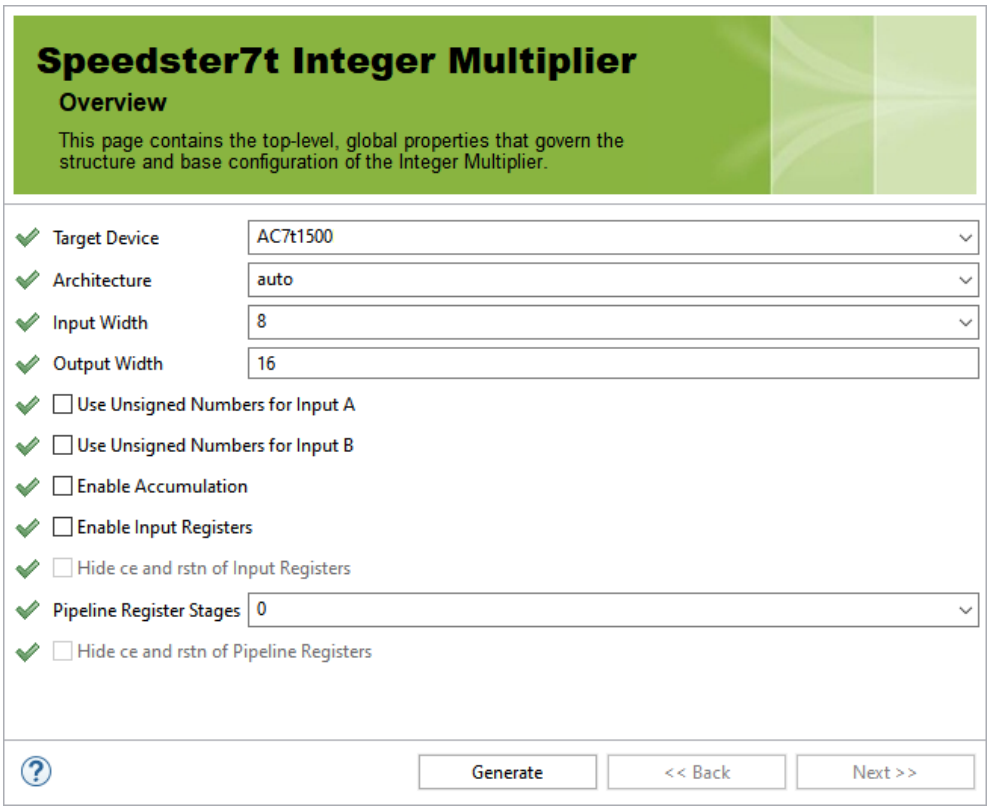


***Figure 11 • Integer Multiplier Soft IP Configurator***

***Table 7 • Configuration Options***

| Name | Default | Range | Description |
|---|---|---|---|
| Target Device | AC7t1500 | All Speedster7t devices | Set to match the target device of the project. |
| Architecture | auto | auto, rlb, mlp | Determines which primitives should be used to implement the multiplier.<br>• **auto**: Allow the tool to choose the most appropriate primitive based on the number formats and sizes selected.<br>• **rlb**: Implement the multiplier in fabric logic. The Speedster7t FPGA has a unique MLUT structure which supports very efficient multiplier arrays using logic.<br>• **mlp**: Use the MLP primitive to implement the multiplier. |
| Input Width | 8 | 3, 4, 5, 6, 7, 8, 16 or 32 | Width of both a and b inputs. |
| Use Unsigned Numbers for Input A/B | Off | On, Off | When set, configures the appropriate input to use unsigned numbers. By default, the inputs are set to signed. |
| Enable Input Registers | Off | On, Off | When set, enables a register stage for both A and B inputs. This stage adds a cycle of latency to all results. Enabling the input registers adds the inputs `i_in_reg_a_ce`, `i_in_reg_b_ce` and `i_in_reg_rstn` to the resultant soft IP. |
| Hide `ce` and `rstn` of Input Registers | No | Yes, No | If selected, the `i_in_reg_a_ce`, `i_in_reg_b_ce`, and `i_in_reg_rstn` inputs are automatically tied high (`1'b1`). |
| Enable Accumulation(1) | Off | On, Off | Enables accumulation:<br>• Off – no accumulation: `o_dout = i_din_a × i_din_b`.<br>• On – accumulation: `o_dout` is the accumulated value. The start of accumulation is signaled by asserting `i_load = 1`.<br>When `i_load` is high, the previous value of the internal accumulation register is ignored, and the new value is stored. The output is then set to this value. When `i_load` is low, the old and new values are added, and the sum is stored. The output is `o_dout = (i_din_a × i_din_b) + dout_prev`.<br>The `i_load` signal is internally pipelined to have the same latency as the input. If a set of inputs start a new accumulation, `i_load` must be high when those inputs are presented. If accumulation is not enabled, the `i_load` signal is ignored. |

| Name | Default | Range | Description |
|---|---|---|---|
| Pipeline Register Stages | 0 | 0, 1, 2, 3 | Add pipeline register stages through the multiplication process. Enabling pipeline registers improves timing performance at the cost of an additional cycle of latency for each stage enabled.<br>When any pipeline stages are enabled, the inputs `i_pipeline_ce` and `i_pipeline_rstn` are added to the resultant soft IP. |
| Hide `ce` and `rstn` of Pipeline Registers | No | Yes, No | If selected, the `i_pipeline_ce` and `i_pipeline_rstn` inputs are automatically tied high (`1'b1`). |
| Output Width | 16 | 3 to 128 | Width of the data output. Automatically updated by the configurator when Input Width is updated. In addition, the value can be modified to meet requirements.<br>The valid range changes dependent upon the Input Width and Architecture. (1) |

**Table Notes**

1. When accumulation is enabled, it might be necessary to increase the data output width to account for the growth of the result over multiple accumulation cycles. The maximum result width can be calculated as (2 × Input Width) + ⌈ log 2 ( number of accumulation cycles ) ⌉.

### Table 8 • Ports

| Name | Direction | Description |
|---|---|---|
| i_clk | Input | Clock input, used for the (optional) registers and accumulator. |
| i_din_a[(Input Width – 1):0] | Input | 'A' data input to the multiplier. |
| i_din_b[(Input Width – 1):0] | Input | 'B' data input to the multiplier. |
| i_in_reg_a_ce | Input | (Optional) Clock enable for i_din_a. Present when **Enable Input Registers** is set to **On**. |
| i_in_reg_b_ce | Input | (Optional) Clock enable for i_din_b. Present when **Enable Input Registers** is set to **On**. |
| i_in_reg_rstn | Input | (Optional) Synchronous active-low reset for input registers. Present when **Enable Input Registers** is set to **On**. |
| i_pipeline_ce | Input | (Optional) Clock enable for pipeline and accumulator registers. Present when **Enable Accumulation** is set to **On** or when **Pipeline Register Stages** is greater than 0. |
| i_pipeline_rstn | Input | (Optional) Synchronous active-low reset for pipeline and accumulator registers. Present when **Enable Accumulation** is set to **On** or when **Pipeline Register Stages** is greater than 0. |
| i_load (1) | Input | (Optional) When asserted to 1'b1, resets the accumulator to i_din_a × i_din_b, ignoring the previous value. Present when **Enable Accumulator** is set to **On**. |
| o_dout[(Output Width – 1):0] | Output | Result of multiplication and accumulation. |

**Table Notes**

1. This signal is internally pipelined to have the same latency as i_din_a and i_din_b.

# Examples

The following example shows the integer multiplier configured for signed 32 × 32 inputs, with accumulation and a single pipeline stage:

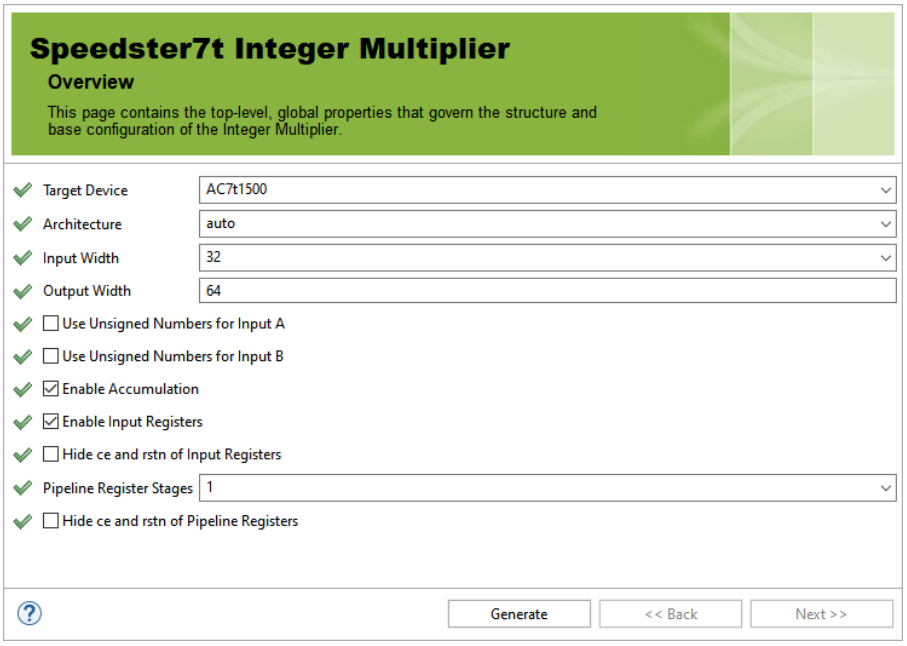


***Figure 12 • 32 × 32 Signed Integer Multiplier Configuration***

The following figure shows the IP diagram for this configuration:

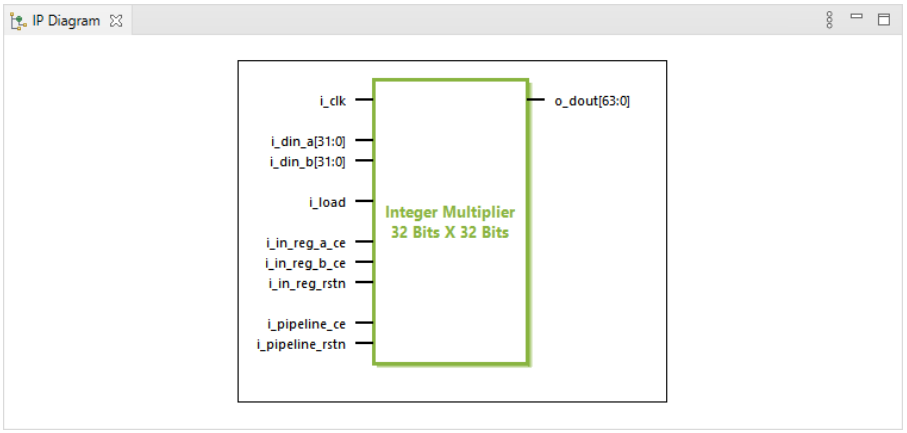


***Figure 13 • 32 x 32 Signed Integer Multiplier I/O***

# Chapter 6 : Integer Parallel Multiplier Soft IP

## Description

The Integer Parallel Multiplier soft IP core configures multiple integer multipliers. This soft IP core is implemented with the MLP primitive which contains an array of integer multipliers. There can be up to 8 separate multipliers ranging from 3 × 3 to 16 × 16 bits. The multipliers support both signed and unsigned arithmetic.

## Configuration

The Integer Parallel Multiplier soft IP configurator has the following options:

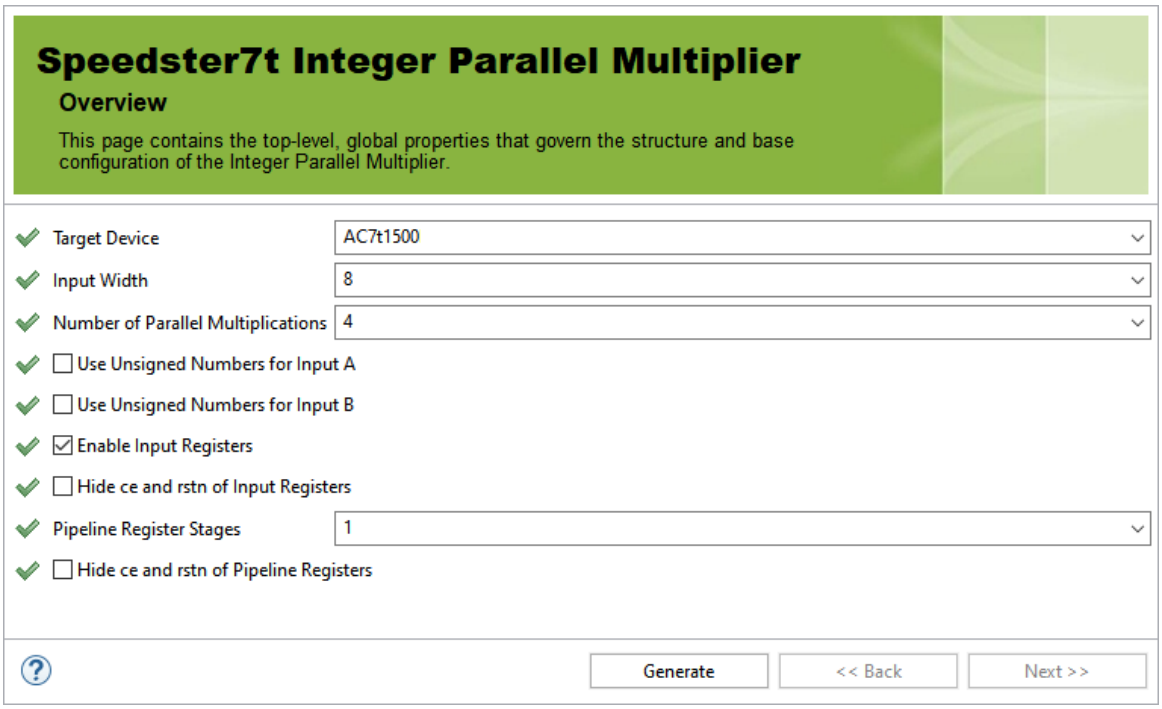


***Figure 14 • Integer Parallel Multiplier Soft IP Configurator***

***Table 9 • Configuration Options***

| Name | Default | Range | Description |
|---|---|---|---|
| Target Device | AC7t1500 | All Speedster7t devices | Set to match the target device of the project. |
| Input Width | 8 | 3, 4, 5, 6, 7, 8 or 16 | Width of the two inputs to each multiplier |
| Number of Parallel Multiplications | 4 | 2 to 8 | The number of parallel multipliers to be implemented. The maximum number of multipliers is determined by the Input Width. Refer to the Number of Multipliers Per Input Width (page 25) table for details. |
| Use Unsigned Numbers for Input A/B | Off | On, Off | When set, configures the appropriate inputs to use unsigned numbers. By default, the inputs are set to signed. |
| Enable Input Registers | Off | On, Off | When set, enables a register stage for all multiplier inputs. This adds a cycle of latency to all results. Enabling the input registers adds these inputs to the resultant soft IP core:<br>• `i_in_reg_a_ce`<br>• `i_in_reg_b_ce`<br>• `i_in_reg_rstn` |
| Hide `ce` and `rstn` of Input Registers | No | Yes, No | If selected, the `i_in_reg_a_ce`, `i_in_reg_b_ce`, and `i_in_reg_rstn` inputs are automatically tied high (`1'b1`). |
| Pipeline Register Stages | 0 | 0, 1 | Adds a pipeline register stage to the multiplication process. Enabling pipeline registers improves timing performance at the cost of an additional cycle of latency. When any pipeline stages are enabled, these inputs are added to the resultant soft IP core:<br>• `i_pipeline_ce`<br>• `i_pipeline_rstn` |
| Hide `ce` and `rstn` of Pipeline Registers | No | Yes, No | If selected, the `i_pipeline_ce` and `i_pipeline_rstn` inputs are automatically tied high (`1'b1`). |

***Table 10 • Number of Multipliers Per Input Width***

| Input Width | Maximum Number of Multipliers |
|---|---|
| 3 | 8 |
| 4 | 8[1] |
| 5 | 4 |
| 6 | 4 |
| 7 | 4 |
| 8 | 4 |
| 16 | 2 |

**Table**

1. If Input Width=4 and if either input is unsigned, the maximum number of multipliers is 4.

***Table 11 • Ports***

| Name | Direction | Description |
|---|---|---|
| `i_clk` | Input | Clock input to drive the (optional) registers and accumulator. |
| `i_din_a`[(Input Width – 1):0] | Input | Packed (page 26) vector of data to 'A' inputs of multipliers. |
| `i_din_b`[(Input Width – 1):0] | Input | Packed (page 26) vector of data to 'B' inputs of multipliers. |
| `i_in_reg_a_ce` | Input | (Optional) Clock enable for `i_din_a`. Present when **Enable Input Registers** is set to **On**. |
| `i_in_reg_b_ce` | Input | (Optional) Clock enable for `i_din_b`. Present when **Enable Input Registers** is set to **On**. |

| Name | Direction | Description |
|---|---|---|
| i_in_reg_rstn | Input | (Optional) Synchronous active-low reset for input registers. Present when **Enable Input Registers** is set to **On**. |
| i_pipeline_ce | Input | (Optional) Clock enable for pipeline registers. Present when **Pipeline Register Stages** is set to **1**. |
| i_pipeline_rstn | Input | (Optional) Synchronous active-low reset for pipeline registers. Present when **Pipeline Register Stages** is set to **1**. |
| o_dout[(Output Width – 1):0] | Output | Output bus consisting of the results from all the multipliers in parallel. Output Width is dynamically calculated by the configurator. See Output Format (page 26) for details of how the results are assembled in the single output bus. |

## Input Format

Each multiplier input is formed from an array of the individual inputs packed in a single input vector:

**Code**

```
i_din_a/b(i) = i_din_a/b[i * int_size +: int_size];
```

## Output Format

For each multiplier, the result width in bits is equal to 2 × **Input Width**.

The results from all the parallel multiplications are output as a concatenation on the o_dout output. The width of this output is calculated as follows:

Output Width = **Number of Parallel Multiplications** × 2 × **Input Width**.

The bit lanes used for the result of an individual multiplier are found by multiplying the number of the multiplier (starting at 0) by the result width.

### Example

If four 8 × 8 multipliers are configured:

- Result Width = 2 × 8 = 16 bits
- Output Width = 4 × 16 = 64 bits

Each individual multiplier result appears in the lanes detailed in the following table.

