Speedster7t Soft IP User Guide (UG103)

Speedster FPGAs

Preliminary Data

Achronix®

Data Acceleration

Copyrights, Trademarks and Disclaimers

Preliminary Data

This document contains preliminary information and is subject to change without notice. Information provided herein is based on internal engineering specifications and/or initial characterization data.

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: Introduction

There are a number of available soft IP cores for the Speedster®7t family of devices. Each of these cores has an IP configurator within ACE that allows configuration of the soft IP core to the user's specifications. When configured, the generated wrapper for the core can be instantiated within a user project enabling both synthesis and simulation of the design.

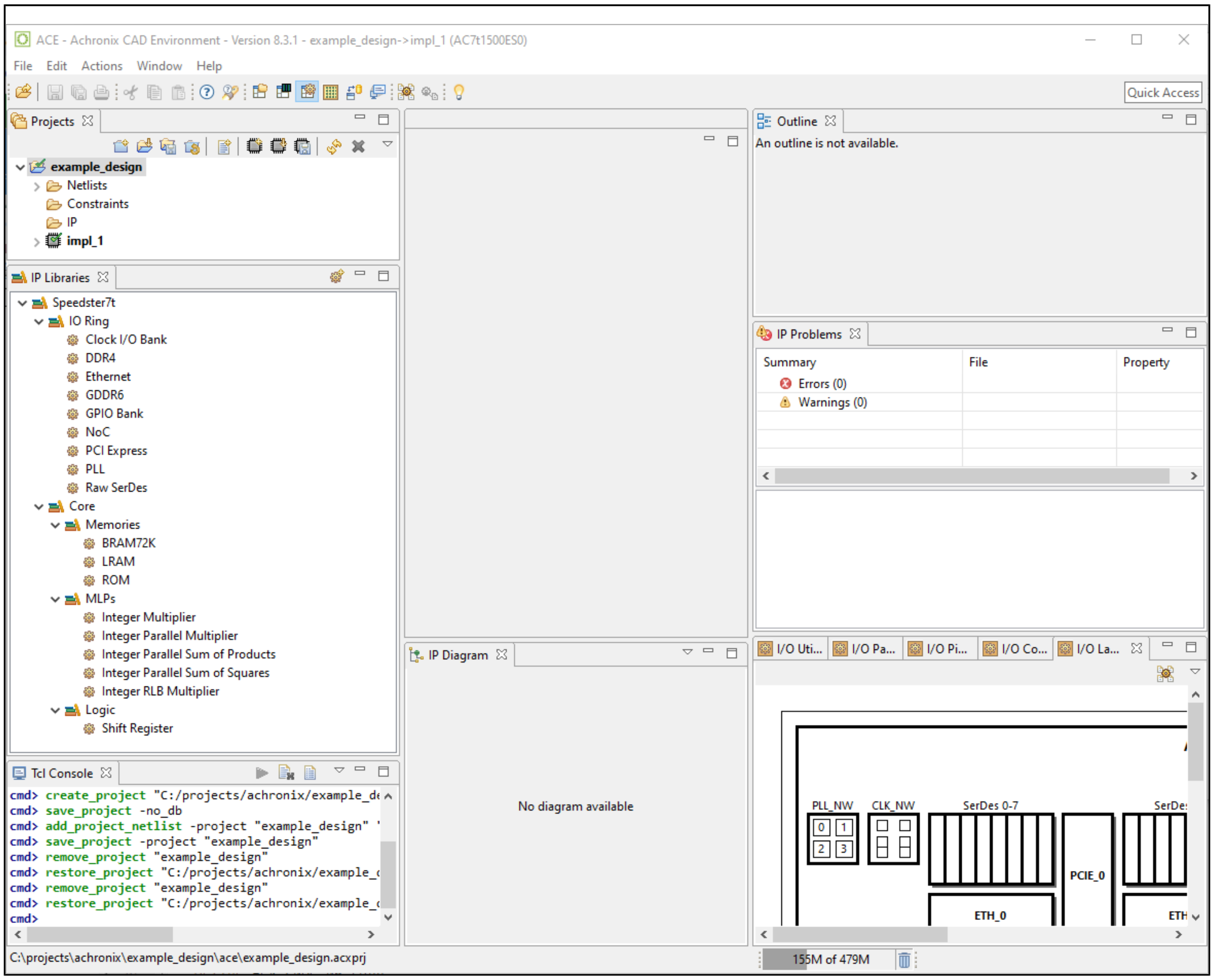
This document describes the available soft IP cores and the methods for configuration and instantiation of each. Soft IP cores are primarily implemented using the components present in the FPGA programmable fabric. For details of these components, see the *Speedster7t IP Component Library User Guide*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

Chapter - 2: Instructions

Within ACE, all IP cores are accessed using the IP perspective (see the "Perspectives" chapter in the *ACE User Guide* (UG070). 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 below, use the instructions in the ACE User Guide.

IO Ring and Core

Within the Speedster7t device there are two categories of IP core. These are listed within the IP Libraries pane as **IO Ring** and **Core**. For the Speedster7t family of devices, the following soft IP cores are available:



An outline is not available.

No diagram available

```
cmd> create_project "C:/projects/achronix/example_de
cmd> save_project -no_db
cmd> add_project_netlist -project "example_design" '
cmd> save_project -project "example_design"
cmd> remove_project "example_design"
cmd> restore_project "C:/projects/achronix/example_
cmd> remove_project "example_design"
cmd> restore_project "C:/projects/achronix/example_
cmd>
```

C:\projects\achronix\example_design\ace\example_design.acxprj

Figure 1: *Speedster7t IP Libraries View*

IO Ring

The IO Ring contains the configuration for each of the IP cores located in the IO ring of the device such as the Clock Banks, PLLs, GDDR, DDR, GPIO, PCIe, 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 device used in the project. For the Speedster7t devices, the soft cores shown above (see page 7), and listed below (see page 9) are available. The details of how to configure each of these cores are given on the appropriate page.

Chapter - 3: Available Configurators

Memories

- BRAM72K Soft IP (see page 10) - For creating large block RAM memory arrays
- LRAM Soft IP (see page 17) - For creating large logic RAM memory arrays
- ROM Soft IP (see page 24) - For creating ROMs constructed of either block RAM or logic RAM

MLPs

- Integer Multiplier Soft IP (see page 31) - For creating a single multiplier of up to 32 × 32
- Integer Parallel Multiplier Soft IP (see page 37) - For creating parallel multipliers of up to 32 × 32
- Integer Parallel Sum of Products Soft IP (see page 44) - For integer sum of products from up to 24 multipliers
- Integer Parallel Sum of Squares Soft IP (see page 50) - For integer sum of squared inputs
- Integer RLB Multiplier Soft IP (see page 56) - For creating a single multiplier using RLBs in the fabric logic

Logic

- Speedster7t Shift Register Soft IP (see page 62) - For creating DFF-based shift registers