***Table 12 • Output Bus Organization***

| Multiplier Number | Result |
|---|---|
| 0 | o_dout[15:0] |
| 1 | o_dout[31:16] |
| 2 | o_dout[47:32] |
| 3 | o_dout[63:48] |

# Examples

The following example shows the integer parallel multiplier configured for two signed 16 × 16 multiplications, with input registers and and an internal pipeline stage:

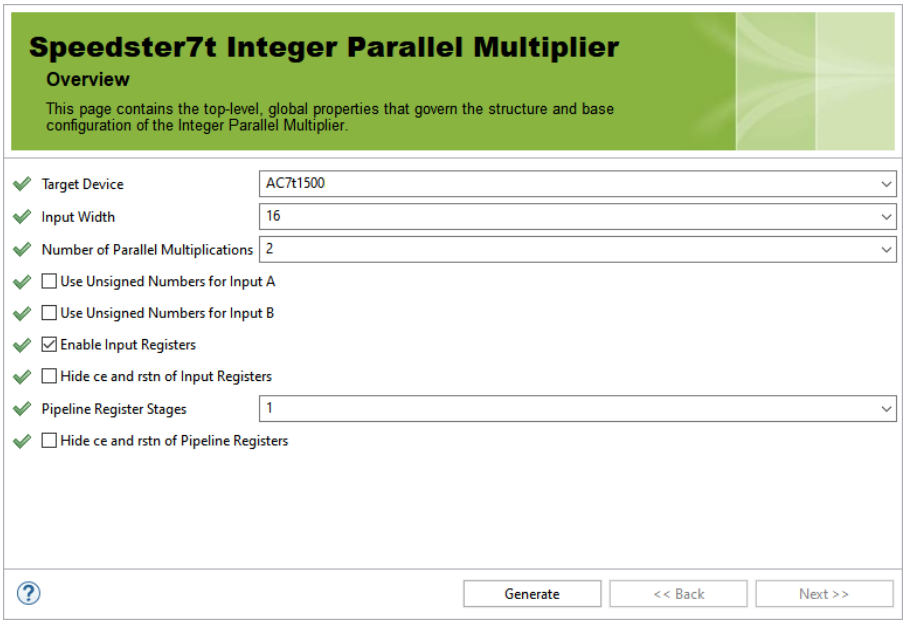


***Figure 15 • Two 16 × 16 Signed Parallel Integer Multiplier Configuration***

The following figure shows the IP diagram for this configuration:

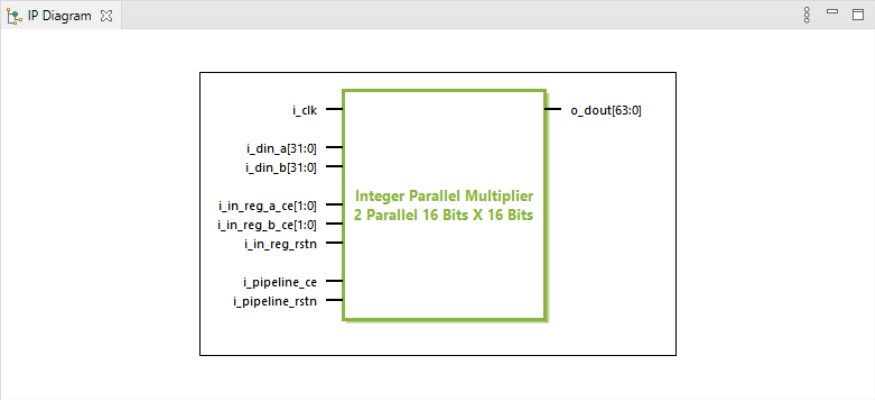


***Figure 16 • Two 16 × 16 Signed Parallel Integer Multiplier I/O***

# Chapter 7 : Integer Parallel Sum of Products Soft IP

## Description

The Integer Parallel Sum of Products soft IP core configures multiple parallel integer multipliers with a single summed result. This soft IP core is implemented with the MLP primitive which contains an array of integer multipliers and associated adders. Up to 24 parallel multipliers, ranging from 3 × 3 to 16 × 16 bits can be used. The multipliers support both signed and unsigned arithmetic. The final output can optionally be accumulated.

## Configuration

The integer parallel sum of products soft IP configurator has the following options:

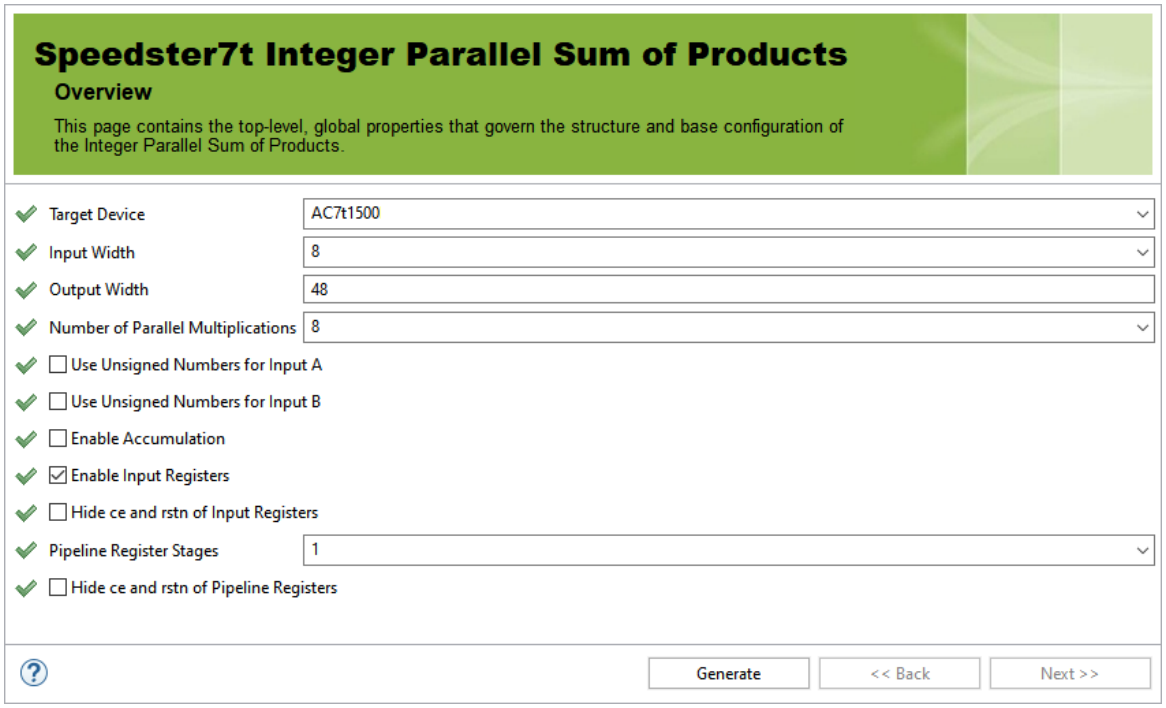


***Figure 17 • Integer Parallel Sum of Products Configurator***

***Table 13 • Configuration Options***

| Name | Default | Range | Description |
|---|---|---|---|
| Target Device | AC7t1500 | All Speedster7t devices | Set to match the target device of the project. |
| Input Width | 8 | 3, 4, 5, 6, 7, 8 or 16 | Width of the two inputs to each multiplier. |
| Number of Parallel Multiplications | 8 | 1 to 24 | The number of parallel multipliers to be implemented and their results summed. The maximum number of multipliers is determined by the Input Width. Refer to the Maximum Number of Multipliers Per Input Width table for details. |
| Use Unsigned Numbers for Input A/B | Off | On, Off | When set, configures the appropriate inputs to use unsigned numbers. The inputs are signed by default. |
| Enable Input Registers | Off | On, Off | When set, enables a register stage for all multiplier inputs. This adds a cycle of latency to all results. Enabling the input registers adds these inputs to the resultant soft IP core:<br>• `i_in_reg_a_ce`<br>• `i_in_reg_b_ce`<br>• `i_in_reg_rstn` |
| Hide ce and rstn of Input Registers | No | Yes, No | If selected, the `i_in_reg_a_ce`, `i_in_reg_b_ce`, and `i_in_reg_rstn` inputs are automatically tied high (`1'b1`). |
| Enable Accumulation | Off | On, Off | Enables accumulation:<br>• Off – no accumulation<br>• On – accumulation: `o_dout` is the accumulated value. The start of accumulation is signaled by asserting `i_load = 1`.<br>When Accumulation is enabled, these inputs are added to the resultant soft IP core:<br>• `i_load`<br>• `i_pipeline_ce`<br>• `i_pipeline_rstn`<br>When `i_load` is high, the previous value of the internal accumulation register is ignored, and the new value is stored. The output is then set to this value. When `i_load` is low, the old and new values are added, and the sum is stored.<br>The `i_load` signal is internally pipelined to have the same latency as the input. If a set of inputs start a new accumulation, `i_load` must be high when those inputs are presented. If accumulation is not enabled, the `i_load` signal is ignored. |

| Name | Default | Range | Description |
|---|---|---|---|
| Pipeline Register Stages | 0 | 0, 1 or 2 | Adds pipeline register stages to the multiplication process. Enabling pipeline register stages improves timing performance at the cost of an additional cycle of latency per stage. When any pipeline stages are enabled, these inputs are added to the resultant soft IP core:<br>• `i_pipeline_ce`<br>• `i_pipeline_rstn` |
| Hide `ce` and `rstn` of Pipeline Registers | No | Yes, No | If selected, the `i_pipeline_ce` and `i_pipeline_rstn` inputs are automatically tied high (`1'b1`). |
| Output Width (1) | 48 | 3 to 48 | Width of the data output. By default, this value is set to 48 bits. This value can be reduced if required. |

**Table Notes**

1. Ensure that Output Width is sufficient to represent the maximum result that can be accumulated. In the event of overflow, the higher order bits of any result are truncated.

   When accumulation is enabled, it might be necessary to increase the data output width to account for the growth in the result over multiple accumulation cycles. The maximum result width can be calculated as:

   $(2 \times \text{Input Width}) + \lceil \log 2\ (\text{Number of Parallel Multiplications} \times \text{Number of Accumulation Cycles}) \rceil$

### *Table 14 • Maximum Number of Multipliers Per Input Width*

| | Wide Mode(2) | | Normal Mode | |
|---|---|---|---|---|
| Input Width | Max Signed Multiplications (1) | Max Unsigned Multiplications | Max Signed Multiplications (1) | Max Unsigned Multiplications |
| 3 | 24 | 16 | 12 | 8 |
| 4 | 16 | 12 | 8 | 6 |
| 5 | 12 | 12 | 6 | 6 |
| 6 | 12 | 10 | 6 | 5 |
| 7 | 10 | 8 | 5 | 4 |
| 8 | 8 | 8 | 4 | 4 |
| 16 | 4 | 4 | 2 | 2 |

| | Wide Mode[2] | | Normal Mode | |
|---|---|---|---|---|
| Input Width | Max Signed Multiplications [1] | Max Unsigned Multiplications | Max Signed Multiplications [1] | Max Unsigned Multiplications |

**Table Notes**

1. Both inputs must be signed. If either input is unsigned, the corresponding unsigned column applies.
2. When more than half the multipliers are used, the MLP72 is automatically configured in wide mode, which blocks the adjacent BRAM72K from being used.

### *Table 15 • Ports*

| Name | Direction | Description |
|---|---|---|
| `i_clk` | Input | Clock input, used for the (optional) registers and accumulator. |
| `i_din_a`[(*data width* – 1):0] | Input | Packed (page 33) data vector to the 'A' inputs of the multipliers where *data width* = **Input Width** × **Number of Parallel Multiplications**. |
| `i_din_b`[(*data width* – 1):0] | Input | Packed (page 33) data vector to the 'B' inputs of the multipliers where *data width* = **Input Width** × **Number of Parallel Multiplications**. |
| `i_in_reg_a_ce` | Input | (Optional) Clock enable for `i_din_a`. Present when **Enable Input Registers** is set to **On**. |
| `i_in_reg_b_ce` | Input | (Optional) Clock enable for `i_din_b`. Present when **Enable Input Registers** is set to **On**. |
| `i_in_reg_rstn` | Input | (Optional) Synchronous active-low reset for input registers. Present when **Enable Input Registers** is set to **On**. |
| `i_pipeline_ce` | Input | (Optional) Clock enable for pipeline and accumulation registers. Present when **Enable Accumulation** is set to **On** or when **Pipeline Register Stages** is greater than 0. |
| `i_pipeline_rstn` | Input | (Optional) Synchronous active-low reset for pipeline and accumulation registers. Present when **Enable Accumulation** is set to **On** or when **Pipeline Register Stages** is greater than 0. |
| `i_load` [1] | Input | (Optional) When asserted to `1'b1`, resets the accumulator to `i_din_a` × `i_din_b`, ignoring the previous value. Present when **Enable Accumulator** is set to On. |
| `o_dout`[(Output Width – 1):0] | Output | Output bus consisting of the sum of products from all of the multipliers in parallel. |

| Name | Direction | Description |
|---|---|---|

**Table Notes**

1. This signal is internally pipelined to have the same latency as `i_din_a` and `i_din_b`.

## Input Format

Each multiplier input is formed from an array of the individual inputs, packed in a single input vector.

```
i_din_a / b(i) = i_din_a / b[i*int_size +: int_size];
```

# Examples

The following example shows the integer parallel sum of products configured for four signed 16 × 16 multiplications with input registers, accumulation and a single internal pipeline stage.

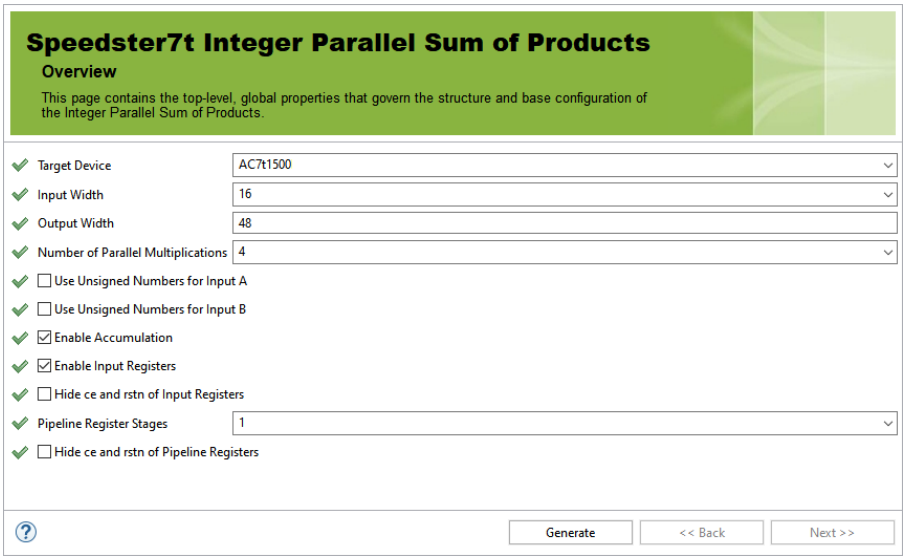


***Figure 18 • Four 16 × 16 Signed Multiplier Sum of Products Configuration***

The following figure shows the IP diagram for this configuration:

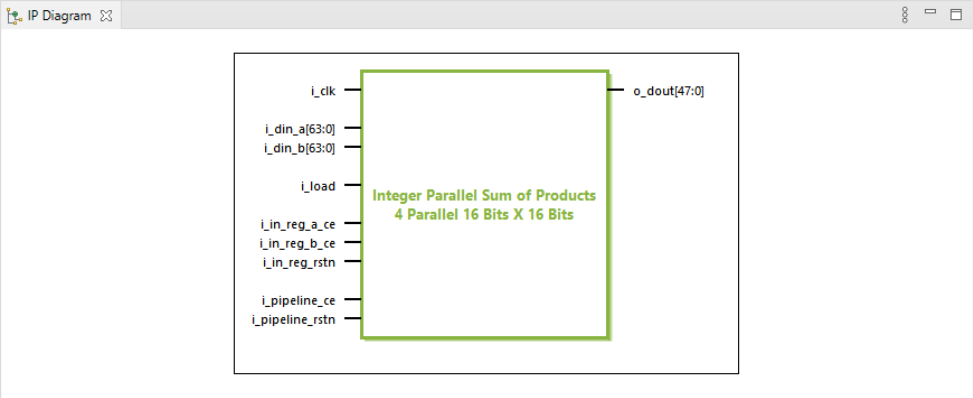


***Figure 19 • Four 16 × 16 Signed Multiplier Sum of Products I/O***

# Chapter 8 : Integer Parallel Sum of Squares Soft IP

## Description

The Integer Parallel Sum of Squares soft IP core configures multiple parallel integer square ($n^2$) multipliers with a single summed result. This soft IP core is implemented with the MLP primitive which contains an array of integer multipliers and associated adders. Up to 32 parallel multipliers, ranging from 3 × 3 to 16 × 16 bits can be configured. The multipliers support both signed and unsigned arithmetic. The final output can optionally be accumulated.

## Configuration

The integer parallel sum of squares soft IP configurator has the following options:

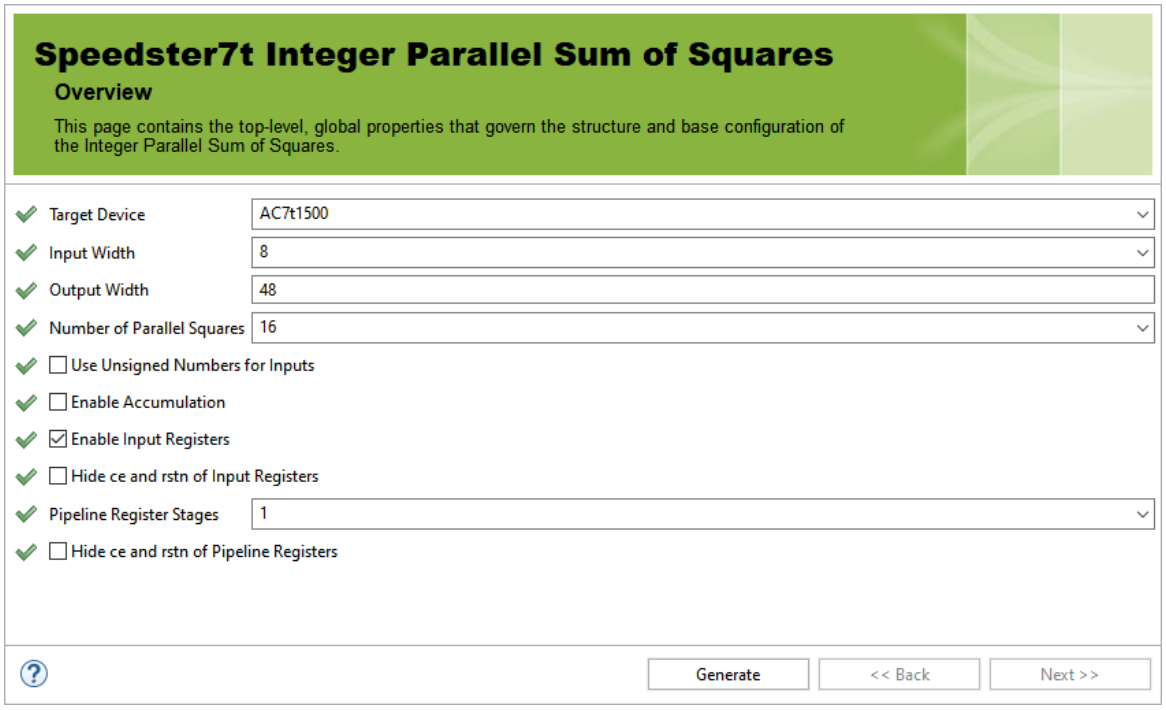


***Figure 20 • Integer Parallel Sum of Squares Configurator***

***Table 16 • Configuration Options***

| Name | Default | Range | Description |
|---|---|---|---|
| Target Device | AC7t1500 | All Speedster7t devices | Set to match the target device of the project. |
| Input Width | 8 | 3, 4, 5, 6, 7, 8 or 16 | Width of the input to be squared. |
| Number of Parallel Squares | 8 | 1 to 32 | The number of parallel squaring multipliers to be implemented and their results summed. The maximum number of multipliers is determined by the Input Width. Refer to the Maximum Number of Multipliers Per Input Width table, for details. |
| Use Unsigned Numbers for Input | Off | On, Off | When set, configures the input to use unsigned numbers. The input is signed by default. |
| Enable Input Registers | Off | On, Off | When set, enables a register stage for the input. Adds a cycle of latency to all results. Enabling the input registers adds these inputs to the resultant soft IP core:<br>• `i_in_reg_ce`<br>• `i_in_reg_rstn` |
| Hide `ce` and `rstn` of Input Registers | No | Yes, No | If selected, the `i_in_reg_ce` and `i_in_reg_rstn` inputs are automatically tied high (`1'b1`). |
| Enable Accumulator | Off | On, Off | Enables accumulation:<br>• Off – no accumulation<br>• On – accumulation: `o_dout` is the accumulated value. The start of accumulation is signaled by asserting `i_load = 1`.<br>When Accumulation is enabled, these inputs are added to the resultant soft IP core:<br>• `i_load`<br>• `i_pipeline_ce`<br>• `i_pipeline_rstn`<br>When `i_load` is high, the previous value of the internal accumulation register is ignored, and the new value is stored. The output is then set to this value. When `i_load` is low, the old and new values are added, and the sum is stored.<br>The `i_load` signal is internally pipelined to have the same latency as the input. If a set of inputs start a new accumulation, `i_load` must be high when those inputs are presented. If accumulation is not enabled, the `i_load` signal is ignored. |

| Name | Default | Range | Description |
|---|---|---|---|
| Pipeline Register Stages | 0 | 0, 1 or 2 | Adds pipeline register stages to the multiplication process. Enabling pipeline register stages improves timing performance at the cost of an additional cycle of latency per stage. When any pipeline stages are enabled, these inputs are added to the resultant soft IP core:<br>• `i_pipeline_ce`<br>• `i_pipeline_rstn` |
| Hide `ce` and `rstn` of Pipeline Registers | No | Yes, No | If selected, the `i_pipeline_ce` and `i_pipeline_rstn` inputs are automatically tied high (`1'b1`). |
| Output Width (1) | 48 | 3 to 48 | Width of the data output. By default, this value is set to 48 bits. This value can be reduced if required. |

**Table Notes**

1. Ensure that **Output Width** is sufficient to represent the maximum result that can be accumulated. In the event of overflow, the higher order bits of any result are truncated. When accumulation is enabled, it might be necessary to increase the data output width to account for the growth in the result over multiple accumulation cycles.