Chapter - 4: BRAM72K Soft IP

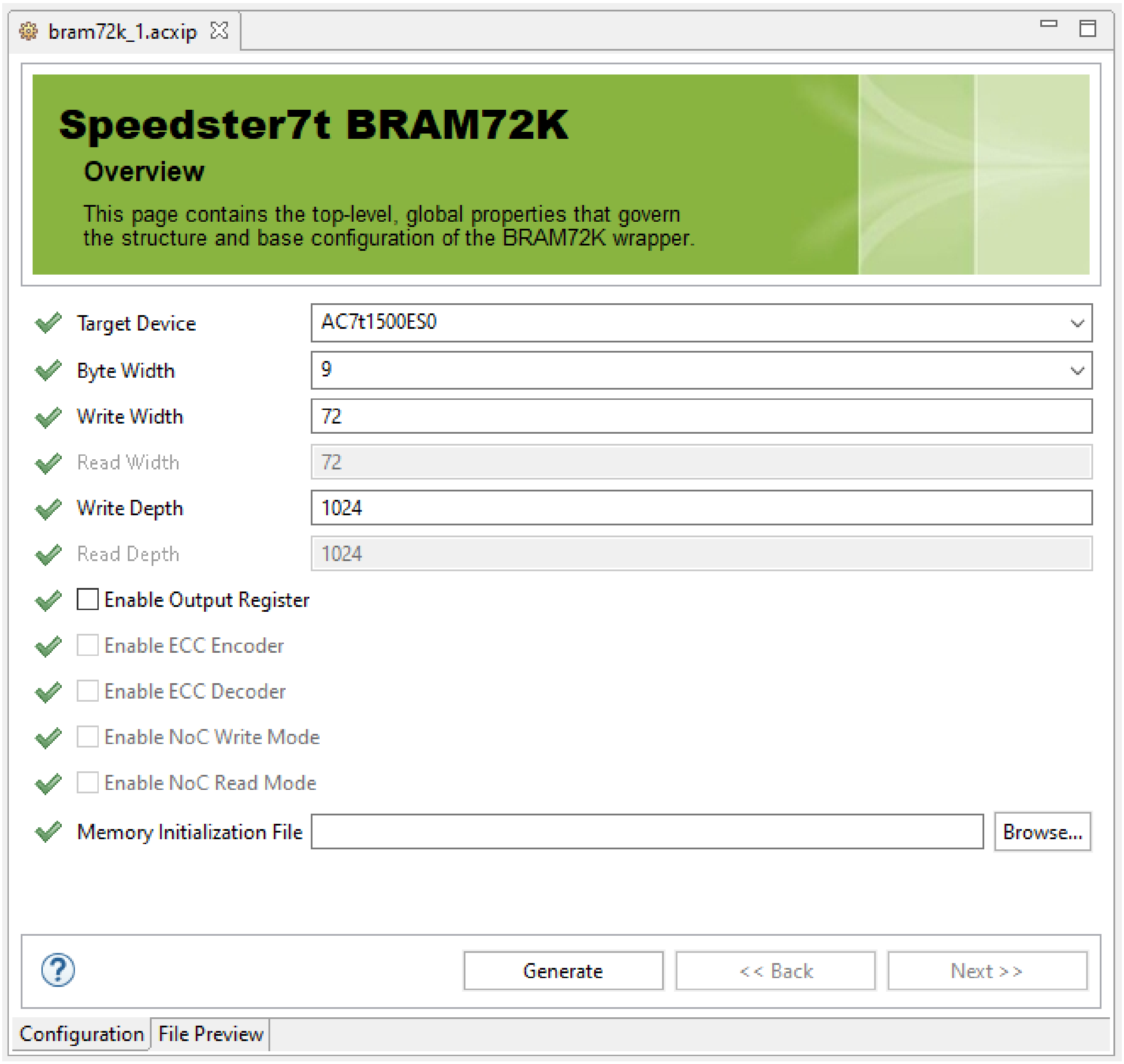
Description

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:



Speedster7t BRAM72K

Overview

This page contains the top-level, global properties that govern the structure and base configuration of the BRAM72K wrapper.

Target Device	AC7t1500ES0
Byte Width	9
Write Width	72
Read Width	72
Write Depth	1024
Read Depth	1024
☐ Enable Output Register	
☐ Enable ECC Encoder	
☐ Enable ECC Decoder	
☐ Enable NoC Write Mode	
☐ Enable NoC Read Mode	
Memory Initialization File	

Figure 2: ***BRAM72K Soft IP Configuration***

Table 1: ***Configuration Options***

Name	Default	Range	Description
Target Device	AC7t1500ES0	All Speedster7t devices	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. Currently set to match the write width. This value cannot be changed by the user. Future releases of this user macro will allow the user to configure different write and read widths.
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 dependant on the write width as detailed in Write and Read Depths versus Data Width (see page 14)
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 will allow the user to configure 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.
Enable NoC Write Mode	Off	On, Off	Determines whether the BRAM can be written directly from the NoC. This option is currently disabled and cannot be set by the user.
Enable NoC Read Mode	Off	On, Off	Determines whether the BRAM can be read directly from the NoC. This option is currently disabled and cannot be set by the user.

Name	Default	Range	Description
Use Memory Initialization File	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. **Note** 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 table below:

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

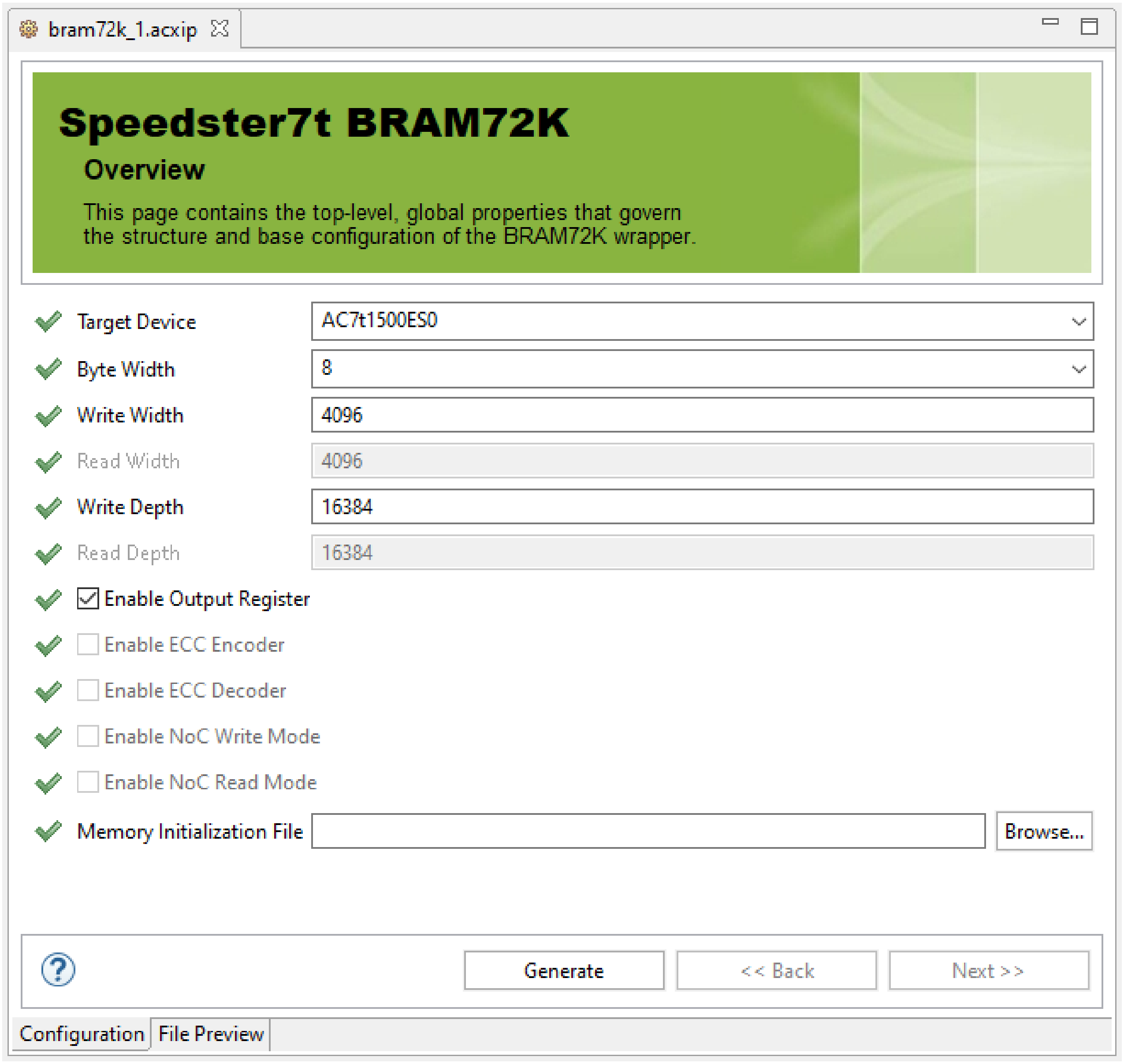
Memory Width	Maximum Memory Depth
1 to 144	1048576
145 to 9216	16384

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:



bram72k_1.acxip

Speedster7t BRAM72K

Overview

This page contains the top-level, global properties that govern the structure and base configuration of the BRAM72K wrapper.