   The maximum result width can be calculated as:

   $(2 \times \textbf{Input Width}) + \lceil \log 2\ (\textbf{Number of Parallel Squares} \times\ \textbf{Number of Accumulation Cycles}) \rceil$

### *Table 17 • Maximum Number of Multipliers Per Input Width*

| | Wide Mode(1) | | Normal Mode | |
|---|---|---|---|---|
| Input Width | Max Signed Multiplications | Max Unsigned Multiplications | Max Signed Multiplications | Max Unsigned Multiplications |
| 3 | 32 | 32 | 24 | 16 |
| 4 | 32 | 16 | 16 | 12 |
| 5 | 16 | 16 | 12 | 12 |
| 6 | 16 | 16 | 12 | 10 |
| 7 | 16 | 16 | 10 | 8 |
| 8 | 16 | 16 | 8 | 8 |
| 16 | – | – | 4 | 4 |

| | Wide Mode[1] | | Normal Mode | |
|---|---|---|---|---|
| Input Width | Max Signed Multiplications | Max Unsigned Multiplications | Max Signed Multiplications | Max Unsigned Multiplications |

**Table Notes**

1. When required, based on the selected number of multipliers, the MLP72 is automatically configured in wide mode, which blocks the adjacent BRAM72K from being used.

***Table 18 • Ports***

| Name | Direction | Description |
|---|---|---|
| i_clk | Input | Clock input to drive the (optional) registers and accumulator. |
| i_din[(*data width* – 1):0] | Input | Packed (page 38) vector of data input to the squaring multipliers where *data width* = **Input Width** × **Number of Parallel Squares**. |
| i_in_reg_ce | Input | (Optional) Clock enable for i_din. Present when **Enable Input Registers** is set to **On**. |
| i_in_reg_rstn | Input | (Optional) Synchronous active-low reset for input register. Present when **Enable Input Registers** is set to **On**. |
| i_pipeline_ce | Input | (Optional) Clock enable for accumulation and pipeline registers. Present when **Enable Accumulation** is set to **On** or when **Pipeline Register Stages** is greater than 0. |
| i_pipeline_rstn | Input | (Optional) Synchronous active-low reset for accumulation and pipeline registers. Present when **Enable Accumulation** is set to **On** or when **Pipeline Register Stages** is greater than 0. |
| i_load [(1)] | Input | (Optional) When asserted to 1'b1, resets the accumulator to $i\_din^2$ ignoring the previous value. Present when **Enable Accumulator** is set to **On**. |
| o_dout[(Output Width – 1):0] | Output | Output bus consisting of the sum of squares from all of the multipliers in parallel. |

**Table Notes**

1. This signal is internally pipelined to have the same latency as i_din.

# Input Packing

Inputs are packed in a single input vector.

```
din(i) = i_din[i * int_size +: int_size];
```

# Examples

The following example shows the integer parallel sum of squares configured for four signed 16 bit inputs, with input registers, accumulation and a single internal pipeline stage:

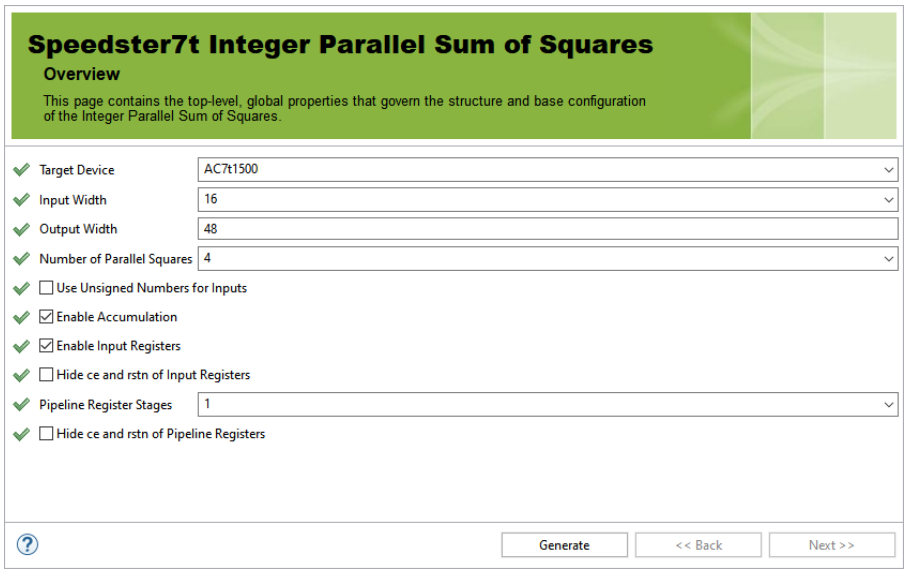


***Figure 21 • Four 16 × 16 Signed Multiplier Sum of Squares Configuration***

The following figure shows the IP diagram for this configuration:

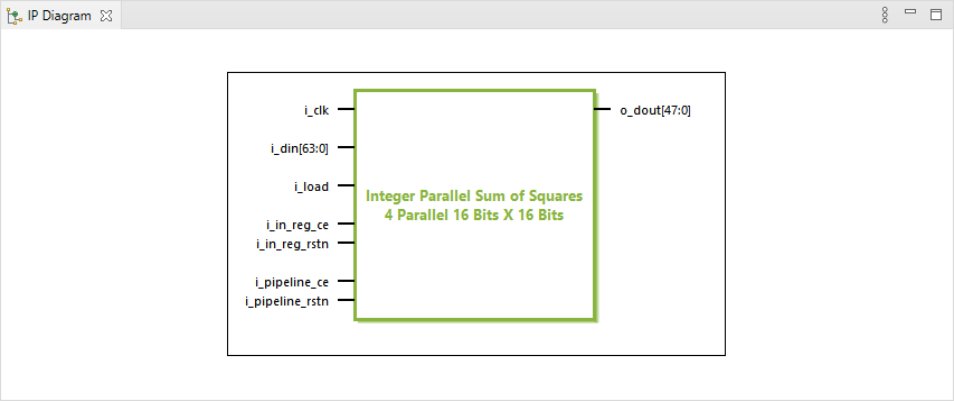


***Figure 22 • Four 16 × 16 Signed Multiplier Sum of Squares I/O***

# Chapter 9 : Integer RLB Multiplier Soft IP

## Description

The Integer Reconfigurable Logic Block (RLB) Multiplier soft IP core configures a two-input integer multiplier using RLB (logic) based multipliers. The multiplier can also support optional result accumulation. The configurator supports sizes of up to 9 × 9, with signed inputs and outputs.

## Configuration

The integer RLB multiplier soft IP configurator has the following options:

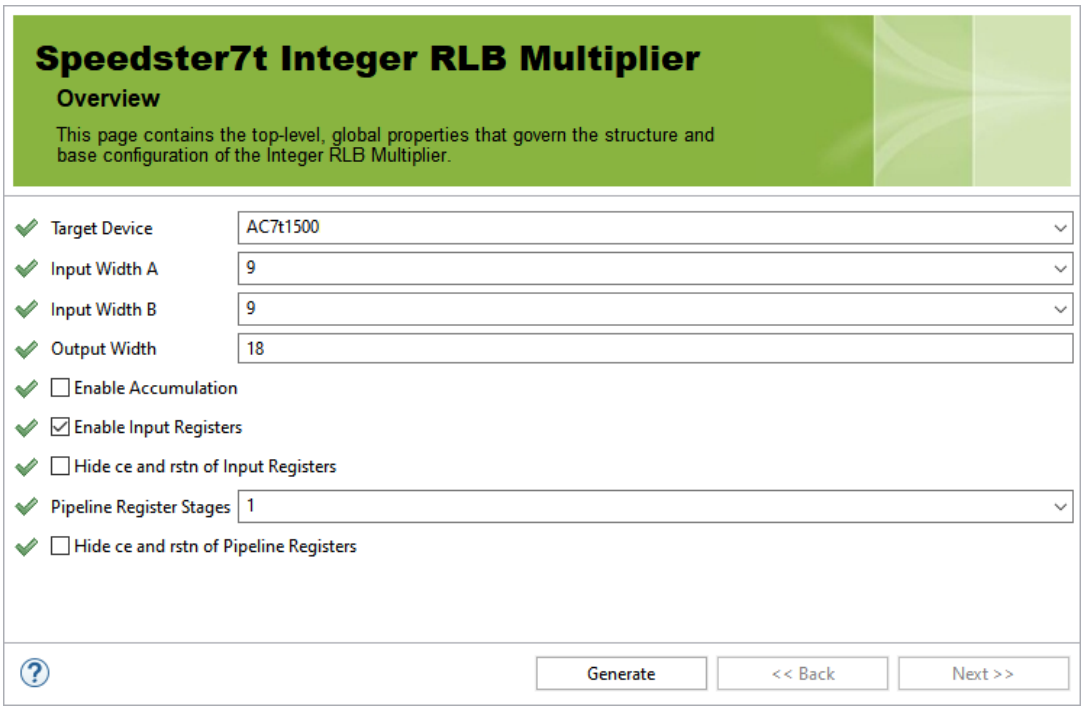


***Figure 23 • Integer RLB Multiplier Soft IP Configurator***

***Table 19 • Configuration Options***

| Name | Default | Range | Description |
|---|---|---|---|
| Target Device | AC7t1500 | All Speedster7t devices | Set to match the target device of the project. |
| Input Width A | 9 | 3 to 9 | Width of 'A' inputs. |
| Input Width B | 9 | 3 to 9 | Width of 'B' inputs. |
| Enable Input Registers | Off | On, Off | When set, enables a register stage for both 'A' and 'B' inputs. Adds a cycle of latency to all results. Enabling the input registers adds these inputs to the resultant soft IP core:<br>• `i_in_reg_a_ce`<br>• `i_in_reg_b_ce`<br>• `i_in_reg_rstn` |
| Hide `ce` and `rstn` of Input Registers | No | Yes, No | If selected, the `i_in_reg_a_ce`, `i_in_reg_b_ce`, and `i_in_reg_rstn` inputs are automatically tied high (`1'b1`). |
| Enable Accumulator | Off | On, Off | Enables accumulation:<br>• Off – no accumulation<br>• On – accumulation: `o_dout` is the accumulated value. The start of accumulation is signaled by asserting `i_load = 1`.<br>When Accumulation is enabled, these inputs are added to the resultant soft IP core:<br>• `i_load`<br>• `i_pipeline_ce`<br>• `i_pipeline_rstn`<br>When `i_load` is high, the previous value of the internal accumulation register is ignored, and the new value is stored. The output is then set to this value. When `i_load` is low, the old and new values are added, and the sum is stored. The output is `o_dout = (i_din_a × i_din_b) + dout_prev`.<br>The `i_load` signal is internally pipelined to have the same latency as the input. If a set of inputs start a new accumulation, `i_load` must be high when those inputs are presented. If accumulation is not enabled, the `i_load` signal is ignored. |
| Pipeline Register Stages | 0 | 0, 1, 2 | Adds pipeline register stages through the multiplication process. Enabling pipeline registers improves timing performance at the cost of an additional cycle of latency for each stage enabled. When any pipeline stages are enabled, these inputs are added to the resultant soft IP core:<br>• `i_pipeline_ce`<br>• `i_pipeline_rstn` |

| Name | Default | Range | Description |
|---|---|---|---|
| Hide ce and rstn of Pipeline Registers | No | Yes, No | If selected, the i_pipeline_ce and i_pipeline_rstn inputs are automatically tied high (1'b1). |
| Output Width (1) | 18 | 2 to 48 | Width of the data output. Automatically updated by the configurator when Input Width A/B is updated. In addition, the value can be modified to match requirements. |

**Table Notes**

1. When accumulation is enabled, it might be necessary to increase the data output width to account for the growth in the result over multiple accumulation cycles. The maximum result width can be calculated as:

   (Input Width A + Input Width B) + ⌈ log 2 ( Number of Accumulation Cycles )⌉

### *Table 20 • Ports*

| Name | Direction | Description |
|---|---|---|
| i_clk | Input | Clock input to drive the (optional) registers and accumulator. |
| i_din_a[(Input Width A – 1):0] | Input | 'A' data input to the multiplier. |
| i_din_b[(Input Width B – 1):0] | Input | 'B' data input to the multiplier. |
| i_in_reg_a_ce | Input | (Optional) Clock enable for i_din_a. Present when **Enable Input Registers** is set to **On**. |
| i_in_reg_b_ce | Input | (Optional) Clock enable for i_din_b. Present when **Enable Input Registers** is set to **On**. |
| i_in_reg_rstn | Input | (Optional) Synchronous active-low reset for the input registers. Present when **Enable Input Registers** is set to **On**. |
| i_pipeline_ce | Input | (Optional) Clock enable for the pipeline and accumulator registers. Present when **Enable Accumulation** is set to **On** or when **Pipeline Register Stages** is greater than 0. |
| i_pipeline_rstn | Input | (Optional) Synchronous active-low reset for the pipeline and accumulator registers. Present when **Enable Accumulation** is set to **On** or when **Pipeline Register Stages** is greater than 0. |
| i_load (1) | Input | (Optional) When asserted to 1'b1, resets the accumulator to i_din_a × i_din_b ignoring the previous value. Present when **Enable Accumulator** is set to **On**. |
| o_dout[(Output Width – 1):0] | Output | Result of multiplication and accumulation. |

| Name | Direction | Description |
|---|---|---|

**Table Notes**

1. This signal is internally pipelined to have the same latency as `i_din_a` and `i_din_b`.

# Examples

The following example shows the integer multiplier configured for signed 9 × 9 inputs with input registers, accumulation and a single pipeline stage:

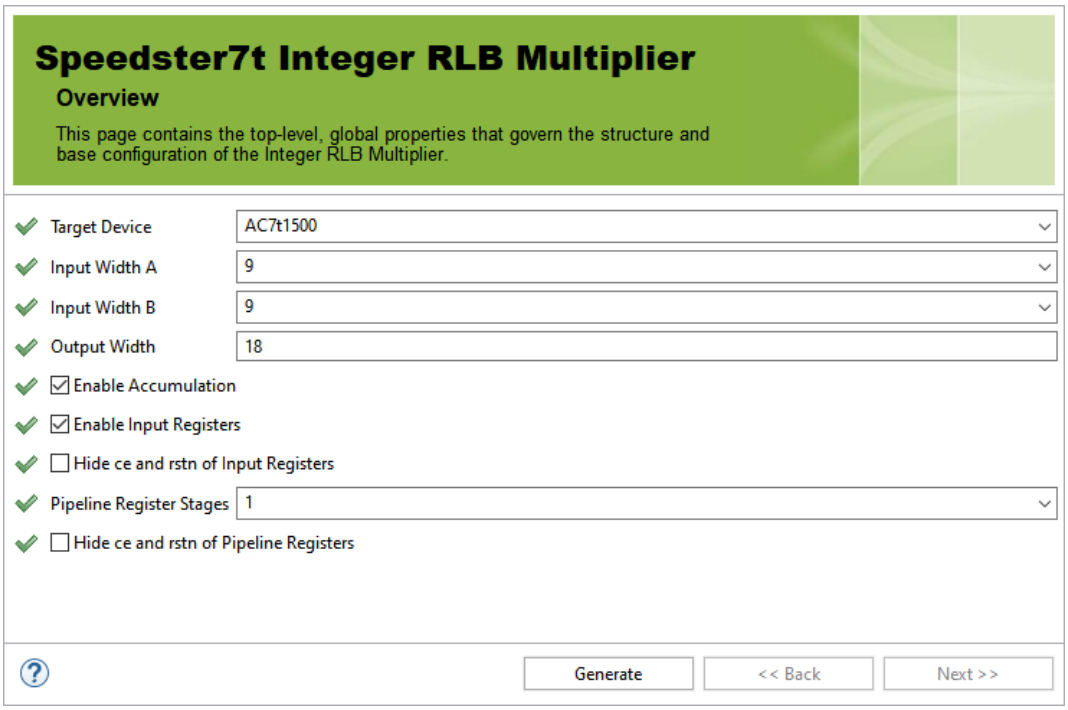


***Figure 24 • 9 × 9 Signed Integer RLB Multiplier Configuration***

The following figure shows the IP diagram for this configuration:

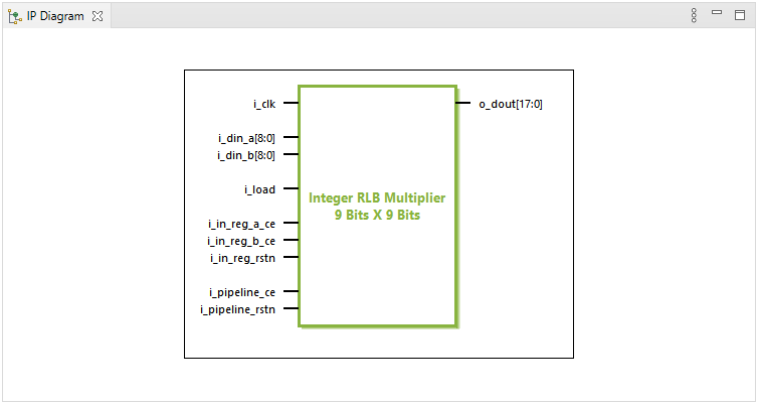


***Figure 25 • 9 × 9 Signed Integer RLB Multiplier I/O***

# Chapter 10 : Floating-Point Adder

## Description

The Floating Point Adder soft IP core configures a two-input floating-point adder with optional result accumulation. The adder instantiates the ACX_FP_ADD module shown in the ***Speedster7t Component Library User Guide*** **(UG086)**[3].