Target Device	AC7t1500ES0
Byte Width	8
Write Width	4096
Read Width	4096
Write Depth	16384
Read Depth	16384

[✓] Enable Output Register
[ ] Enable ECC Encoder
[ ] Enable ECC Decoder
[ ] Enable NoC Write Mode
[ ] Enable NoC Read Mode

Memory Initialization File

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

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

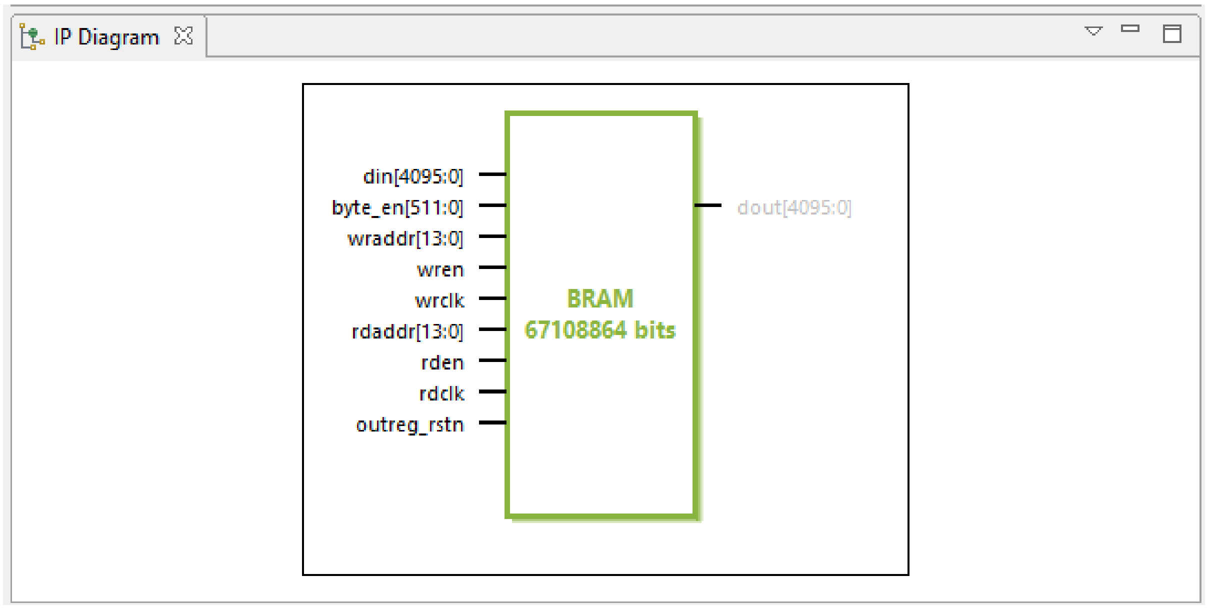


Figure 4: *4096 × 16K Memory IP Diagram*

Chapter - 5: 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.

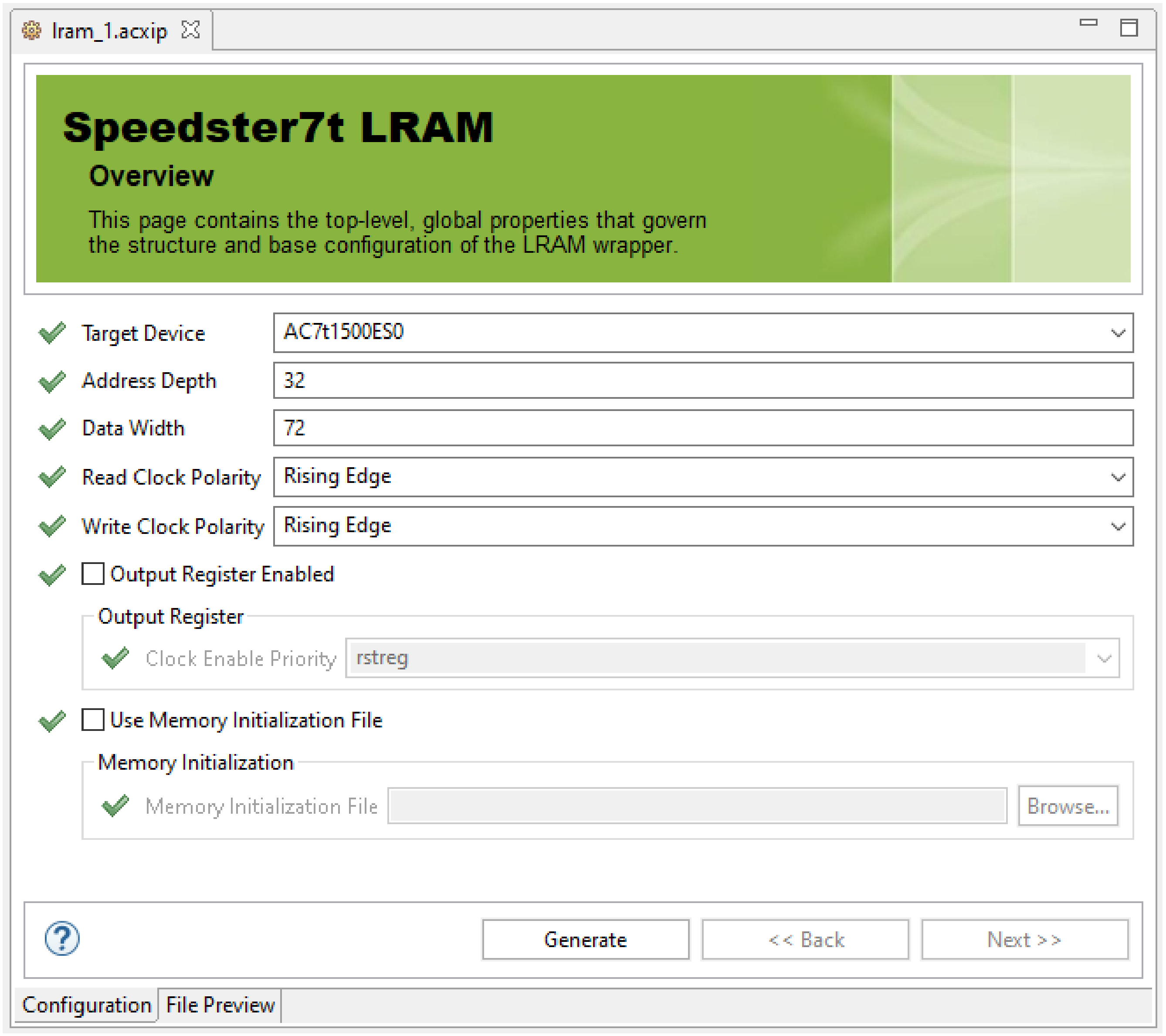
Utilization

Note

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

Configuration

The LRAM2K soft IP configurator has the following configuration options:



Speedster7t LRAM

Overview

This page contains the top-level, global properties that govern the structure and base configuration of the LRAM wrapper.

Target Device	AC7t1500ES0
Address Depth	32
Data Width	72
Read Clock Polarity	Rising Edge
Write Clock Polarity	Rising Edge

☐ Output Register Enabled

Output Register

Clock Enable Priority	rstreg

☐ Use Memory Initialization File

Memory Initialization

Memory Initialization File	

Figure 5: ***LRAM Soft IP Configurator***

Table 3: *Configuration Options*

Name	Default	Range	Description
Target Device	AC7t1500ES0	All Speedster7t devices	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 (see page 20).
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 will occur.
Write Clock Polarity	Rising Edge	Falling or Rising Edge	The `wrclk` active edge on which all write transactions will 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. • **rstreg:** The `outregce` input is ignored when `rstregn` = 1'b0. The output register is reset on the next active `rdclk` edge. • **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.

Name	Default	Range	Description
Use Memory Initialization File	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. **Note** 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 table below:

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

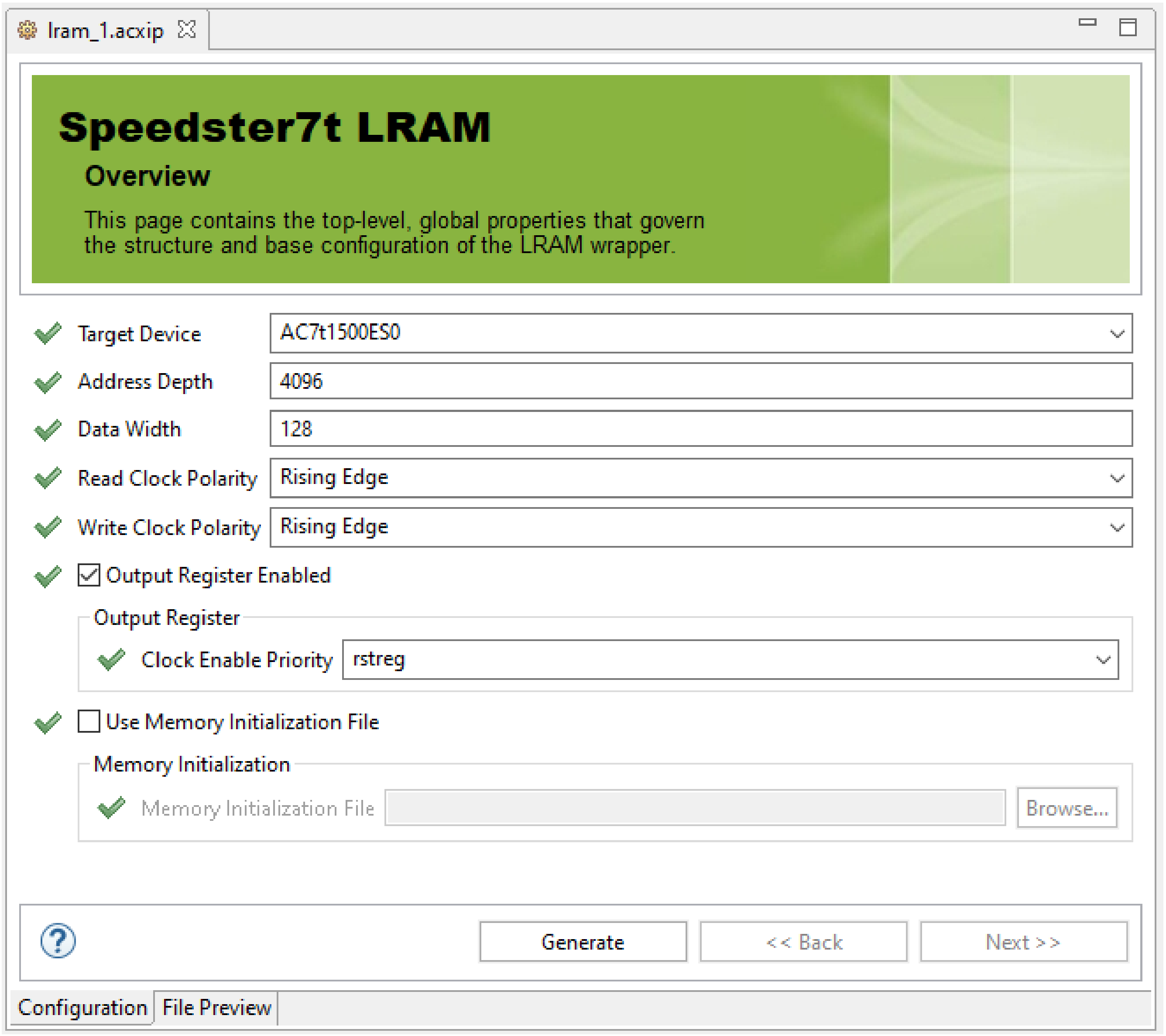
Device Specific Limits

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:


lram_1.acxip

Speedster7t LRAM

Overview

This page contains the top-level, global properties that govern the structure and base configuration of the LRAM wrapper.

Target Device	AC7t1500ES0
Address Depth	4096
Data Width	128
Read Clock Polarity	Rising Edge
Write Clock Polarity	Rising Edge

☑ Output Register Enabled

Output Register

Clock Enable Priority	rstreg

☐ Use Memory Initialization File

Memory Initialization

Memory Initialization File

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

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

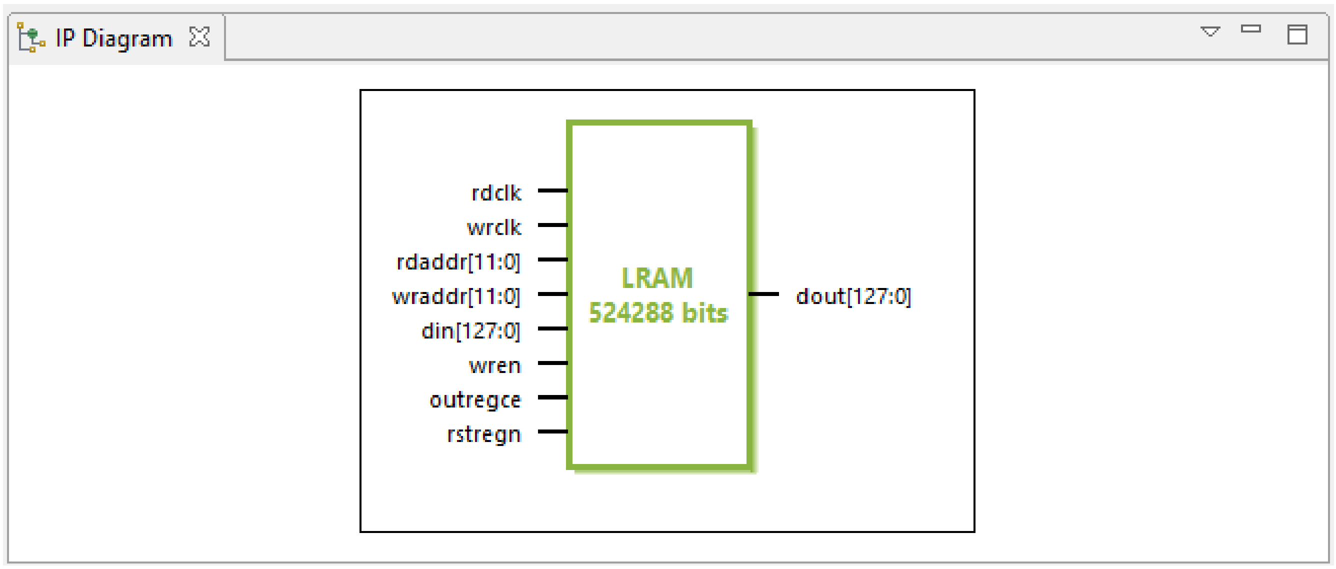


Figure 7: ***128-Bit x 4096-Word LRAM Memory IP Diagram***

Chapter - 6: ROM Soft IP

Description

The Speedster7t ROM soft IP core creates an arbitrary sized ROM, using either BRAM or LRAM primitives.

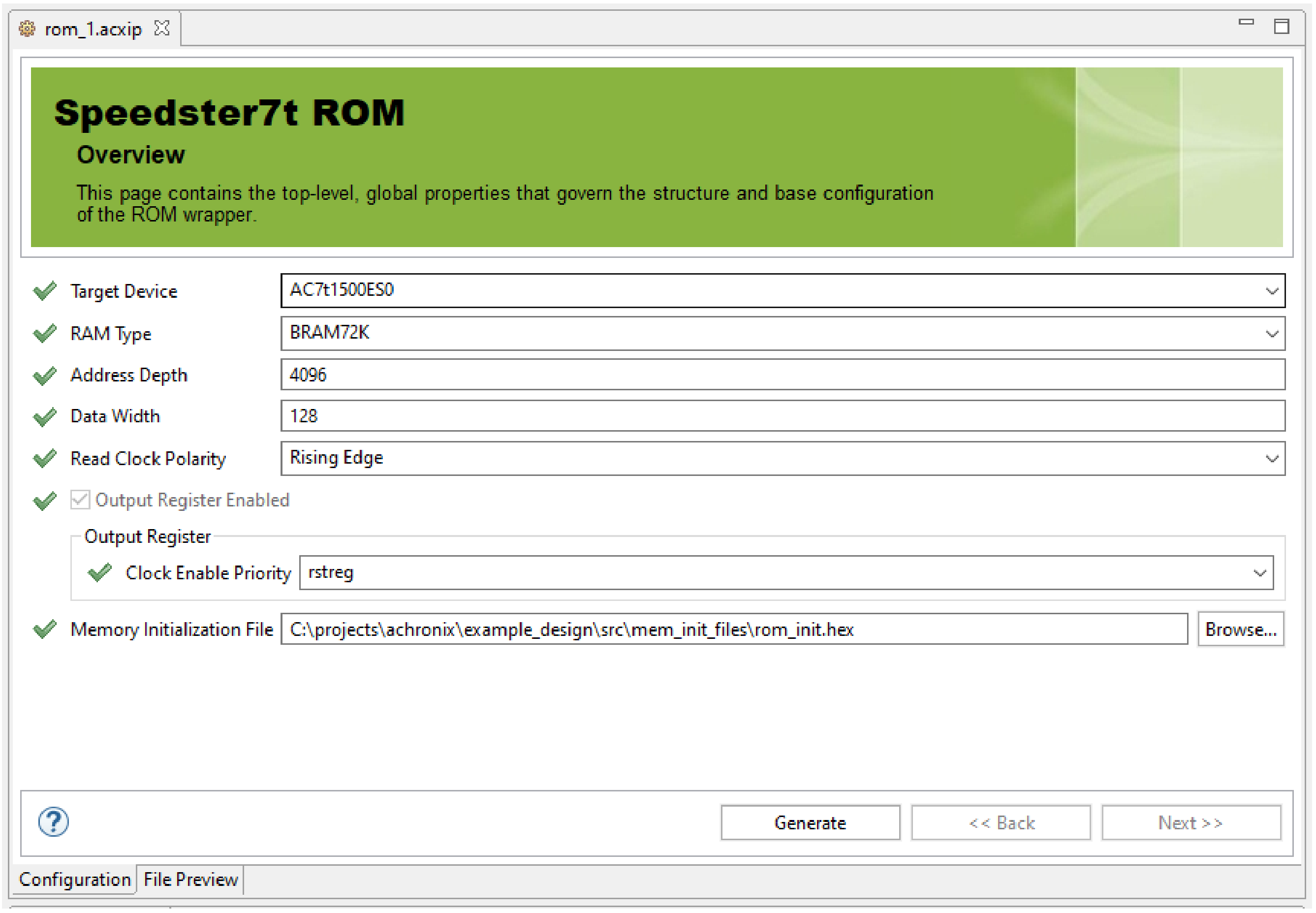
Utilization

Note

Within the Speedster7t family, the LRAM and MLP primitives share a site. Therefore, if using a Speedster7t device, and if the ROM soft IP configuration selects an LRAM to implement the ROM, the MLPs on the sites used by the ROM cannot be used.