## Configuration

The floating point adder soft IP configurator has the following options:

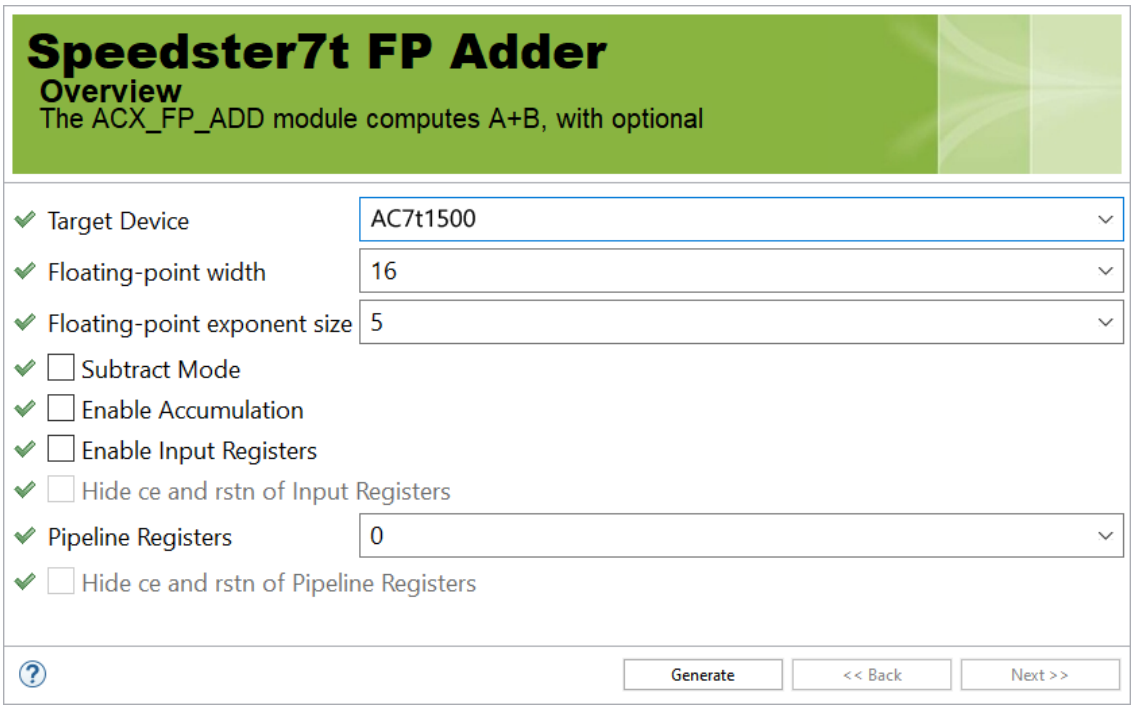


***Figure 26 • Floating Point Adder Soft IP Configurator***

3 https://www.achronix.com/documentation/speedster7t-component-library-user-guide-ug086

***Table 21 • Configuration Options***

| Name | Default | Range | Description |
|---|---|---|---|
| Target Device | AC7t1500 | All Speedster7t devices | Set to match the target device of the project. |
| Floating-point width | 16 | 16, 24 | Width of floating point number. Supports fp24, fp16, and fp16e8. |
| Floating-point exponent size | 5 | 5, 8 | Size of floating-point exponent. For floating point width of 16, the exponent size can be either 5(fp16) or 8(fp16e8). For floating point width of 24, the exponent size must be 8. |
| Subtract Mode | 0 | 0, 1 | Enable subtract mode:<br>• 0 – compute `i_din_a + i_din_b`.<br>• 1 – compute `i_din_a - i_din_b`. |
| Enable Accumulation | 0 | 0, 1 | Enable accumulation:<br>• 0 – no accumulation: `o_dout = i_din_a ± i_din_b` (determined by the `subtract` parameter).<br>• 1 – accumulation: `o_dout` is the accumulated value. The start of accumulation is signaled by asserting `i_load = 1`.<br>When `i_load` is high, the previous value of the internal accumulation register is ignored, and the new value is stored. The output is then set to this value. When `i_load` is low, the old and new values are added. The output is this sum which is stored.<br>The `i_load` signal is internally pipelined to have the same latency as the input. If a set of inputs start a new accumulation, then `i_load` must be high when those inputs are presented. If accumulation is not enabled, then the `i_load` signal is ignored. |
| Enable Input Registers | 0 | 0, 1 | Enable input registers:<br>• 0 – no input registers.<br>• 1 – `i_din_a` and `i_din_b` are registered. The input registers are controlled by the `i_in_reg_a_ce`, `i_in_reg_b_ce`, and `i_in_reg_rstn` inputs. Enabling the input registers adds one cycle of latency. |
| Hide ce and rstn of Input registers | 0 | 0, 1 | If selected, the `i_in_reg_a_ce`, `i_in_reg_b_ce`, and `i_in_reg_rstn` inputs are automatically tied high (1'b1). |
| Pipeline Registers | 0 | 0–5 | The number of pipeline registers, not counting the input register. The total latency is `pipeline_regs + in_reg_enable`. |

| Name | Default | Range | Description |
|---|---|---|---|
| Hide ce and rstn of Pipeline Registers | 0 | 0, 1 | If selected, the i_pipeline_ce and i_pipeline_rstn inputs are automatically tied high (1'b1). |

# Ports

***Table 22 • Floating Point Adder GUI-Generated Pin Descriptions***

| Name | Direction | Description |
|---|---|---|
| i_clk | Input | Clock input. Used by the (optional) registers and accumulator. |
| i_din_a[(fp_size-1):0] | Input | 'A' data input to adder. |
| i_din_b[(fp_size-1):0] | Input | 'B' data input to adder. |
| i_in_reg_a_ce | Input | If **Enable Input Registers** is checked, i_in_reg_a_ce is the clock enable for i_din_a.<br>If **Enable Input Registers** is not enabled, i_in_reg_a_ce does not exist. |
| i_in_reg_b_ce | Input | If **Enable Input Registers** is checked, i_in_reg_b_ce is the clock enable for i_din_b.<br>If **Enable Input Registers** is not enabled, i_in_reg_b_ce does not exist. |
| i_in_reg_rstn | Input | If **Enable Input Registers** is checked, i_in_reg_rstn is the synchronous active-low reset for input registers.<br>If **Enable Input Registers** is not enabled, i_in_reg_rstn does not exist. |
| i_pipeline_ce | Input | If **Accumulation** is enabled, i_pipeline_ce is the clock enable for the accumulator register.<br>If **Pipeline Registers** are selected (greater than 0), i_pipeline_ce is the clock enable for pipeline registers.<br>If neither is enabled, i_pipeline_ce does not exist. |
| i_pipeline_rstn | Input | If **Accumulation** is enabled, i_pipeline_rstn is the synchronous, active-low reset for the accumulator register.<br>If **Pipeline Registers** are selected (greater than 0), i_pipeline_rstn is the synchronous, active-low reset for pipeline registers.<br>If neither is enabled, i_pipeline_rstn does not exist. |
| i_load | Input | If **Accumulation** is enabled, i_load resets the accumulator to i_din_a ± i_din_b, ignoring the previous value. This signal is internally pipelined to have the same latency as i_din_a ± i_din_b.<br>If **Accumulation** is not enabled, i_load does not exist. |

| Name | Direction | Description |
|---|---|---|
| o_dout[(fp_size-1):0] | Output | Result of addition and accumulation. |
| o_status[1:0] | Output | Error status of o_dout:<br>• 2'b00 – normal.<br>• 2'b01 – result is ± 0.0.<br>• 2'b10 – Result is ± infinity.<br>• 2'b11 – last operation had underflow, the result is ± 0.0. |

## Files

The configurator produces a .v (verilog ) or a .vhd (VHDL) wrapper file based on the chosen RTL model which must be instantiated in the user design. By default, the file is written to the current working directory of the project, but a different location can be chosen by clicking **Browse...** as shown in the following figure.

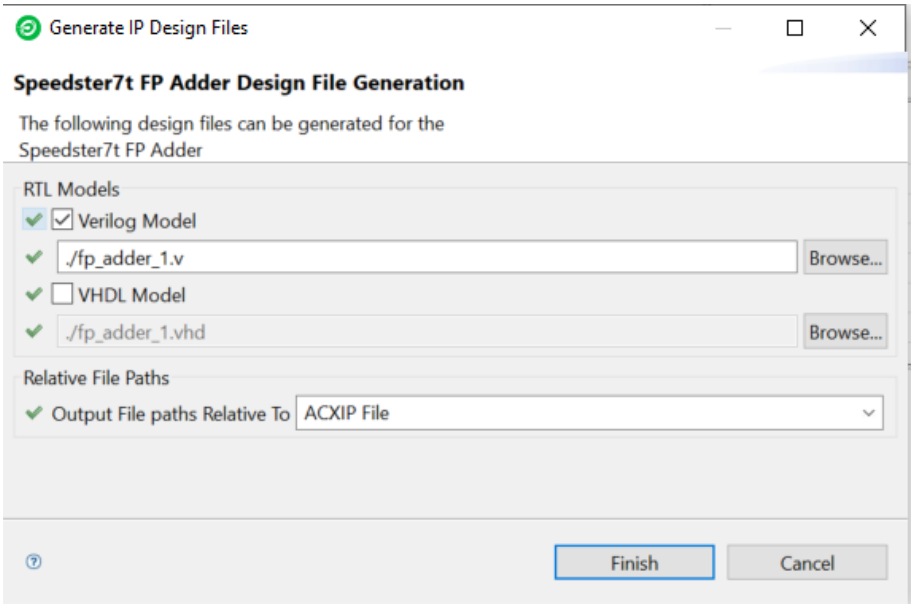


***Figure 27 • FP Adder Design File Generation***

# Examples

The following example shows the floating point adder configured for 16-bit floating point inputs with input registers, accumulation, two pipeline stages, and 16-bit output:

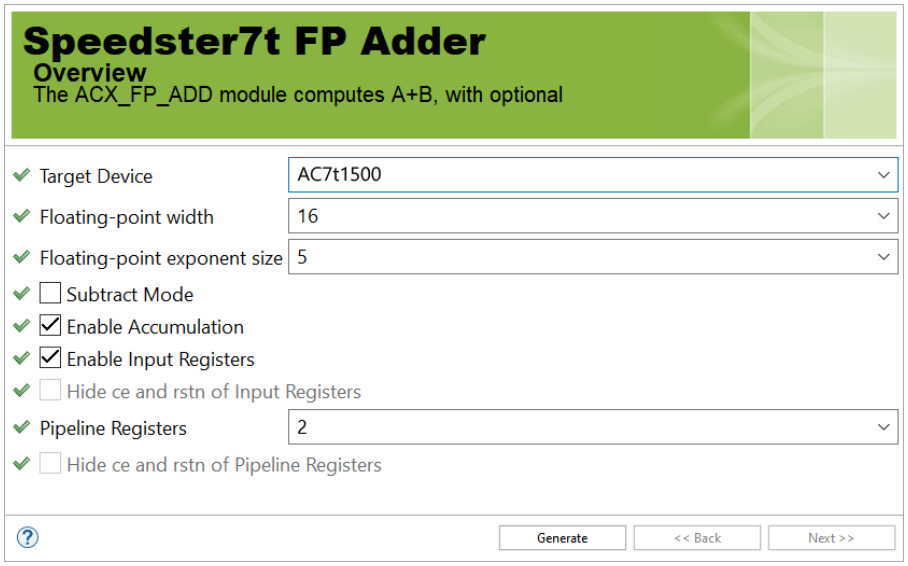


***Figure 28 • Two Pipeline Stage Floating Point Adder Configuration***

The following figure shows the IP diagram for this configuration:

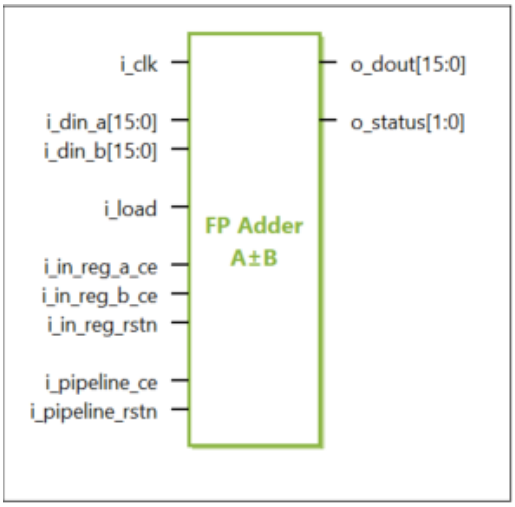


***Figure 29 • Two Pipeline Stage Floating Point Adder IP Diagram***

# Chapter 11 : Floating-Point Multiplier

## Description

The Floating Point Multiplier soft IP core configures a two-input floating-point multiplier with optional result accumulation. The multiplier instantiates the ACX_FP_MULT module shown in the ***Speedster7t Component Library User Guide*** **(UG086)**[4].

## Configuration

The Floating Point Multiplier soft IP configurator has the following options:

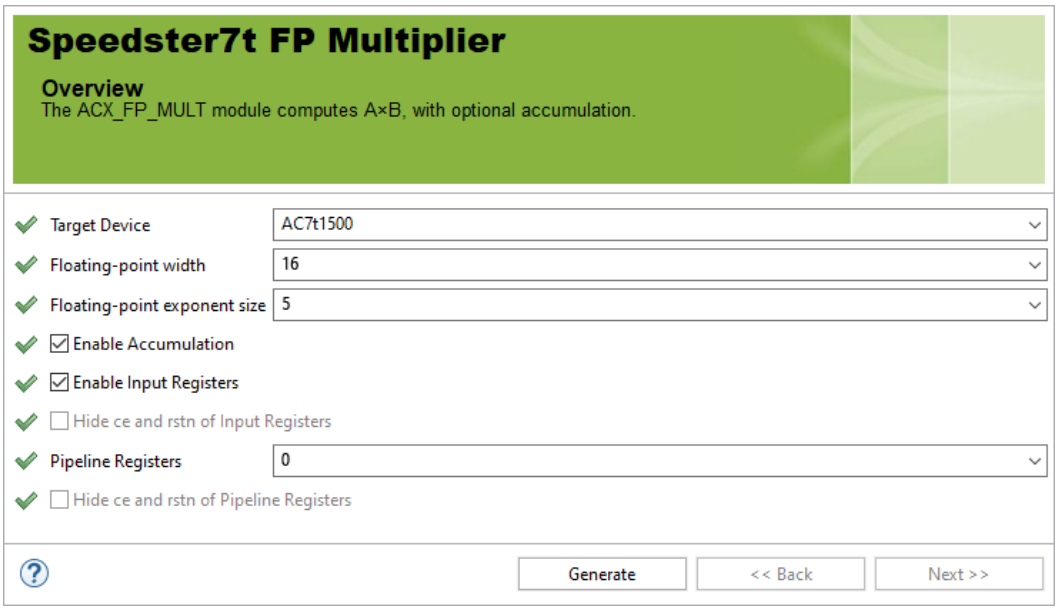


***Figure 30 • Floating Point Multiplier Soft IP Configurator***

4 https://www.achronix.com/documentation/speedster7t-component-library-user-guide-ug086

***Table 23 • Configuration Options***

| Name | Default | Range | Description |
|---|---|---|---|
| Target Device | AC7t1500 | All Speedster7t devices | Set to match the target device of the project. |
| Floating-point width | 16 | 16, 24 | Width of floating-point number. Supports fp24, fp16, and fp16e8. |
| Floating point exponent size | 5 | 5, 8 | Size of floating-point exponent. For floating point width of 16, the exponent size can either be 5(fp16) or 8(fp16e8). For floating point width of 24, the exponent size must be 8. |
| Enable Accumulation | 0 | 0, 1 | Enable accumulation:<br>• 0 – no accumulation: `o_dout = i_din_a × i_din_b`.<br>• 1 – accumulation: `o_dout` is the accumulated value. The start of accumulation is signaled by asserting `i_load = 1`.<br>When `i_load` is high, the previous value of the internal accumulation register is ignored, and the new value is stored. The output is then set to this value. When `i_load` is low, the old and new values are added, and the sum is stored. The output is `o_dout = (i_din_a × i_din_b) + dout_prev`.<br>The `i_load` signal is internally pipelined to have the same latency as the input. If a set of inputs start a new accumulation, `i_load` must be high when those inputs are presented. If accumulation is not enabled, the `i_load` signal is ignored. |
| Enable Input Registers | 0 | 0, 1 | Enable input registers:<br>• 0 – no input registers.<br>• 1 – `i_din_a` and `i_din_b` are registered. The input registers are controlled by the `i_in_reg_a_ce`, `i_in_reg_b_ce`, and `i_in_reg_rstn` inputs. Enabling the input registers adds one cycle of latency. |
| Hide `ce` and `rstn` of Input registers | 0 | 0, 1 | If selected, the `i_in_reg_a_ce`, `i_in_reg_b_ce`, and `i_in_reg_rstn` inputs are automatically tied high (1'b1). |
| Pipeline registers | 0 | 0–4 | The number of pipeline registers, not counting the input register. The total latency is `pipeline_regs + in_reg_enable`. |
| Hide `ce` and `rstn` of Pipeline registers | 0 | 0, 1 | If selected, `i_pipeline_ce` and `i_pipeline_rstn` inputs are automatically tied high (1'b1). |

# Ports

***Table 24 • Floating Point Multiplier GUI-Generated Pin Descriptions***

| Name | Direction | Description |
|---|---|---|
| i_clk | Input | Clock input, used for the (optional) registers and accumulator. |
| i_din_a[(fp_size-1):0] | Input | 'A' data input to multiplier. |
| i_din_b[(fp_size-1):0] | Input | 'B' data input to multiplier. |
| i_in_reg_a_ce | Input | If **Enable Input Registers** is enabled, i_in_reg_a_ce is the clock enable for i_din_a.<br>If **Enable Input Registers** is not enabled, i_in_reg_a_ce does not exist. |
| i_in_reg_b_ce | Input | If **Enable Input Registers** is enabled, i_in_reg_b_ce is the clock enable for i_din_b.<br>If **Enable Input Registers** is not enabled, i_in_reg_b_ce does not exist. |
| i_in_reg_rstn | Input | If **Enable Input Registers** is enabled, i_in_reg_rstn is the synchronous active-low reset for input registers.<br>If **Enable Input Registers** is not enabled, i_in_reg_rstn doesn't exist. |
| i_pipeline_ce | Input | If **Accumulation** is enabled, i_pipeline_ce is the clock enable for the accumulator register.<br>If **Pipeline Registers** are selected (greater than 0), i_pipeline_ce is the clock enable for pipeline registers.<br>If neither is enabled, i_pipeline_ce does not exist. |
| i_pipeline_rstn | Input | If **Accumulation** is enabled, i_pipeline_rstn is the synchronous, active-low reset for the accumulator register.<br>If **Pipeline Registers** are selected (greater than 0), i_pipeline_rstn is the synchronous, active-low reset for pipeline registers.<br>If neither is enabled, i_pipeline_rstn does not exist. |
| i_load | Input | If **Accumulation** is enabled, i_load resets the accumulator to i_din_a × i_din_b, ignoring the previous value. This signal is internally pipelined to have the same latency as i_din_a × i_din_b.<br>If **Accumulation** is not enabled, i_load does not exist. |
| o_dout[(fp_size-1):0] | Output | Result of multiplication and optional accumulation. |
| o_status[1:0] | Output | Error status of o_dout:<br>• 2'b00 – normal.<br>• 2'b01 – result is ± 0.0.<br>• 2'b10 – result is ± infinity.<br>• 2'b11 – last operation had underflow. The result is ± 0.0. |

## Files

The configurator produces a `.v` (verilog) or `.vhd` (VHDL) wrapper file based on the chosen RTL model which must be instantiated in the the user design. By default, the file is written to the current working directory of the project, but a different location can be chosen by clicking **Browse...** as shown in the following figure.

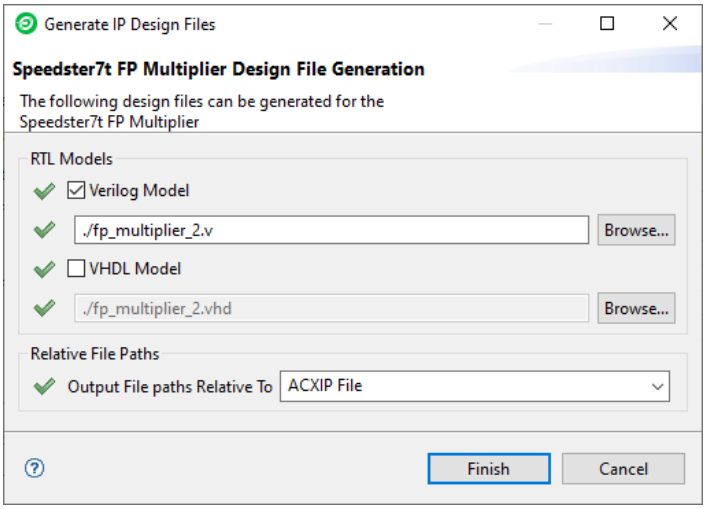


***Figure 31 • FP Multiplier Design File Generation***

## Examples

The following example shows the floating point multiplication configured for 24-bit floating point inputs with input registers, accumulation, two pipeline stages, and 24-bit output:

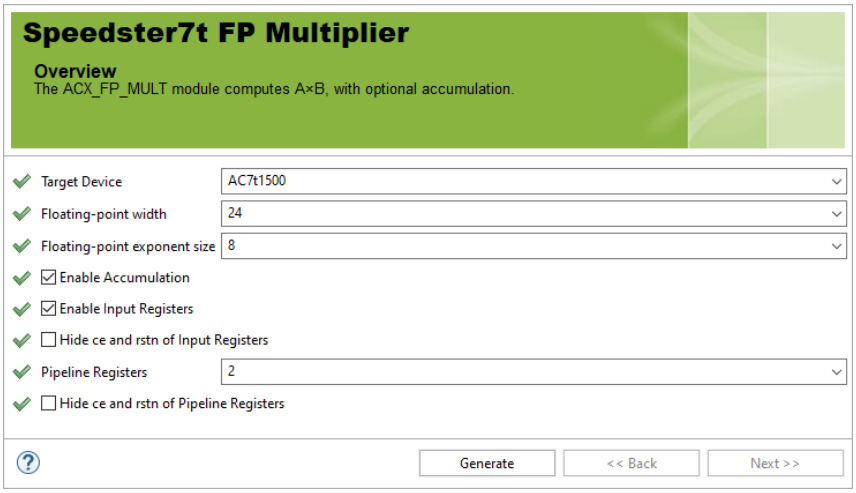


***Figure 32 • Two Pipeline Stage Floating Point Multiplier Configuration***

The following figure shows the IP diagram for this configuration:

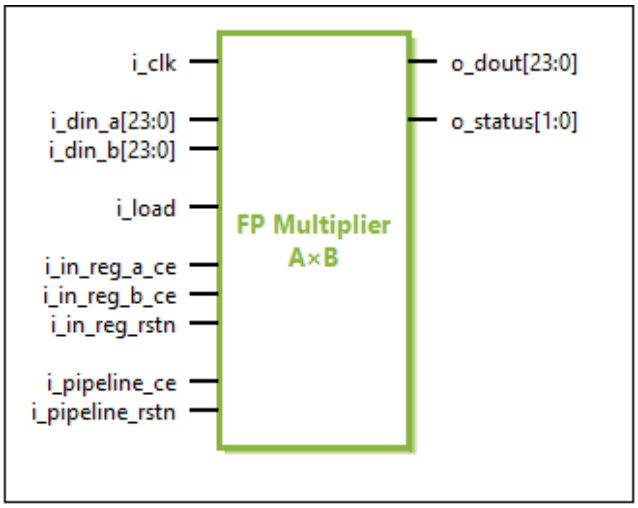


***Figure 33 • Two Pipeline Stage Floating Point Multiplier IP Diagram***

# Chapter 12 : Floating-Point Multiplier Plus

## Description

The Floating Point Multiplier Plus soft IP core configures a two-input multiplier with an adder (A × B + C), with optional result accumulation. The multiplier instantiates the ACX_FP_MULT_PLUS module shown in the *Speedster7t Component Library User Guide* (UG086)[5].