Configuration

The soft IP has the following configuration options:



Speedster7t ROM

Overview

This page contains the top-level, global properties that govern the structure and base configuration of the ROM wrapper.

Target Device	AC7t1500ES0
RAM Type	BRAM72K
Address Depth	4096
Data Width	128
Read Clock Polarity	Rising Edge
Output Register Enabled	✓
Output Register: Clock Enable Priority	rstreg
Memory Initialization File	C:\projects\achronix\example_design\src\mem_init_files\rom_init.hex

Figure 8: ***ROM Soft IP Configuration***

Table 5: ***Configuration Options***

Name	Default	Range	Description
Target Device	AC7t1500ES0	All Speedster7t devices	Set to match the target device of the project.
RAM Type	BRAM72K	BRAM72K, LRAM2K	Determines which type of RAM primitives used to form the ROM.
Address Depth	1024	4 to 16384	Address depth of the ROM in words.
Data Width	20	1 to 184320	Port width in bits of the `dout` port. The width selected affects the available address depth. The maximum values of width versus depth are detailed in Read and Write Depths Versus Data Widths (see page 28).
Read Clock Polarity	Rising Edge	Falling or Rising Edge	The `rdclk` active edge on which all read transactions will occur.
Enable Output Register	Off	On, Off	Determines whether the output register in each of the LRAM2K primitives is enabled. 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. • **rstreg**: The `outregce` input is ignored when `rstregn` = 1'b0. The output register is reset on the next active `rdclk` edge. • **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.

Name	Default	Range	Description
Use Memory Initialization File	On	On	Location of the memory initialization file used to initialize the memory contents. This initialization occurs for both synthesis and simulation. **Note** 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 of these values is detailed in the table below:

Table 6: ***Write and Read Depths Versus Data Width***

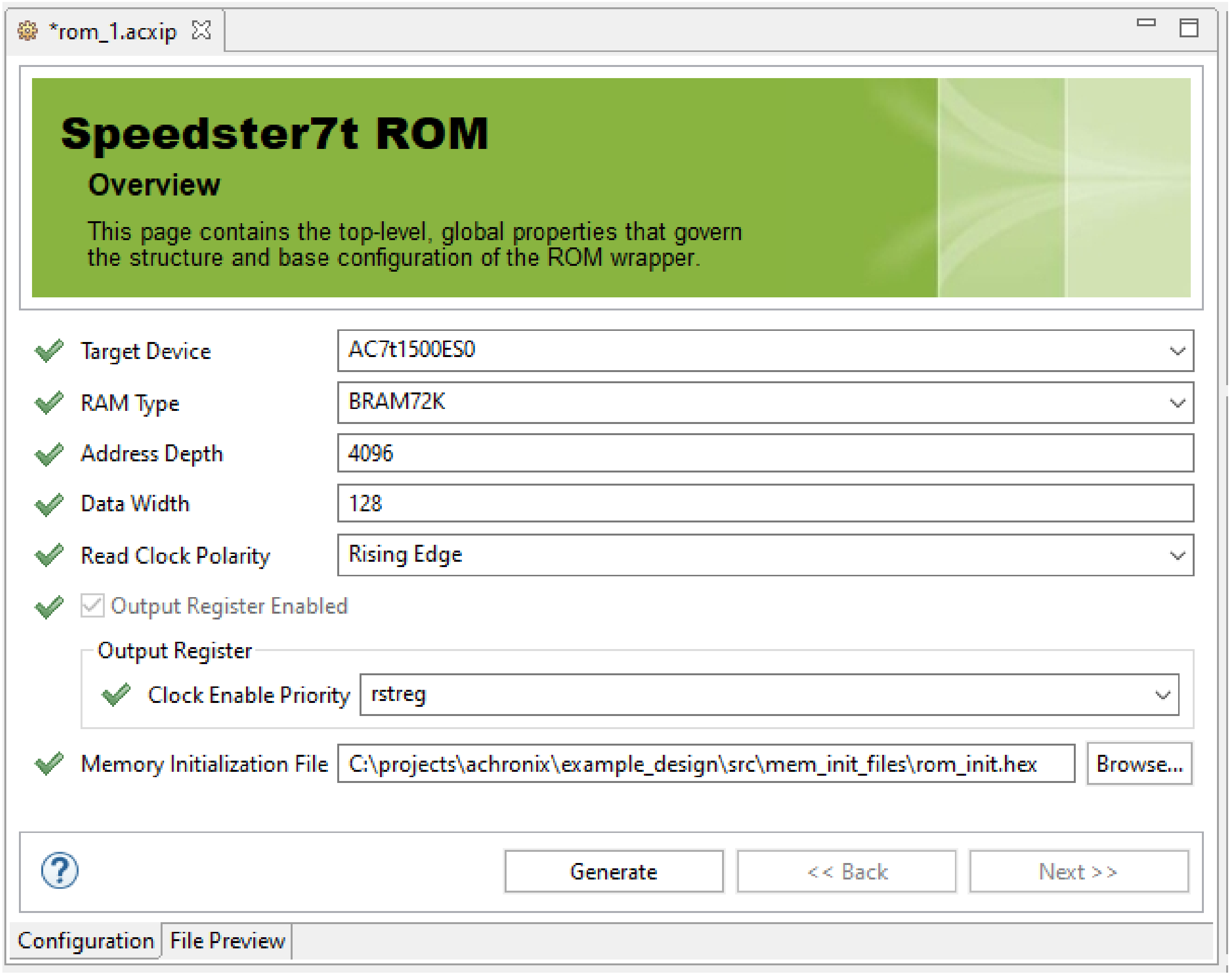
RAM Type	Memory Width	Maximum Memory Depth
ACX_BRAM72K	1 to 11520	16384
	11521 to 23040	8096
	23041 to 46080	4096
	46081 to 92160	2048
	92161 to 184320	1024
ACX_LRAM2K	1–1440	4096
	1441 to 2880	2048
	2881 to 5760	1024
	5761 to 11520	512
	11521 to 23040	256
	23041 to 46080	128
	46081 to 92160	64
	92161 to 184320	32

Device Specific Limits

Within a device, the largest memory that can be generated is limited by the number of RAM primitives in a column, and the choice of RAM type. The overall ROM limit in bits is equivalent to the number of RAM primitives in a column multiplied by the number of bits in the primitive, (72K for ACX_BRAM72K, and 2K for ACX_LRAM2K).

Examples

The following figure shows the soft IP configured for a 128-bit × 4096-word ROM formed of BRAM72K primitives with the memory output register enabled.



This page contains the top-level, global properties that govern the structure and base configuration of the ROM wrapper.

Target Device	AC7t1500ES0
RAM Type	BRAM72K
Address Depth	4096
Data Width	128
Read Clock Polarity	Rising Edge
Output Register Enabled	✓

Output Register

Clock Enable Priority	rstreg
Memory Initialization File	C:\projects\achronix\example_design\src\mem_init_files\rom_init.hex

Figure 9: *128-bit × 4096-word ROM Configuration*

The following figure shows the soft IP IO diagram for the above configuration.