## Configuration

The Floating Point Multiplier Plus soft IP configurator has the following options:

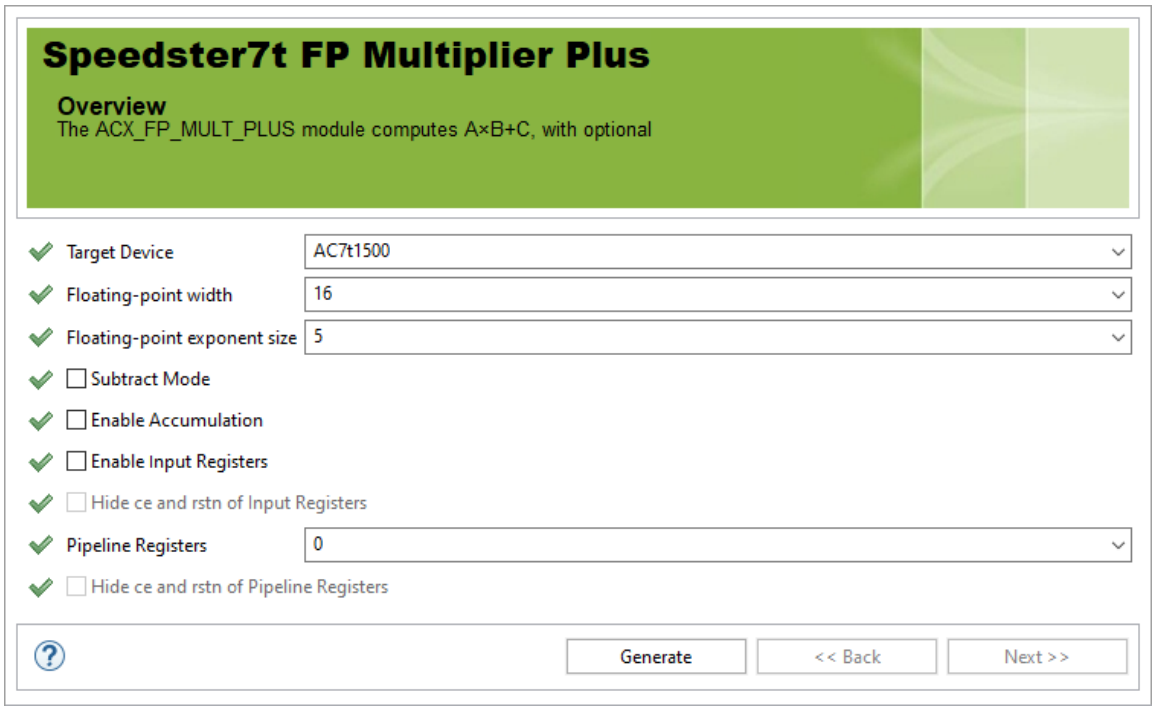


***Figure 34 • Floating Point Multiplier Plus Soft IP Configurator***

5 https://www.achronix.com/documentation/speedster7t-component-library-user-guide-ug086

***Table 25 • Configuration Options***

| Name | Default | Range | Description |
|---|---|---|---|
| Target Device | AC7t1500 | All Speedster7t devices | Set to match the target device of the project. |
| Floating Point width | 16 | 16, 24 | Width of floating-point number. Supports fp24, fp16, and fp16e8. |
| Floating point exponent size | 5 | 5, 8 | Size of floating-point exponent. For floating point width of 16, the exponent size can be either 5 (fp16) or 8 (fp16e8). For floating point width of 24, the exponent size must be 8. |
| Subtract Mode | 0 | 0, 1 | Subtract mode:<br>• 0 – compute `i_din_a × i_din_b + i_din_c`.<br>• 1 – compute `i_din_a × i_din_b – i_din_c`. |
| Enable Accumulation | 0 | 0, 1 | Enable accumulation:<br>• 0 – no accumulation: `o_dout = i_din_a × i_din_b ± i_din_c` (determined by the `subtract` parameter).<br>• 1 – accumulation: `o_dout` is the accumulated value. The start of accumulation is signaled by asserting `i_load = 1`.<br>When `i_load` is high, the previous value of the internal accumulation register is ignored, and the new value is stored. The output is set to this value. When `i_load` is low, the old and new values are added, and the sum is stored. The output is `o_dout = (i_din_a × i_din_b ± i_din_c ) + dout_prev`.<br>The `i_load` signal is internally pipelined to have the same latency as the input. If a set of inputs starts a new accumulation, then `i_load` must be high when those inputs are presented. If accumulation is not enabled, then the `i_load` signal is ignored. |
| Enable Input Registers | 0 | 0, 1 | Enable input registers:<br>• 0 – no input registers.<br>• 1 – `i_din_a`, `i_din_b`, and `i_din_c` are registered. The input registers are controlled by the `i_in_reg_a_ce`, `i_in_reg_b_ce`, `i_in_reg_c_ce`, and `i_in_reg_rstn` inputs. Enabling the input registers adds one cycle of latency. |
| Hide `ce` and `rstn` of input registers | 0 | 0, 1 | If selected, the `i_in_reg_a_ce`, `i_in_reg_b_ce`, `i_in_reg_c_ce`, and `i_in_reg_rstn` inputs are automatically tied high (1'b1). |

| Name | Default | Range | Description |
|---|---|---|---|
| Pipeline registers | 0 | 0–5 | The number of pipeline registers, not counting the input register. The total latency is `pipeline_regs + in_reg_enable`. |
| Hide `ce` and `rstn` of pipeline registers | 0 | 0, 1 | If selected, the `i_pipeline_ce` and `i_pipeline_rstn` inputs are automatically tied high (1'b1). |

# Ports

***Table 26 • Floating Point Multiplier Plus GUI-Generated Pin Descriptions***

| Name | Direction | Description |
|---|---|---|
| `i_clk` | Input | Clock input. All inputs are registered on rising edge of `i_clk`. All outputs are synchronous to `i_clk`. |
| `i_din_a[(fp_size-1):0]` | Input | 'A' data input to multiplier. |
| `i_din_b[(fp_size-1):0]` | Input | 'B' data input to multiplier. |
| `i_din_c[(fp_size-1):0]` | Input | 'C' data input direct to adder. |
| `i_in_reg_a_ce` | Input | If **Enable Input Registers** is enabled, `i_in_reg_a_ce` is the clock enable for `i_din_a`.<br>If **Enable Input Registers** is not enabled, `i_in_reg_a_ce` does not exist. |
| `i_in_reg_b_ce` | Input | If **Enable Input Registers** is enabled, `i_in_reg_b_ce` is the clock enable for `i_din_b`.<br>If **Enable Input Registers** is not enabled, `i_in_reg_b_ce` does not exist. |
| `i_in_reg_c_ce` | Input | If **Enable Input Registers** is enabled, `i_in_reg_c_ce` is the clock enable for `i_din_c`.<br>If "**Enable Input Registers**" is not enabled, `i_in_reg_c_ce` does not exist. |
| `i_in_reg_rstn` | Input | If **Enable Input Registers** is enabled, `i_in_reg_rstn` is the synchronous active-low reset for the input registers.<br>If **Enable Input Registers** is not enabled, `i_in_reg_rstn` does not exist. |
| `i_pipeline_ce` | Input | If **Accumulation** is enabled, `i_pipeline_ce` is the clock enable for the accumulator register.<br>If **Pipeline Registers** are selected (greater than 0), `i_pipeline_ce` is the clock enable for pipeline registers.<br>If neither is enabled, `i_pipeline_ce` does not exist. |

| Name | Direction | Description |
|---|---|---|
| i_pipeline_rstn | Input | If **Accumulation** is enabled, i_pipeline_rstn is the synchronous, active-low reset for the accumulator register.<br>If **Pipeline Registers** are selected (greater than 0), i_pipeline_rstn is the synchronous, active-low reset for pipeline registers.<br>If neither is enabled, i_pipeline_rstn does not exist. |
| i_load | Input | If **Accumulation** is enabled, i_load is the accumulator to i_din_a × i_din_b ± i_din_c, ignoring the previous value. This signal is internally pipelined to have the same latency as i_din_a × i_din_b ± i_din_c.<br>If **Accumulation** is not enabled, i_load does not exist. |
| o_dout[(fp_size-1):0] | Output | Result of multiplication and accumulation. |
| o_status[1:0] | Output | Error status of o_dout.<br>• 2'b00 – normal.<br>• 2'b01 – result is ± 0.0.<br>• 2'b10 – result is ± infinity.<br>• 2'b11 – last operation had underflow. The result is ± 0.0. |

# Files

The configurator produces a .v (verilog) or a .vhd (VHDL) wrapper file based on chosen RTL model which must be instantiated in the user design. By default, the file is written to the current working directory of the project, but a different location can be chosen by clicking **Browse...** as shown in the following figure.

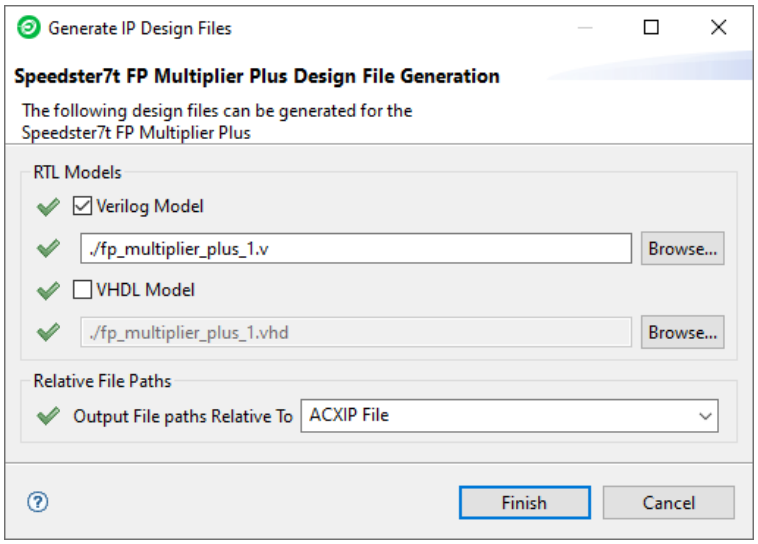


***Figure 35 • Floating Point Multiplier Plus Design File Generation***

# Examples

The following example shows the floating point multiplier plus configured for 16-bit floating point inputs in subtract mode with input registers, accumulation, four pipeline stages, and 16-bit output:

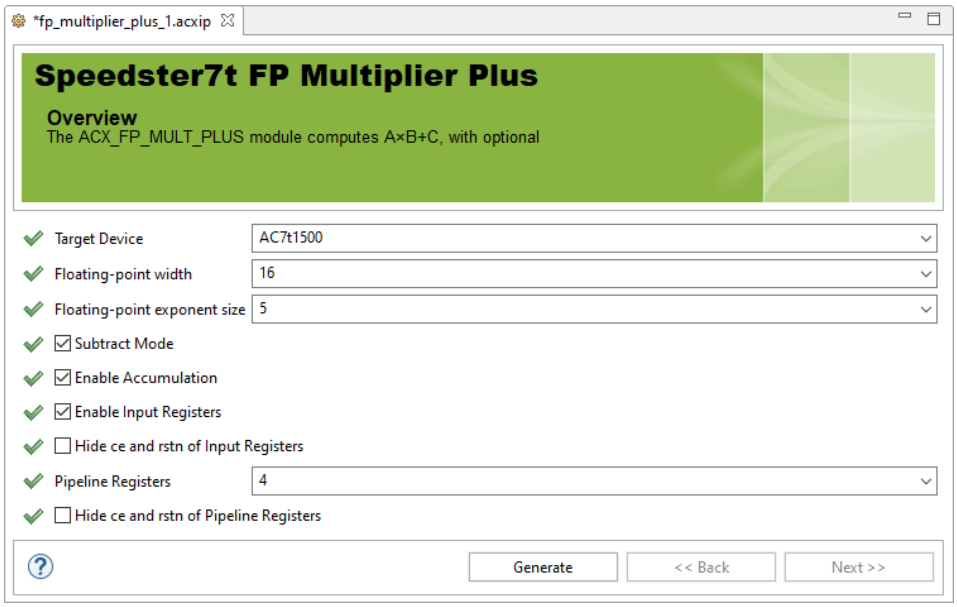


***Figure 36 • Four Pipeline Stage Floating Point Multiplier Plus Configuration***

The following figure shows the IP diagram for this configuration:

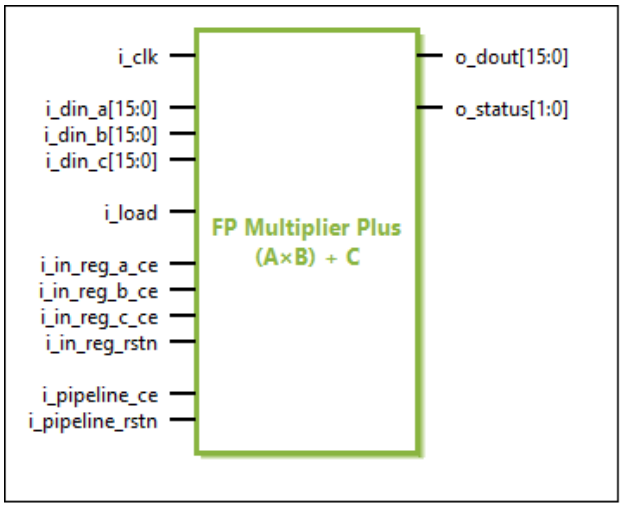


***Figure 37 • Four Pipeline Stage Floating Point Multiplier Plus IP Diagram***

# Chapter 13 : Floating-Point Parallel Multiplier

## Description

The Floating Point Parallel Multiplier soft IP core configures two parallel floating-point multipliers along with optional result accumulation. The multiplier instantiates the ACX_FP_MULT_2X module shown in the ***Speedster7t Component Library User Guide*** (UG086)[6].

## Configuration

The Floating Point parallel multiplier soft IP configurator has the following options:

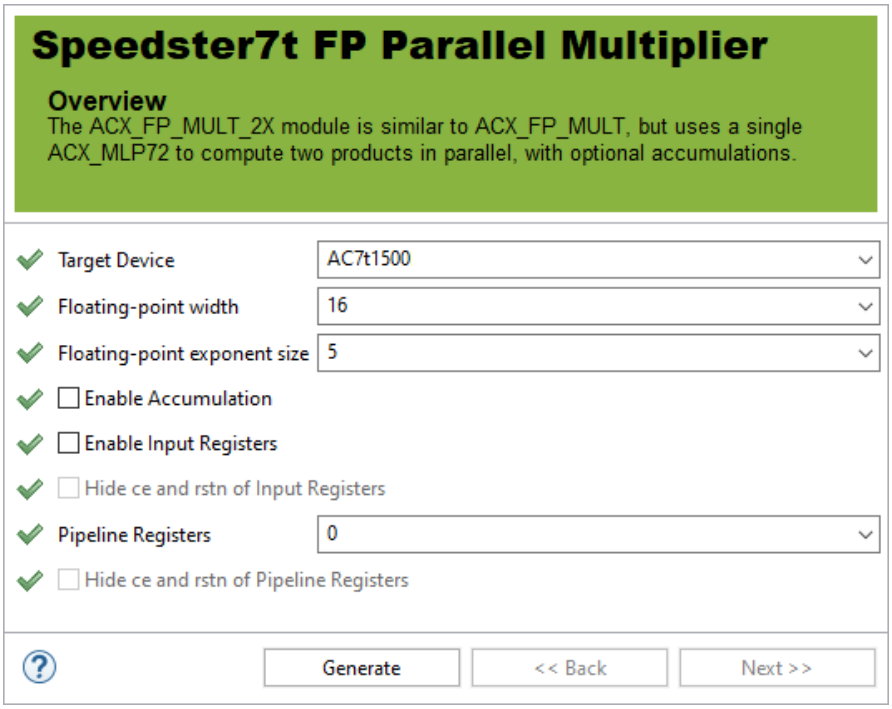


***Figure 38 • Floating Point Parallel Multiplier Soft IP Configurator***

6 https://www.achronix.com/documentation/speedster7t-component-library-user-guide-ug086

***Table 27 • Configuration Options***

| Name | Default | Range | Description |
|---|---|---|---|
| Target Device | AC7t1500 | All Speedster7t devices | Set to match the target device of the project. |
| Floating-point width | 16 | 16, 24 | Width of floating-point number. Supports fp24, fp16, and fp16e8. |
| Floating point exponent size | 5 | 5, 8 | Size of floating-point exponent. For floating point width of 16, the exponent size can either be 5(fp16) or 8(fp16e8). For floating point width of 24, the exponent size must be 8. |
| Enable Accumulation | 0 | 0, 1 | Enable accumulation:<br>• 0 – no accumulation: `o_dout_ab = i_din_a × i_din_b`, `o_dout_cd = i_din_c × i_din_d`.<br>• 1 – accumulation: `o_dout_ab` and `o_dout_cd` are the accumulated values. The start of accumulation is signaled by asserting `i_load_ab = 1` or `i_load_cd = 1`, respectively.<br>When `i_load_ab` or `i_load_cd` is high, the previous value of the respective internal accumulation register is ignored, and the new value is stored. The output is set to this value.<br>When `i_load_ab` or `i_load_cd` is low, the old and new values are added, and the sum is stored. The output is `o_dout_ab = (i_din_a × i_din_b) + dout_ab_prev`, `o_dout_cd = (i_din_c × i_din_d) + dout_cd_prev`.<br>The `i_load_ab` and `i_load_cd` signal is internally pipelined to have the same latency as the input. If a set of inputs starts a new accumulation, then `i_load_ab` and `i_load_cd` must be high when those respective inputs are presented. If accumulation is not enabled, the `i_load_ab` and `i_load_cd` signals are ignored. |
| Enable Input Registers | 0 | 0, 1 | Enable input registers:<br>• 0 – no input registers.<br>• 1 – `i_din_a`, `i_din_b`, `i_din_c` and `i_din_d` are registered.<br>The input registers are controlled by the `i_in_reg_a_ce`, `i_in_reg_b_ce`, `i_in_reg_c_ce`, `i_in_reg_d_ce` and `i_in_reg_rstn` inputs. Enabling the input registers adds one cycle of latency. |
| Hide `ce` and `rstn` of Input registers | 0 | 0, 1 | If selected, the `i_in_reg_a_ce`, `i_in_reg_b_ce`, `i_in_reg_c_ce`, `i_in_reg_d_ce` and `i_in_reg_rstn` inputs are automatically tied high (1'b1). |

| Name | Default | Range | Description |
|---|---|---|---|
| Pipeline registers | 0 | 0–4 | The number of pipeline registers, not counting the input register. The total latency is `pipeline_regs + in_reg_enable`. |
| Hide `ce` and `rstn` of Pipeline registers | 0 | 0, 1 | If selected, the `i_pipeline_ce`, `i_pipeline_rstn` inputs are automatically tied high (1'b1). |

# Ports

***Table 28 • Floating Point Parallel Multiplier GUI Generated Pin Descriptions***

| Name | Direction | Description |
|---|---|---|
| `i_clk` | Input | Clock input, used for the (optional) registers and accumulator. |
| `i_din_a[(fp_size - 1):0]` | Input | 'A' data input to AB multiplier. |
| `i_din_b[(fp_size - 1):0]` | Input | 'B' data input to AB multiplier. |
| `i_din_c[(fp_size - 1):0]` | Input | 'C' data input to CD multiplier. |
| `i_din_d[(fp_size - 1):0]` | Input | 'D' data input to CD multiplier. |
| `i_in_reg_a_ce` | Input | If **Enable Input Registers** is enabled, `i_in_reg_a_ce` is the clock enable for `i_din_a`.<br>If **Enable Input Registers** is not enabled, `i_in_reg_a_ce` does not exist. |
| `i_in_reg_b_ce` | Input | If **Enable Input Registers** is enabled, `i_in_reg_b_ce` is the clock enable for `i_din_b`.<br>If **Enable Input Registers** is not enabled, `i_in_reg_b_ce` does not exist. |
| `i_in_reg_c_ce` | Input | If **Enable Input Registers** is enabled, `i_in_reg_c_ce` is the clock enable for `i_din_c`.<br>If **Enable Input Registers** is not enabled, `i_in_reg_c_ce` does not exist. |
| `i_in_reg_d_ce` | Input | If **Enable Input Registers** is enabled, `i_in_reg_d_ce` is the clock enable for `i_din_c`.<br>If **Enable Input Registers** is not enabled, `i_in_reg_d_ce` does not exist. |
| `i_in_reg_rstn` | Input | If **Enable Input Registers** is enabled, `i_in_reg_rstn` is the synchronous active-low reset for input registers.<br>If **Enable Input Registers** is not enabled, `i_in_reg_rstn` does not exist. |

| Name | Direction | Description |
|---|---|---|
| i_pipeline_ce | Input | If **Accumulation** is enabled, i_pipeline_ce is the clock enable for the accumulator register.<br>If **Pipeline Registers** are selected (greater than 0), i_pipeline_ce is the clock enable for pipeline registers.<br>If neither is enabled, i_pipeline_ce does not exist. |
| i_pipeline_rstn | Input | If **Accumulation** is enabled, i_pipeline_rstn is the synchronous, active-low reset for the accumulator register.<br>If **Pipeline Registers** are selected (greater than 0), i_pipeline_rstn is the synchronous, active-low reset for pipeline registers.<br>If neither is enabled, i_pipeline_rstn does not exist. |
| i_load_ab | Input | If **Accumulation** is enabled, i_load_ab resets the AB accumulator to i_din_a × i_din_b, ignoring the previous value.<br>If **Accumulation** is not enabled, i_load_ab does not exist.<br>This signal is internally pipelined to have the same latency as i_din_a × i_din_b. |
| i_load_cd | Input | If **Accumulation** is enabled, i_load_cd resets the CD accumulator to i_din_c × i_din_d, ignoring the previous value.<br>If **Accumulation** is not enabled, i_load_cd does not exist.<br>This signal is internally pipelined to have the same latency as i_din_c × i_din_d. |
| o_dout_ab[(fp_size - 1):0] | Output | Result of A × B multiplication and accumulation. |
| o_dout_cd[(fp_size - 1):0] | Output | Result of C × D multiplication and accumulation. |
| o_status_ab[1:0] | Output | Error status of o_dout_ab.<br>• 2'b00 – normal.<br>• 2'b01 – result is ± 0.0.<br>• 2'b10 – result is ± infinity.<br>• 2'b11 – last operation had underflow. The result is ± 0.0. |
| o_status_cd[1:0] | Output | Error status of o_dout_cd.<br>• 2'b00 – normal.<br>• 2'b01 – result is ± 0.0.<br>• 2'b10 – result is ± infinity.<br>• 2'b11 – last operation had underflow. The result is ± 0.0. |

## Files

The configurator produces a `.v` (verilog) or a `.vhd` (VHDL) wrapper file based on the chosen RTL model which must be instantiated in the user design. By default, the file is written to the current working directory of the project, but a different location can be chosen by clicking **Browse...** as shown in the following figure:

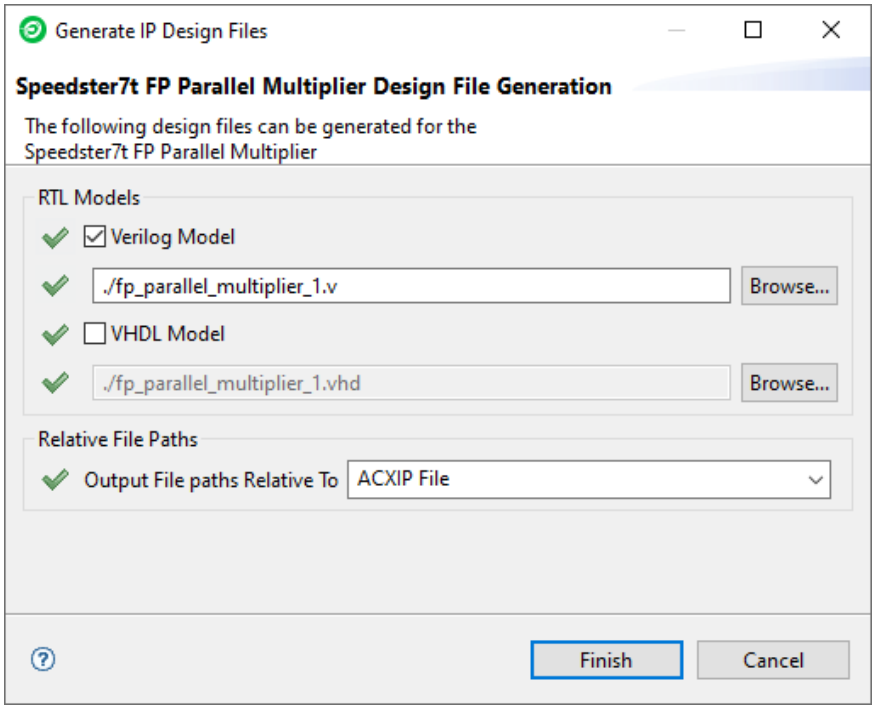


***Figure 39 • FP Parallel Multiplier Design File Generation***

# Examples

The following example shows the floating-point parallel multiplier configured for 16-bit floating point inputs with input registers, accumulation, three pipeline stages, and 16-bit output:

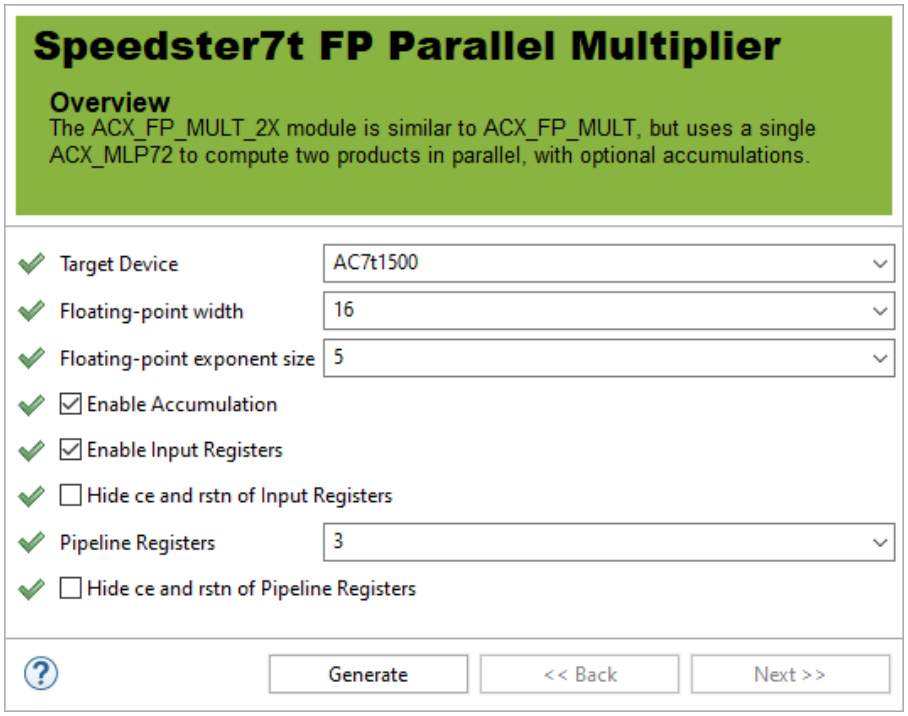


***Figure 40 • Three Pipeline Stage Floating Point Parallel Multiplier Configuration***

The following figure shows the IP diagram for this configuration:

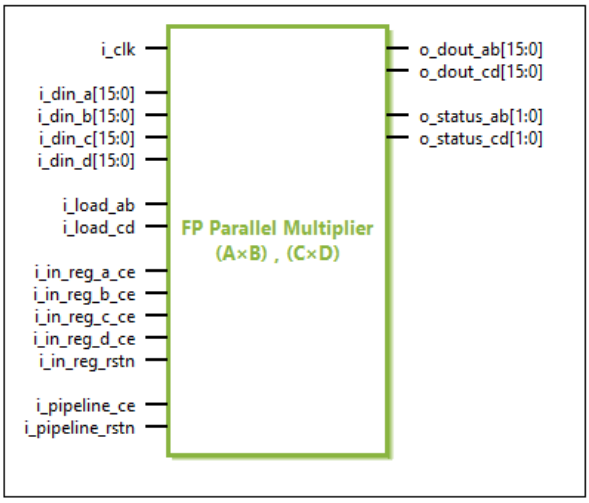


***Figure 41 • Three Pipeline Stage Floating Point Parallel Multiplier IP Diagram***

# Chapter 14 : Floating-Point Sum of Products

## Description

The Floating Point Sum of Products soft IP core configures two floating-point multipliers and an adder to provide the sum of two products ((A × B) ± (C × D)) along with optional result accumulation. The floating-point sum of products instantiates the ACX_FP_MULT_ADD module shown in the ***Speedster7t Component Library User Guide*** **(UG086)**[7].

## Configuration

The Floating Point Sum of Products soft IP configurator has the following options:

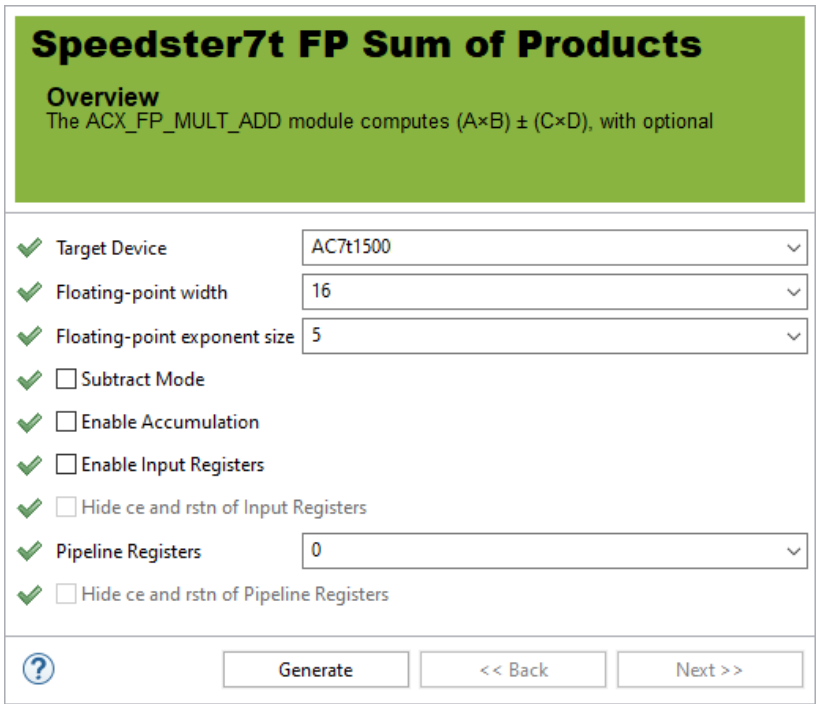


***Figure 42 • Floating Point Sum of Products Soft IP Configurator***

7 https://www.achronix.com/documentation/speedster7t-component-library-user-guide-ug086

***Table 29 • Configuration Options***

| Name | Default | Range | Description |
|---|---|---|---|
| Target Device | AC7t1500 | All Speedster7t devices | Set to match the target device of the project. |
| Floating Point width | 16 | 16, 24 | Width of floating-point number. Supports fp24, fp16, and fp16e8. |
| Floating point exponent size | 5 | 5, 8 | Size of floating-point exponent. For floating point width of 16, the exponent size can be either 5(fp16) or 8(fp16e8). For floating point width of 24, the exponent size must be 8. |
| Subtract Mode | 0 | 0, 1 | Enable subtract mode:<br>• 0 – compute `(i_din_a × i_din_b) + (i_din_c × i_din_d)`.<br>• 1 – compute `(i_din_a × i_din_b) – (i_din_c × i_din_d)`. |
| Enable Accumulation | 0 | 0, 1 | Enable accumulation:<br>• 0 – no accumulation: `o_dout = (i_din_a × i_din_b) ± (i_din_c × i_din_d)` (determined by subtract parameter).<br>• 1 – accumulation: `o_dout` is the accumulated value. The start of accumulation is signaled by asserting `i_load = 1`.<br>When `i_load` is high, the previous value of the internal accumulation register is ignored, and the new value is stored. The output is set to this value. When `i_load` is low, the old and new values are added, and the sum is stored. The output is `o_dout = (i_din_a × i_din_b) ± (i_din_c × i_din_d) + dout_prev`.<br>The `i_load` signal is internally pipelined to have the same latency as the input. If a set of inputs starts a new accumulation, `i_load` must be high when those inputs are presented. If accumulation is not enabled, the `i_load` signal is ignored. |
| Enable Input Registers | 0 | 0, 1 | Enable input registers:<br>• 0 – no input registers.<br>• 1 – `i_din_a`, `i_din_b`, `i_din_c` and `i_din_d` are registered.<br>The input registers are controlled by the `i_in_reg_ac_ce`, `i_in_reg_bd_ce` and `i_in_reg_rstn` inputs. Enabling the input registers adds one cycle of latency. |
| Hide `ce` and `rstn` of Input registers | 0 | 0, 1 | If selected, the `i_in_reg_a_ce`, `i_in_reg_b_ce`, and `i_in_reg_rstn` inputs are automatically tied high (1'b1). |

| Name | Default | Range | Description |
|---|---|---|---|
| Pipeline registers | 0 | 0–5 | The number of pipeline registers, not counting the input register. The total latency is `pipeline_regs + in_reg_enable`. |
| Hide `ce` and `rstn` of Pipeline registers | 0 | 0, 1 | If selected, the `i_pipeline_ce` and `i_pipeline_rstn` inputs are automatically tied high (1'b1). |

# Ports

***Table 30 • Floating Point Sum of Products GUI-Generated Pin Descriptions***

| Name | Direction | Description |
|---|---|---|
| `i_clk` | Input | Clock input, used for the (optional) registers and accumulator. |
| `i_din_a[(fp_size - 1):0]` | Input | 'A' data input to AB multiplier. |
| `i_din_b[(fp_size - 1):0]` | Input | 'B' data input to AB multiplier. |
| `i_din_c[(fp_size - 1):0]` | Input | 'C' data input to CD multiplier. |
| `i_din_d[(fp_size - 1):0]` | Input | 'D' data input to CD multiplier. |
| `i_in_reg_ac_ce` | Input | If **Enable Input Registers** is enabled, `i_in_reg_ac_ce` is the clock enable for `i_din_a` and `i_din_c`.<br>If **Enable Input Registers** is not enabled, `i_in_reg_ac_ce` does not exist. |
| `i_in_reg_bd_ce` | Input | If **Enable Input Registers** is enabled, `i_in_reg_bd_ce` is the clock enable for `i_din_b and i_din_d`.<br>If **Enable Input Registers** is not enabled, `i_in_reg_bd_ce` does not exist. |
| `i_in_reg_rstn` | Input | If **Enable Input Registers** is enabled, `i_in_reg_rstn` is the synchronous active-low reset for input registers.<br>If **Enable Input Registers** is not enabled, `i_in_reg_rstn` does not exist. |
| `i_pipeline_ce` | Input | If **Accumulation** is enabled, `i_pipeline_ce` is the clock enable for the accumulator register.<br>If **Pipeline Registers** are selected (greater than 0), `i_pipeline_ce` is the clock enable for pipeline registers.<br>If neither is enabled, `i_pipeline_ce` does not exist. |

| Name | Direction | Description |
|---|---|---|
| i_pipeline_rstn | Input | If **Accumulation** is enabled, i_pipeline_rstn is the synchronous, active-low reset for the accumulator register.<br>If **Pipeline Registers** are selected (greater than 0), i_pipeline_rstn is the synchronous, active-low reset for pipeline registers.<br>If neither is enabled, i_pipeline_rstn does not exist. |
| i_load | Input | If **Accumulation** is enabled, i_load resets the accumulator to: (i_din_a × i_din_b) ± (i_din_c × i_din_d), ignoring the previous value. This signal is internally pipelined to have the same latency as: (i_din_a × i_din_b) ± (i_din_c × i_din_d).<br>If **Accumulation** is not enabled, i_load does not exist. |
| o_dout[(fp_size - 1):0] | Output | Result of multiplication and accumulation. |
| o_status[1:0] | Output | Error status of o_dout.<br>• 2'b00 – normal.<br>• 2'b01 – result is ± 0.0.<br>• 2'b10 – result is ± infinity<br>• 2'b11 – last operation had underflow. The result is ± 0.0. |

# Files

The configurator produces a .v (verilog) or .vhd (VHDL) wrapper file based on the chosen RTL model which must be instantiated in the user design. By default, the file is written to the current working directory of the project, but a different location can be chosen by clicking **Browse...** as shown in the following figure:

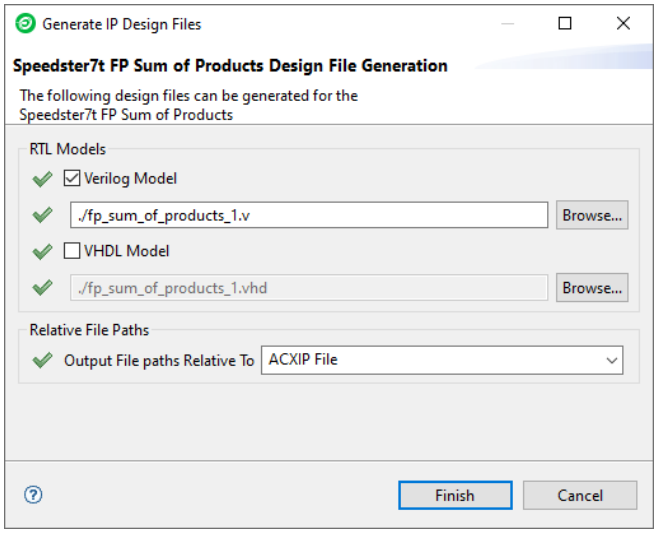


***Figure 43 • FP Sum of Products Design File Generation***

# Examples

The following example shows the floating point sum of products configured for 16-bit floating point inputs with input registers, accumulation, five pipeline stages, and 16-bit output:

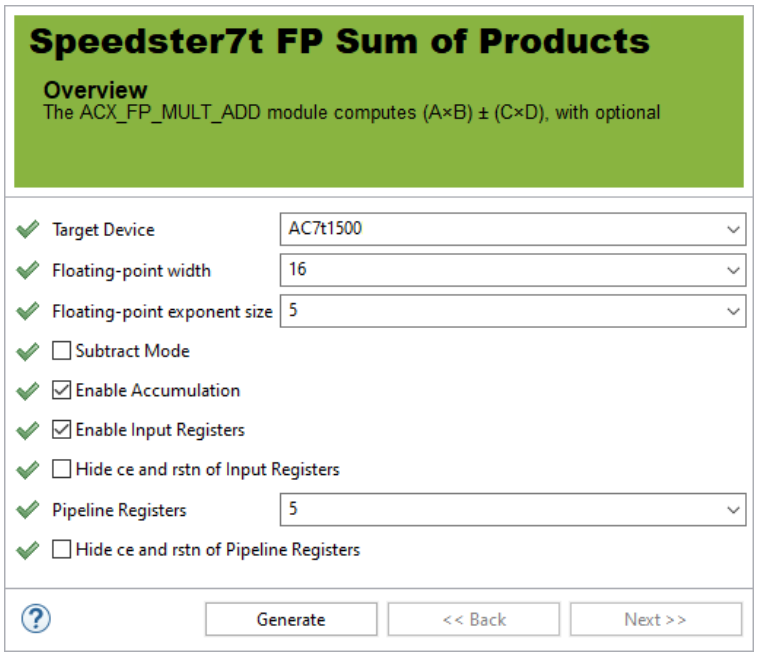


***Figure 44 • Five Pipeline Stage Floating Point Sum of Products Configuration***

The following figure shows the IP diagram for this configuration:

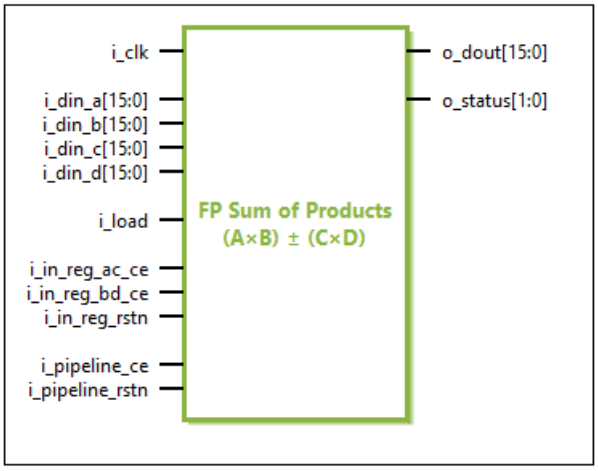


***Figure 45 • Five Pipeline Stage Floating Point Sum of Products IP Diagram***

# Chapter 15 : Speedster7t Shift Register

## Description

The Shift Register macro implements a shift register using fabric flip-flop primitives. The shift register can be configured to support varying widths and depths of shift functions. The generated shift register includes a clock enable and reset.