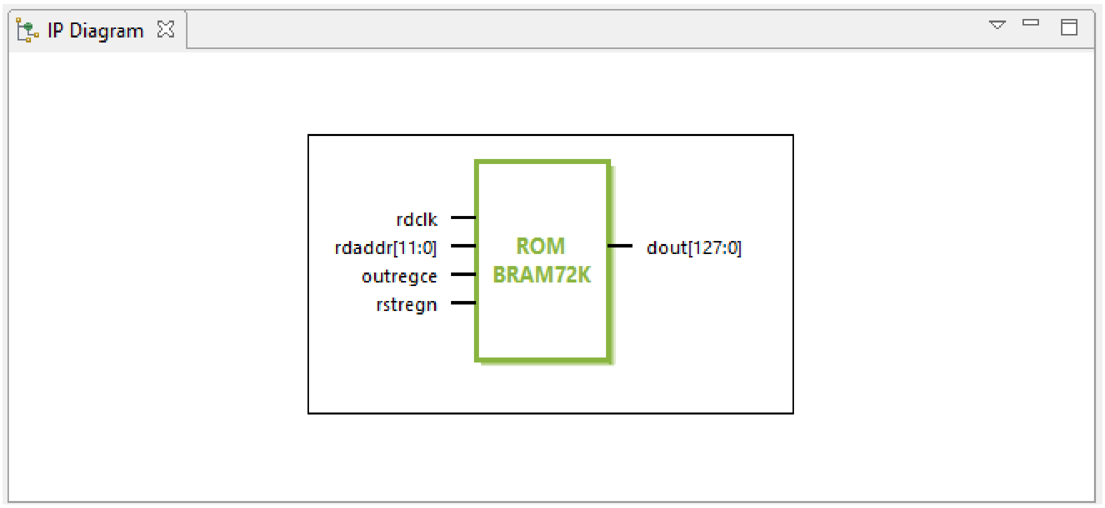


Figure 10: *128-Bit × 4096-Word ROM IP Diagram*

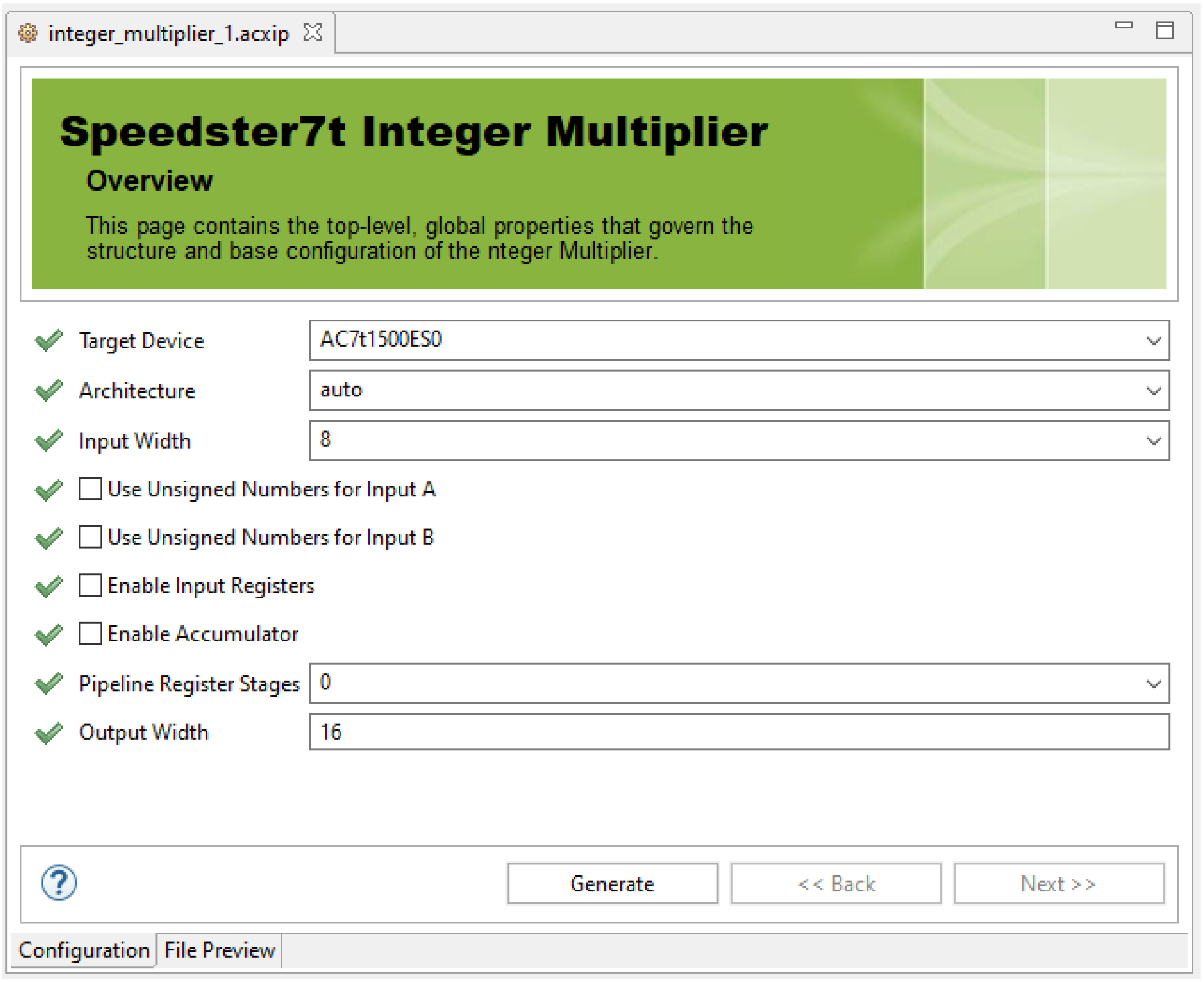
Chapter - 7: Integer Multiplier Soft IP

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:



Speedster7t Integer Multiplier

Overview

This page contains the top-level, global properties that govern the structure and base configuration of the nteger Multiplier.

✓ Target Device	AC7t1500ES0
✓ Architecture	auto
✓ Input Width	8
✓ ☐ Use Unsigned Numbers for Input A	
✓ ☐ Use Unsigned Numbers for Input B	
✓ ☐ Enable Input Registers	
✓ ☐ Enable Accumulator	
✓ Pipeline Register Stages	0
✓ Output Width	16

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

Table 7: *Configuration Options*

Name	Default	Range	Description
Target Device	AC7t1500ES0	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. • **auto**: Allow the tool to choose the most appropriate primitive based on the number formats and sizes selected. • **rlb**: Implement the multiplier in fabric logic. The Speedster7t FPGA has a unique MLUT structure which supports very efficient multiplier arrays using logic. • **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.
Enable Accumulator	Off	On, Off	The output is the accumulation of the result from each clock cycle. Enabling accumulation adds the input `i_load` to the resultant soft IP. The accumulation is cleared when `i_load` is asserted, the output is reset to `i_din_a * i_din_b`. The `i_load` signal has the same pipeline delay to the accumulator as the `i_din_a` and `i_din_b` inputs. Therefore it should be applied on the same cycle as the `i_din_a` and `i_din_b` inputs that are to start a new accumulation cycle.
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. When any pipeline stages are enabled, the inputs `i_pipeline_ce` and `i_pipeline_rstn` are added to the resultant soft IP.

Name	Default	Range	Description
Output Width	16	8 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. The valid range changes dependent upon the Input Width and Architecture. **Note** 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 minimum output width can be calculated as `(2*Input Width)+(number of accumulation cycles)`.

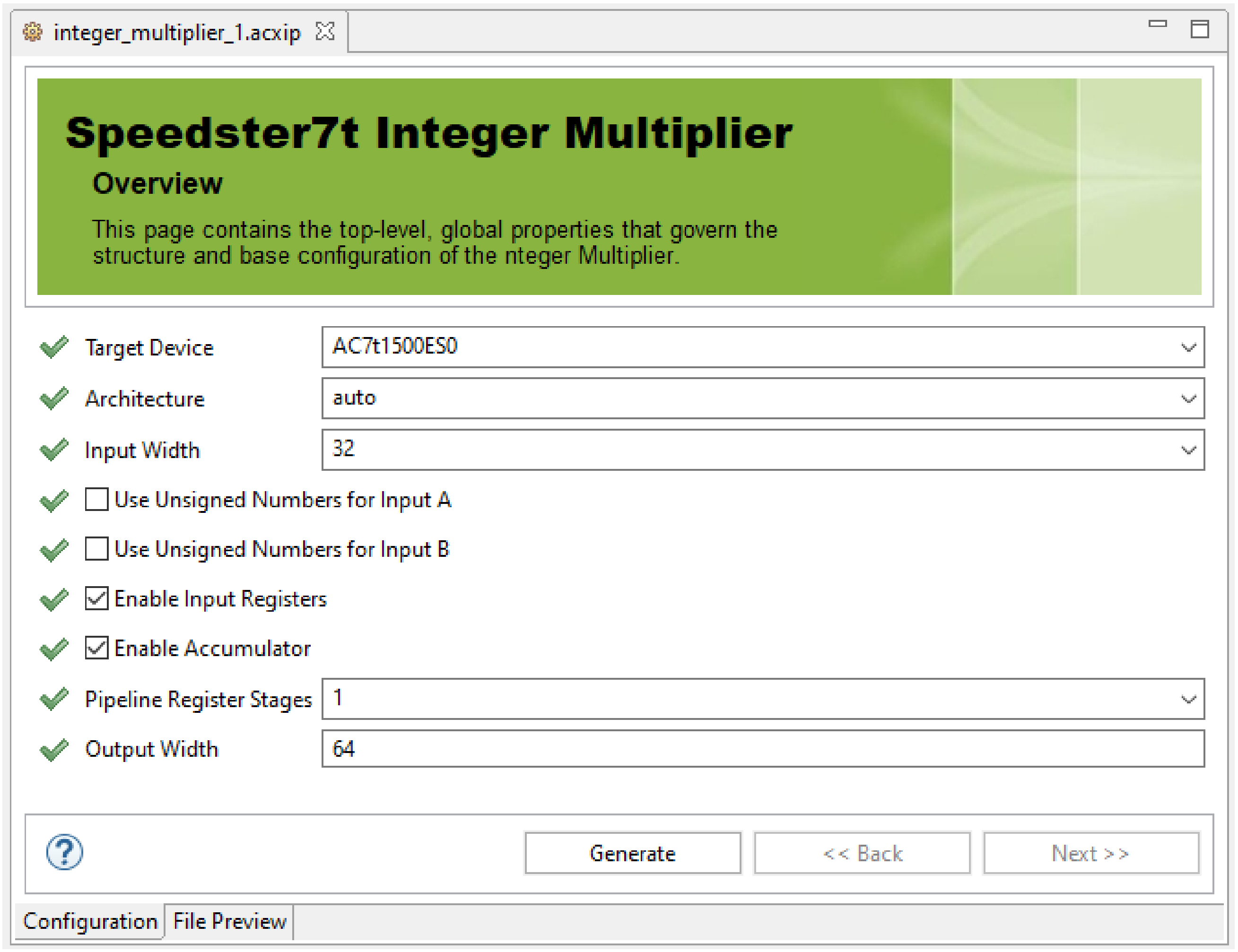
Table 8: *Ports*

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 either **Enable Accumulator** is set to **On** or **Pipeline Register Stages** is greater than 0.
`i_pipeline_rstn`	Input	(Optional) Synchronous active-low reset for pipeline and accumulator registers. Present when either **Enable Accumulator** is set to **On** or **Pipeline Register Stages** is greater than 0.

Name	Direction	Description
`i_load`	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**. **Note** This signal is internally pipelined to have the same latency as `i_din_a` and `i_din_b`.
`o_dout`[(Output Width – 1):0]	Output	Result of multiplication and accumulation.

Examples

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



Speedster7t Integer Multiplier

Overview

This page contains the top-level, global properties that govern the structure and base configuration of the nteger Multiplier.

Target Device	AC7t1500ES0
Architecture	auto
Input Width	32
☐ Use Unsigned Numbers for Input A	
☐ Use Unsigned Numbers for Input B	
☑ Enable Input Registers	
☑ Enable Accumulator	
Pipeline Register Stages	1
Output Width	64

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

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

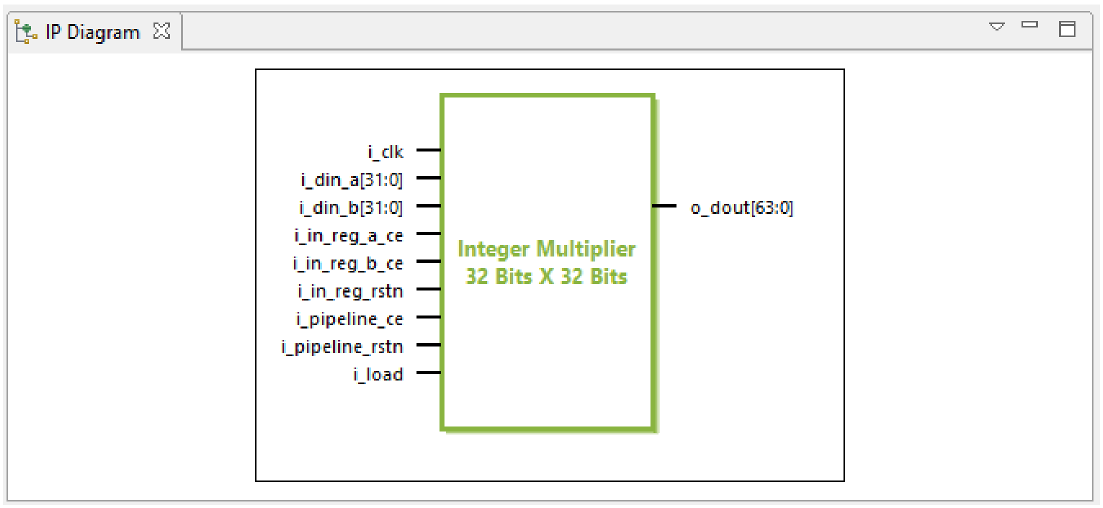


Figure 13: ***32 x 32 Signed Integer Multiplier IO***

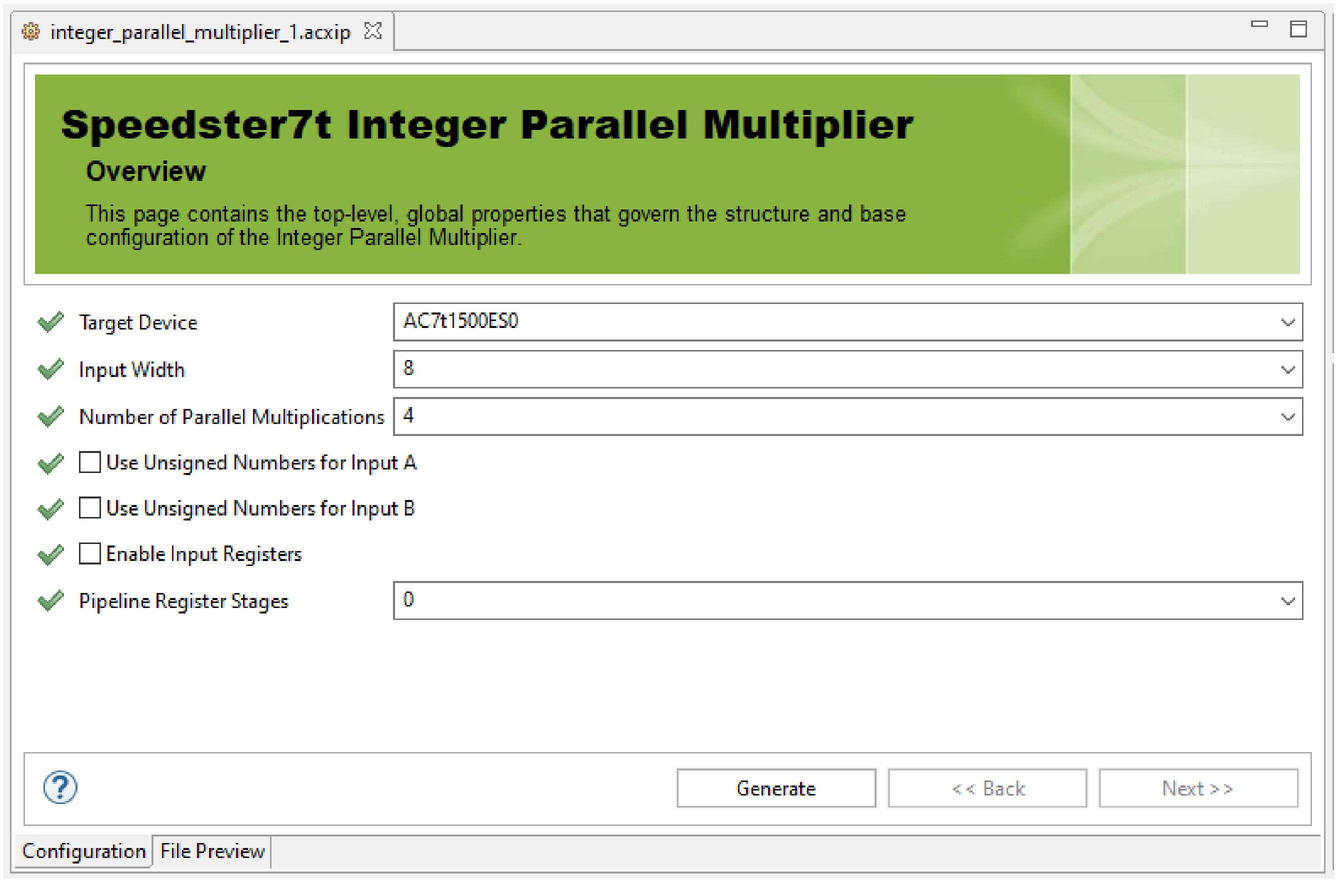
Chapter - 8: Integer Parallel Multiplier Soft IP

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:


integer_parallel_multiplier_1.acxip
Speedster7t Integer Parallel Multiplier
Overview
Generate
<< Back
Next >>
Configuration
File Preview

This page contains the top-level, global properties that govern the structure and base configuration of the Integer Parallel Multiplier.