## Configuration

The shift register macro has the following options:

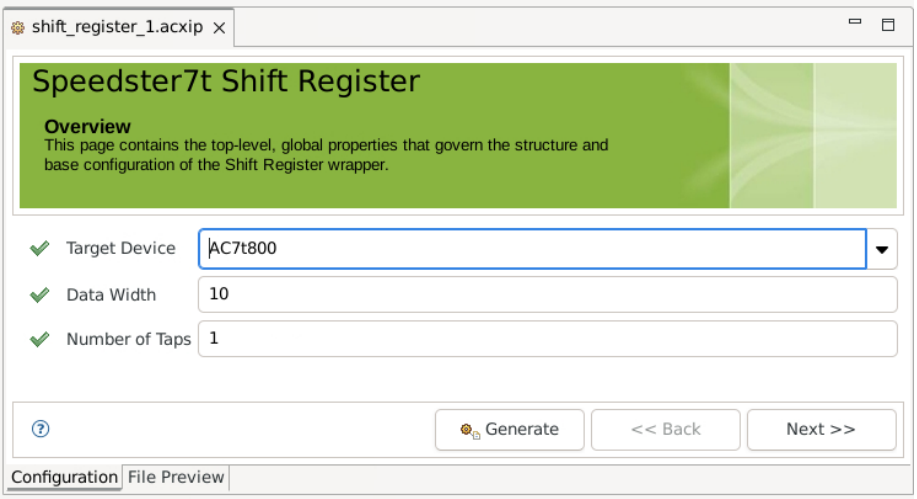


***Figure 46 • Shift Register Configurator***

***Table 31 • Configuration Options***

| Name | Default | Range | Description |
|---|---|---|---|
| Target Device | Selected target device | – | Set to match the target device of the project. |
| Data Width | 10 | 1 to 65,536 | Sets the width of the input and output data bus. |
| Number of Taps | 1 | 1 to 256 | Determines the number of taps in the shift register. All intermediate taps are output from the soft IP instance. |

The number of flip-flops used is calculated by:

*Data Width* × *Number of Taps*

For very wide or deep shift registers, a large number of flip-flops may be used which might result in sub-optimal timing closure. In these circumstances, it is recommended to use a BRAM or LRAM to implement the shift register function.

> **Note**
>
> A shift register implemented with a BRAM or LRAM only provides access to the end tap of the shift register and not the intermediate stages.

***Table 32 • Ports***

| Name | Direction | Description |
|---|---|---|
| `clk` | Input | Clock input to each flip-flop. |
| `din[(Data Width - 1):0]` | Input | Input data bus. |
| `en` | Input | Clock enable:<br>• `en` = `1'b0` – shift register is stopped. No data is input. Current output is maintained.<br>• `en` = `1'b1` – shift register transfers data from tap to tap on each rising edge of `clk`. |
| `[(Number of Taps - 1):0] dout[(Data Width - 1):0]` | Output | Output data bus and intermediate taps. The final tap of the shift register (the input delayed by Number of Taps clock cycles) is output on [(Number of Taps – 1):0] `dout`. |

# Examples

The following example shows the shift register configured for 72 bits wide, by five stages deep:

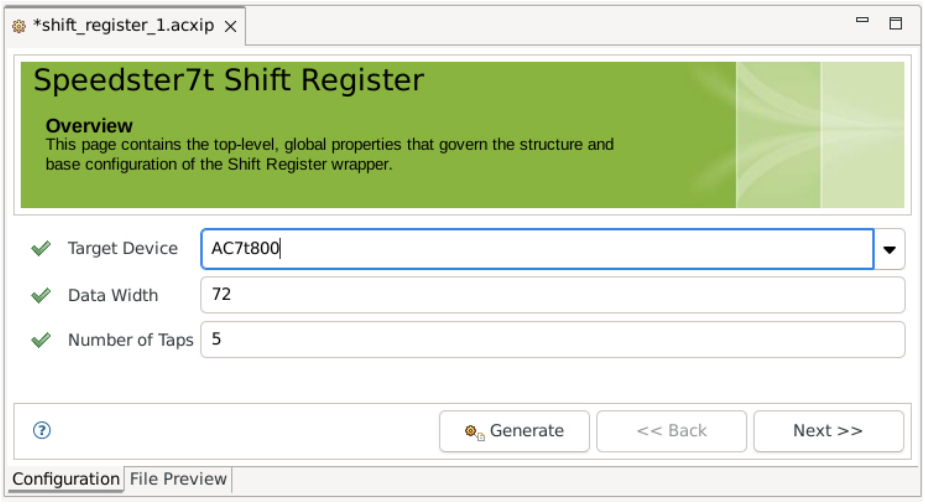


***Figure 47 • 72-Bit, 5-Stage Shift Register Configuration***

The following figure shows the IP diagram for this configuration:

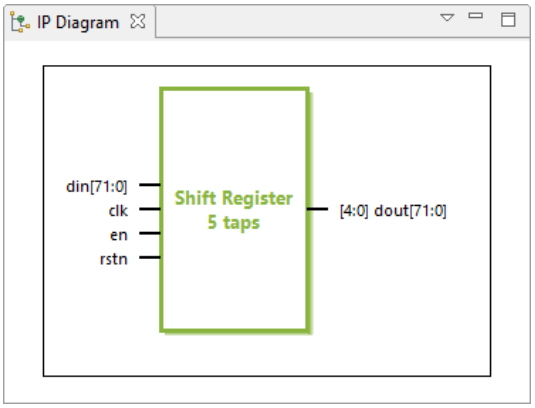


***Figure 48 • 72-bit, 5-Sstage Shift Register IP Diagram***

# Chapter 16 : Speedster7t AXI Initiator NAP

## Description

The AXI initiator NAP soft IP core implements a 2D NoC access point (NAP) that connects to user logic in the fabric, which responds to read and write AXI transactions from the 2D NoC. The soft IP configurator allows choosing the NAP location and setting the south-to-north arbitration weight for the NAP. The AXI initiator NAP is comprised of the ACX_NAP_AXI_MASTER primitive. More details on the primitive can be found in the ACX_NAP_AXI_MASTER page of the ***Speedster7t Component Library User Guide*** (UG086)[8].

## Configuration

The AXI Initiator NAP soft IP configurator has the following options:

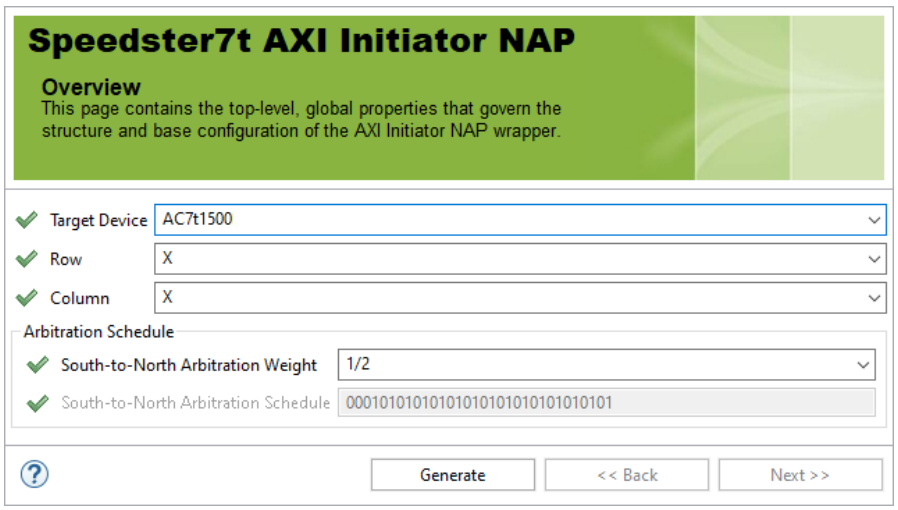


***Figure 49 • AXI Initiator NAP***

8 https://www.achronix.com/documentation/speedster7t-component-library-user-guide-ug086

**Table 33 • Configuration Options**

| Name | Default | Range | Description |
|---|---|---|---|
| Target Device | AC7t1500 | All Speedster7t devices | Set to match the target device of the project. |
| Row | X | 1 to 8 | Set to match the row number location of the NAP. |
| Column | X | 1 to 10 | Set to match the column number location of the NAP. |
| South-to-north arbitration weight | 1/2 | 0, 1/2, 1/3, ... , 1/10, 1, custom | Sets the fraction of remaining bandwidth this NAP is guaranteed in the south-to-north direction. "Remaining bandwidth" is the bandwidth not guaranteed to the downstream NAPs on the same column. |
| South-to-north custom arbitration schedule | 00010101010101010101010101010101 | 00000000000000000000000000000000 to 11111111111111111111111111111111 | A 32-bit number that allows the specific setting of a custom arbitration schedule in the south-to-north direction, where "1" means this NAP gets priority and "0" means the downstream NAPs get priority. Only available when "CUSTOM" is chosen for the arbitration weight. |

# Files

The configuration settings shown in the previous table prompt the generation of an AXI initiator NAP design file in the selected location as shown in the following example:

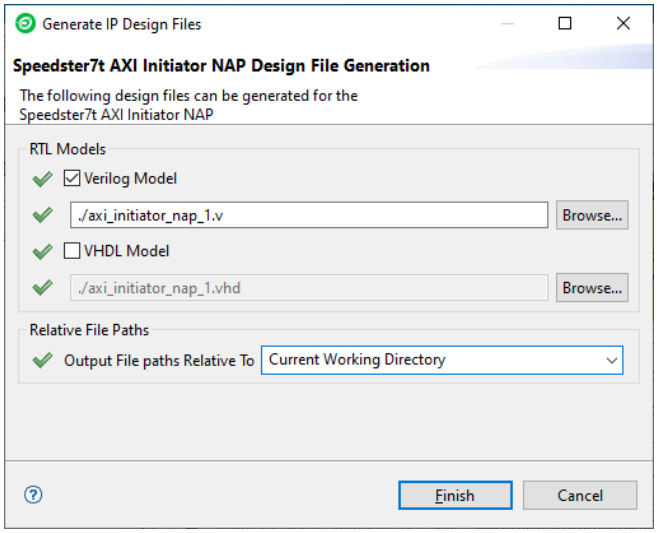


**Figure 50 • AXI Initiator NAP File Generation**

# Examples

The following is an example configuration of the AXI initiator NAP located in row 6, column 3, using an arbitration weight of 1/2:

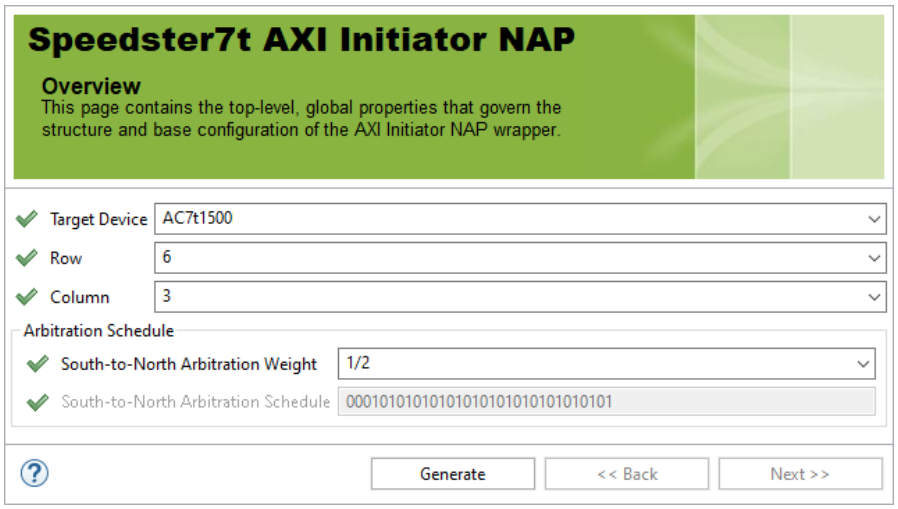


***Figure 51 • AXI Initiator NAP in Row 6, Column 3***

The following figure shows the IP diagram for this configuration:

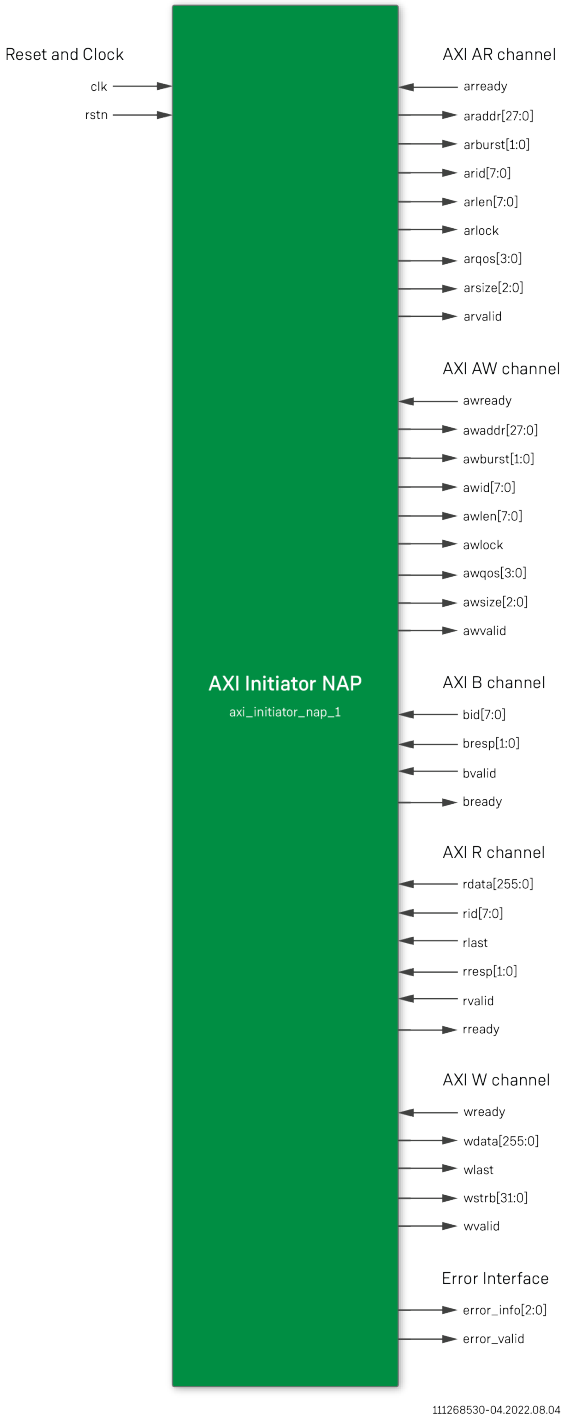


***Figure 52 • AXI Initiator NAP IP Diagram***

# Chapter 17 : Speedster7t AXI Responder NAP

## Description

The AXI responder NAP soft IP core implements a 2D NoC access point (NAP) that connects to user logic in the fabric, which initiates read and write AXI transactions to the 2D NoC. The AXI responder NAP allows choosing the NAP location and setting the east-to-west and west-to-east arbitration weights for the NAP. This soft IP module is comprised of the ACX_NAP_AXI_SLAVE primitive. More details on the primitive can be found in the ACX_NAP_AXI_SLAVE page of the ***Speedster7t Component Library User Guide*** (UG086)[9].

## Configuration

The AXI responder NAP soft IP configuration GUI has the following options:

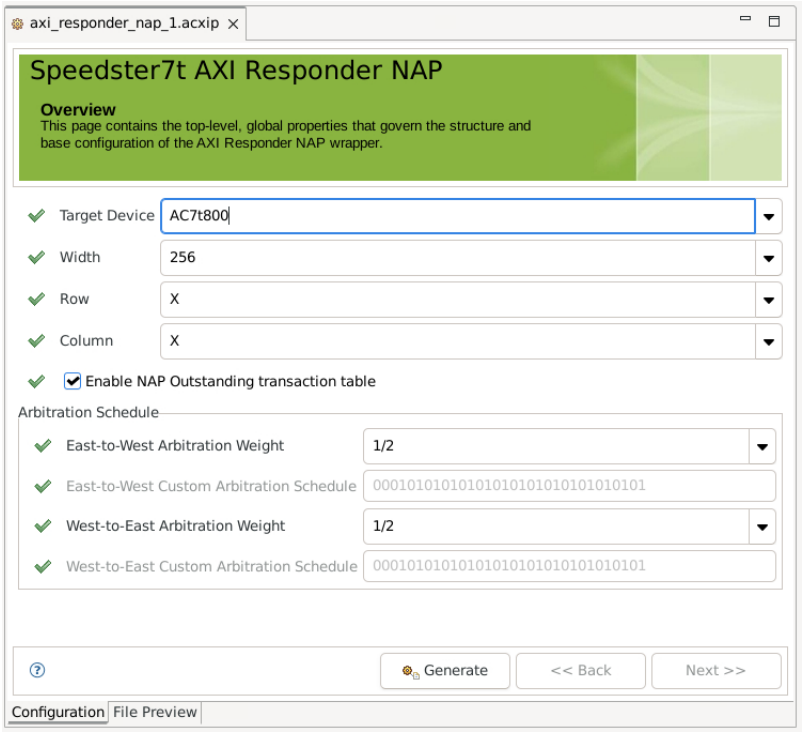


***Figure 53 • AXI Responder NAP Configuration***

9 https://www.achronix.com/documentation/speedster7t-component-library-user-guide-ug086

***Table 34 • Configuration Options***

| Name | Default | Range | Description |
|---|---|---|---|
| Target Device | AC7t1500 | All Speedster7t devices | Set to match the target device of the project. |
| Width | 256 | 32, 64, 128,256 | Data width. If 256 is not chosen, a width adapter will be added. |
| Row | X | 1 to 8 | Set to match the row number location of the NAP. |
| Column | X | 1 to 10 | Set to match the column number location of the NAP. |
| Enable NAP outstanding transaction table | Enabled | Enabled/disabled | Enable or disable the NAP outstanding transaction table. Can be disabled to allow for more outstanding transactions in flight. Only disable if the NAP sends AXI transactions only east or only west, or if eastbound and westbound transactions always have different transaction IDs. |
| East-to-west arbitration weight | 1/2 | 0, 1/2, 1/3, ... , 1/10, 1, custom | Sets the fraction of remaining bandwidth this NAP is guaranteed in the east-to-west direction. "Remaining bandwidth" is the bandwidth not guaranteed to the downstream NAPs on the same row. |
| East-to-west custom arbitration schedule | 00010101010101010101010101010101 | 00000000000000000000000000000000 to 11111111111111111111111111111111 | A 32-bit number that allows the specific setting of a custom arbitration schedule in the east-to-west direction, where "1" means this NAP gets priority and "0" means the downstream NAPs get priority. Only available when **Custom** is chosen for the arbitration weight. |
| West-to-east arbitration weight | 1/2 | 0, 1/2, 1/3, ... , 1/10, 1, custom | Sets the fraction of remaining bandwidth this NAP is guaranteed in the west-to-east direction. "Remaining bandwidth" is the bandwidth not guaranteed to the downstream NAPs on the same row. |
| West-to-east custom arbitration schedule | 00010101010101010101010101010101 | 00000000000000000000000000000000 to 11111111111111111111111111111111 | A 32-bit number that allows the specific setting of a custom arbitration schedule in the west-to-east direction, where "1" means this NAP gets priority and "0" means the downstream NAPs get priority. Only available when **Custom** is chosen for the arbitration weight. |

# Files

The configuration settings shown in the previous table prompt the generation of an AXI responder NAP design file in the selected location as shown in the following example:

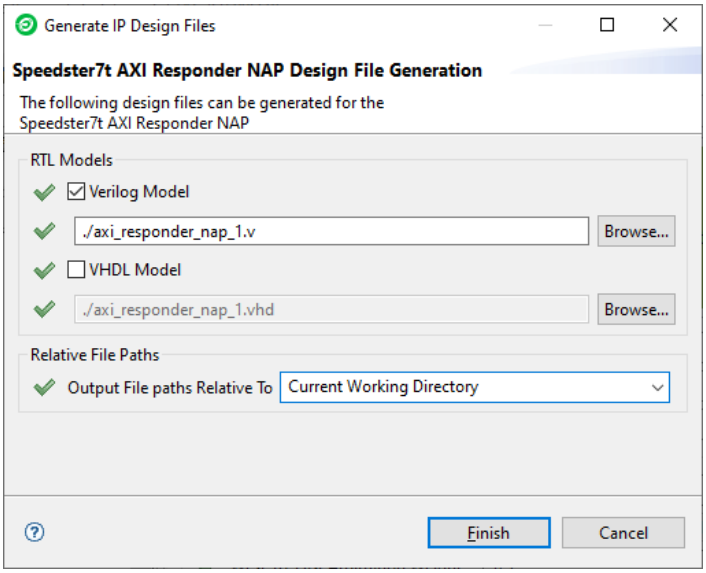


***Figure 54 • AXI Responder NAP File Generation***

# Examples

An sample configuration of the AXI responder NAP located in row 8, column 2, using a 1/3 east-to-west and 1/4 west-to-east arbitration weight is shown in the following example:

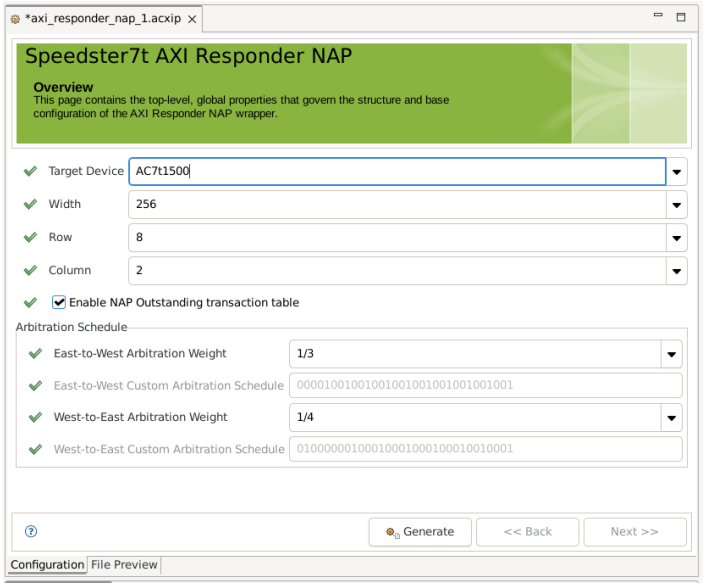


***Figure 55 • AXI Responder NAP Configuration***

The following figure shows the IP diagram for this configuration:

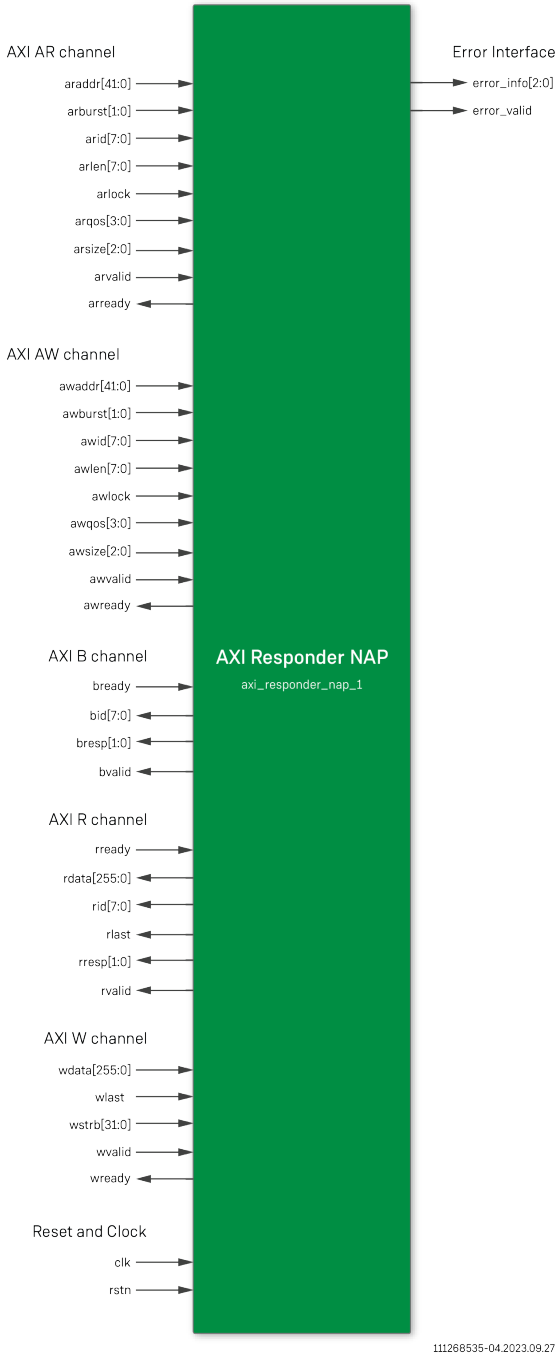


***Figure 56 • AXI Responder NAP IP Diagram***

# Chapter 18 : Speedster7t AXI Width Adapter

## Description

The AXI Width Adapter soft IP core implements an AXI wrapper that can be used with a specified address width, ID width, and data width, other than the defaults used in an AXI initiator NAP or AXI responder NAP. This core can be used to connect from a NAP to the user design, which may have different sized address and data buses while also handling conversion of AXI burst transactions if so configured.

## Configuration

The AXI Width Adapter soft IP configuration GUI has the following options:

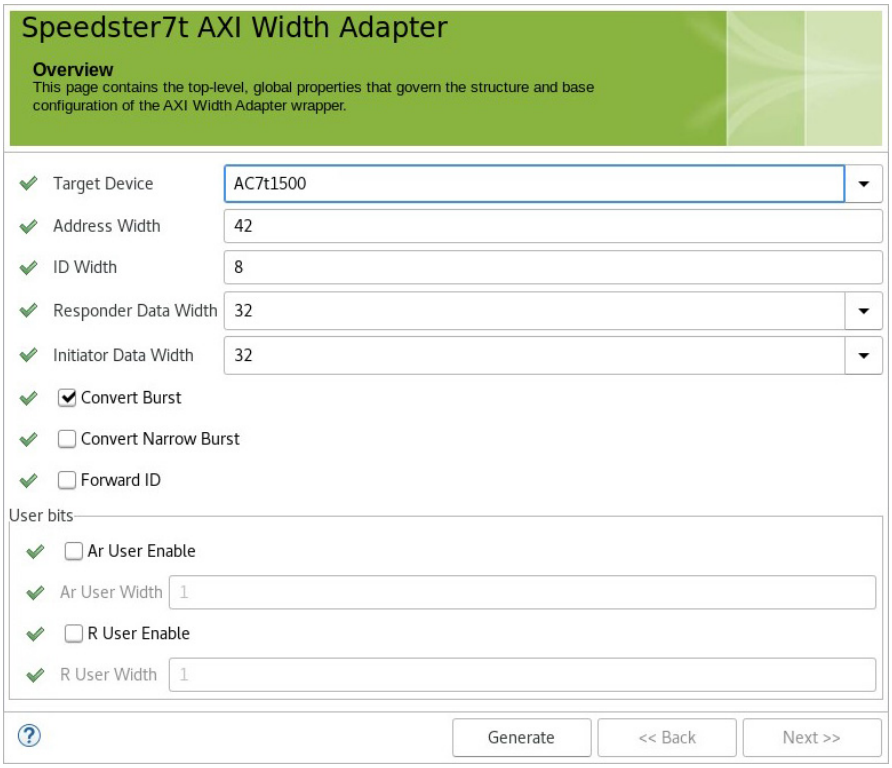


***Figure 57 • AXI Width Adapter Configuration***

***Table 35 • Configuration Options***

| Name | Default | Range | Description |
|---|---|---|---|
| Target Device | AC7t1500 | All Speedster7t devices | Set to match the target device of the project. |
| Address Width | 42 | 8 to 42 | Specifies the width of the ar/aw address bus. |
| ID Width | 8 | 1 to 12 | Specifies the width of the `arid`, `awid`, `rid`, and `bid` signals. |
| Responder Data Width | 32 | 32, 64, 128, 256, 512 | Specifies the width of `rdata` and `wdata` on the responder interface. |
| Initiator Data Width | 32 | 32, 64, 128, 256, 512 | Specifies the width of `rdata` and `wdata` on the initiator interface. |
| Convert Burst | enabled | enabled/disabled | When adapting to a wider bus, re-pack the full-width burst instead of passing the narrow burst. |
| Convert Narrow Burst | disabled | enabled/disabled | When adapting to a wider bus, re-pack all bursts instead of passing the narrow burst. |
| Forward ID | disabled | enabled/disabled | Forward the ID through the adapter. |
| Ar User Enable | disabled | enabled/disabled | Propagate the `aruser` signal. |
| Ar User Width | 1 | 1 to 8 | When enabled, specifies the width of the `aruser` signal. |
| R User Enable | disabled | enabled/disabled | When enabled, propagates the `ruser` signal. |
| R User Width | 1 | 1 to 8 | When enabled, specifies the width of the `ruser` signal. |

# Files

The configuration settings in the previous table prompt the generation of a Verilog or VHDL AXI width adapter design file in the selected location as shown in the following example:

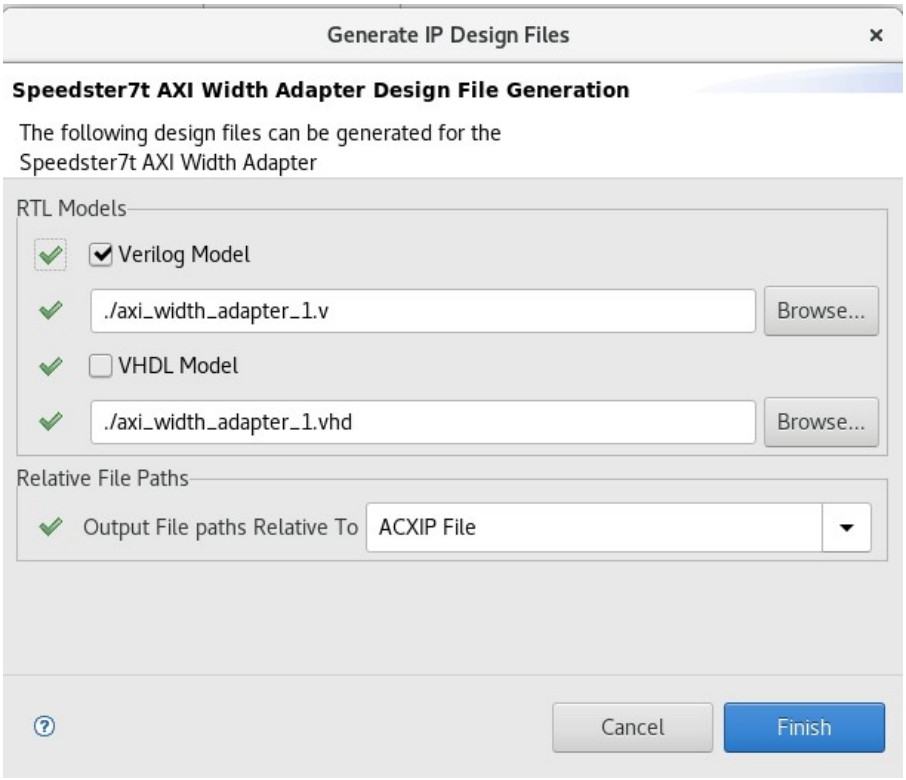


***Figure 58 • AXI Width Adapter File Generation***

# Examples

The following example shows the configuration of an AXI width adapter with a 32-bit address, 8-bit ID, 64-bit responder data width, and 512-bit initiator data width. Convert burst, convert narrow burst, and forward ID are all enabled, along with user buses, Ar user enable, and R user enable, both set to 1 bit.

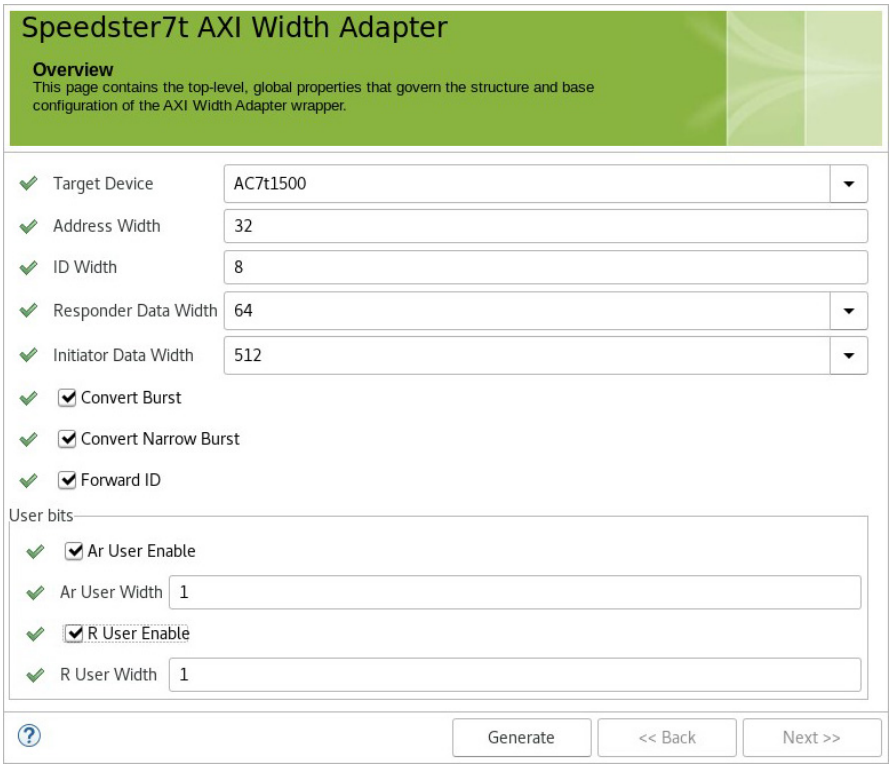


***Figure 59 • AXI Width Adapter Example Configuration***

The following figure shows the IP diagram for this configuration:

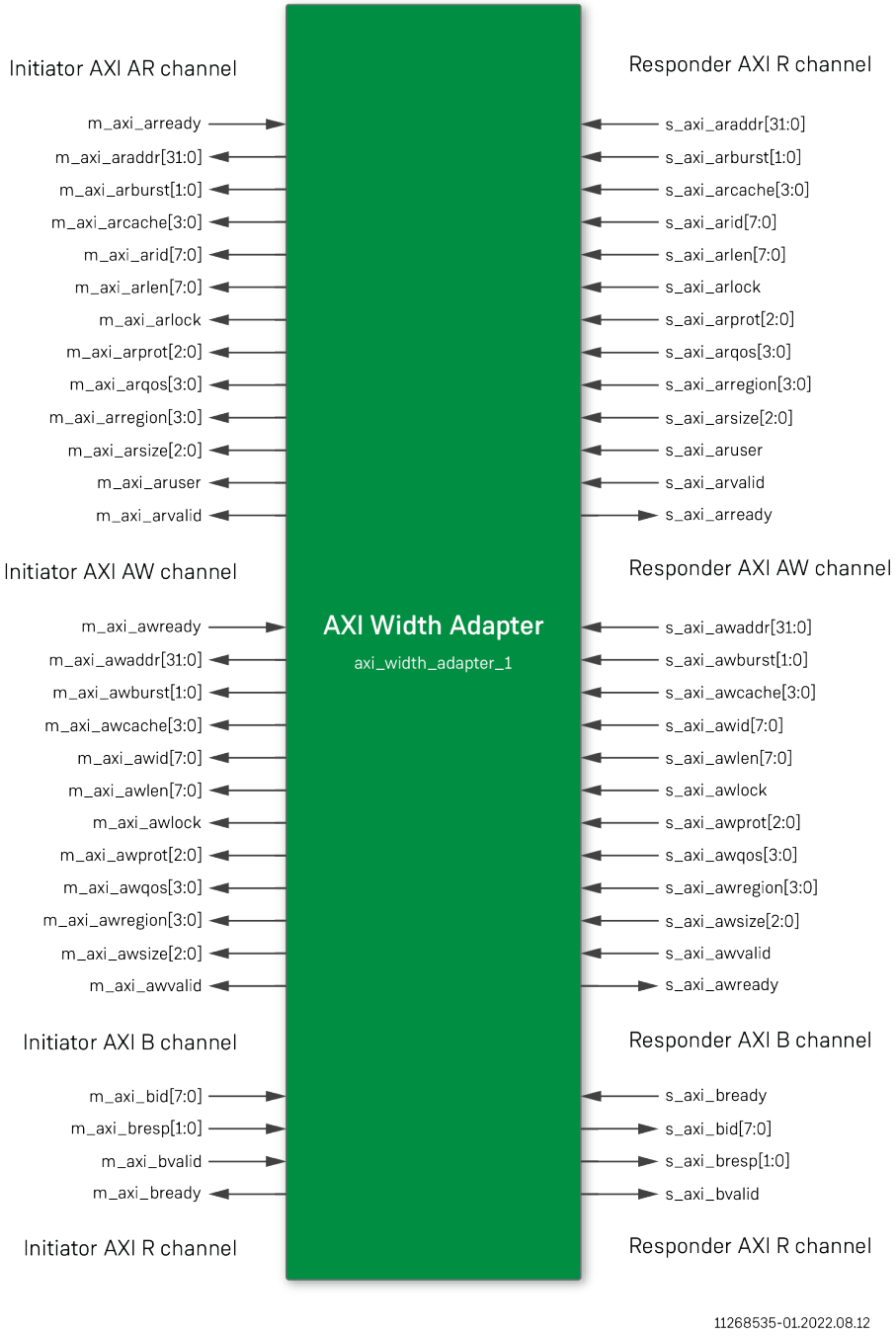


***Figure 60 • AXI Width Adapter Example IP Diagram***

# Chapter 19 : Revision History

| Version | Date | Description |
|---|---|---|
| 1.0 | 13 Sep 2021 | • Initial release. |
| 2.0 | 20 Sep 2022 | • Add device manager. |
| 2.1 | 25 Aug 2023 | • Added the following soft IP:<br>◦ **Speedster7t Floating Point Adder** (page 45)<br>◦ **Speedster7t Floating Point Multiplier** (page 50)<br>◦ **Speedster7t Floating Point Multiplier Plus** (page 55)<br>◦ **Speedster7t Floating Point Parallel Multiplier** (page 60)<br>◦ **Speedster7t Floating Point Sum of Products** (page 66)<br>◦ **Speedster7t AXI Initiator NAP** (page 74)<br>◦ **Speedster7t AXI Responder NAP** (page 78)<br>◦ Speedster7t Cryptographic Engine<br>• Updated Speedster7t Device Manager User Guide. |
| 2.2 | 05 Nov 2023 | • Remove references to end-of-life devices. |
| 2.3 | 29 Dec 2025 | • Updated **Speedster7t BRAM72K Soft IP** (page 5), **Speedster7t AXI Responder NAP** (page 78) with new options.<br>• Removed Speedster7t ROM Soft IP.<br>• Added **Speedster7t AXI Width Adapter** (page 82).<br>• Moved Device Manager to its own user guide, *Speedster7t Device Manager User Guide* (UG115). |