✓ Target Device	AC7t1500ES0
✓ Input Width	8
✓ Number of Parallel Multiplications	4
✓ ☐ Use Unsigned Numbers for Input A	
✓ ☐ Use Unsigned Numbers for Input B	
✓ ☐ Enable Input Registers	
✓ Pipeline Register Stages	0

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

Table 9: ***Configuration Options***

Name	Default	Range	Description
Target Device	AC7t1500ES0	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 table Number of Multipliers Per Input Width (see page 40), below, 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: • `i_in_reg_a_ce` • `i_in_reg_b_ce` • `i_in_reg_rstn`
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: • `i_pipeline_ce` • `i_pipeline_rstn`

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

Input Width	Maximum Number of Multipliers
3	8
4	8
5	4
6	4
7	4
8	4
16	2

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 (see page 41) vector of data to 'A' inputs of multipliers.
`i_din_b`[(Input Width – 1):0]	Input	Packed (see page 41) 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**.
`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 (see page 41) 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

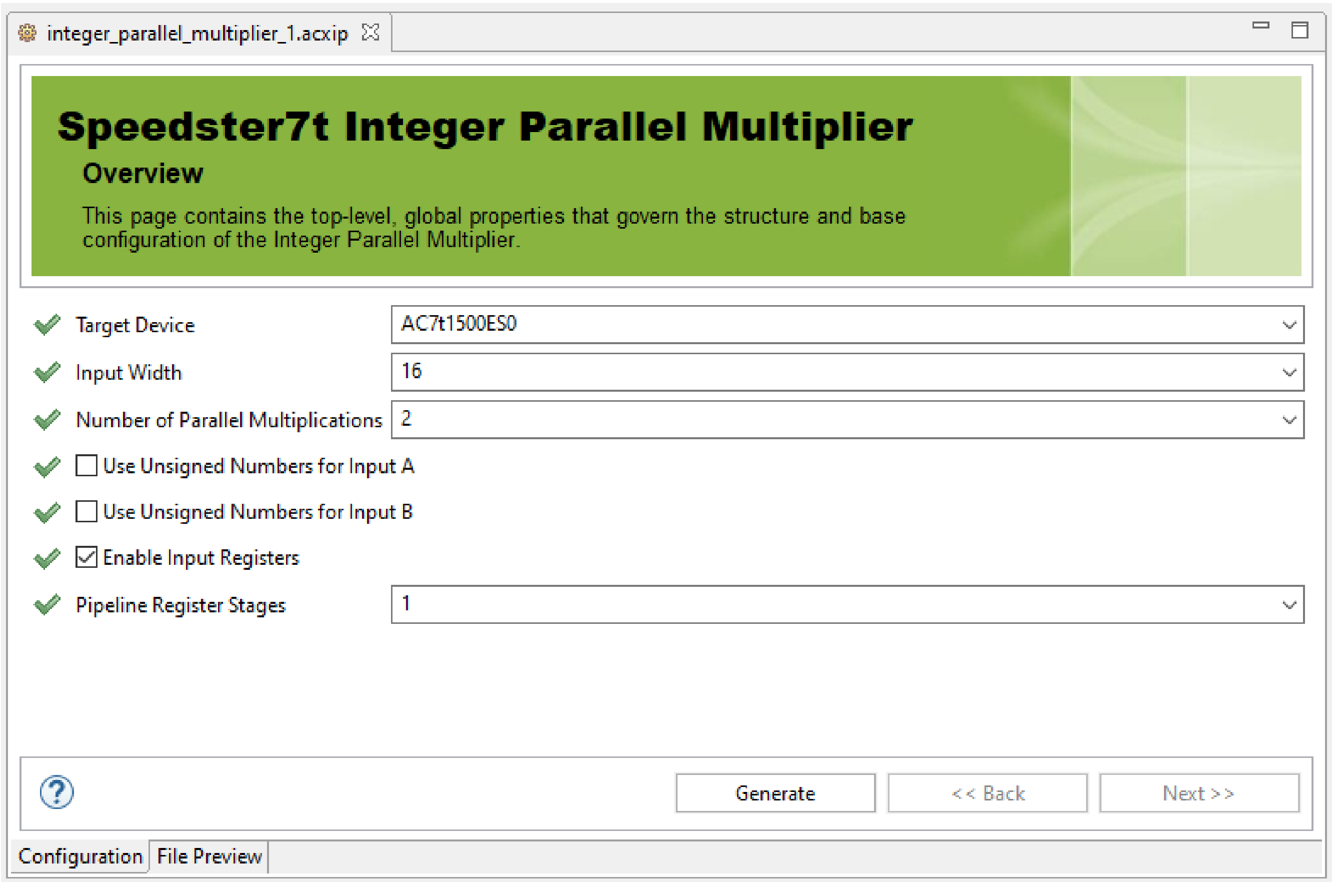
Each individual multiplier result appears in the lanes detailed in the table below.

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:


integer_parallel_multiplier_1.acxip

Speedster7t Integer Parallel Multiplier

Overview

This page contains the top-level, global properties that govern the structure and base configuration of the Integer Parallel Multiplier.

Target Device	AC7t1500ES0
Input Width	16
Number of Parallel Multiplications	2
☐ Use Unsigned Numbers for Input A	
☐ Use Unsigned Numbers for Input B	
☑ Enable Input Registers	
Pipeline Register Stages	1

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

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

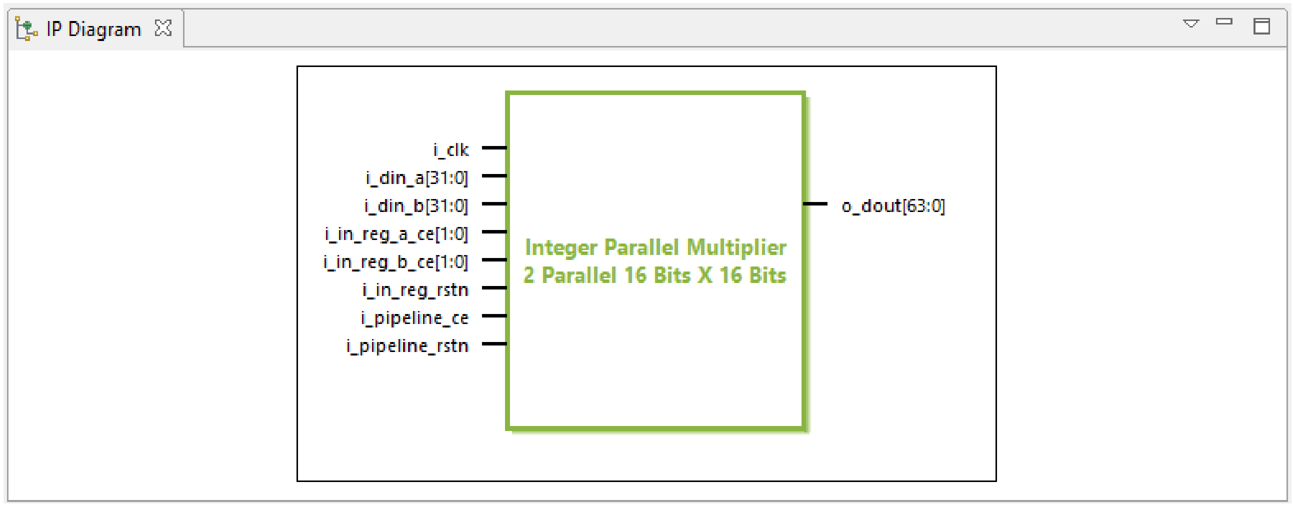


Figure 16: ***Two 16 × 16 Signed Parallel Integer Multiplier I/O***

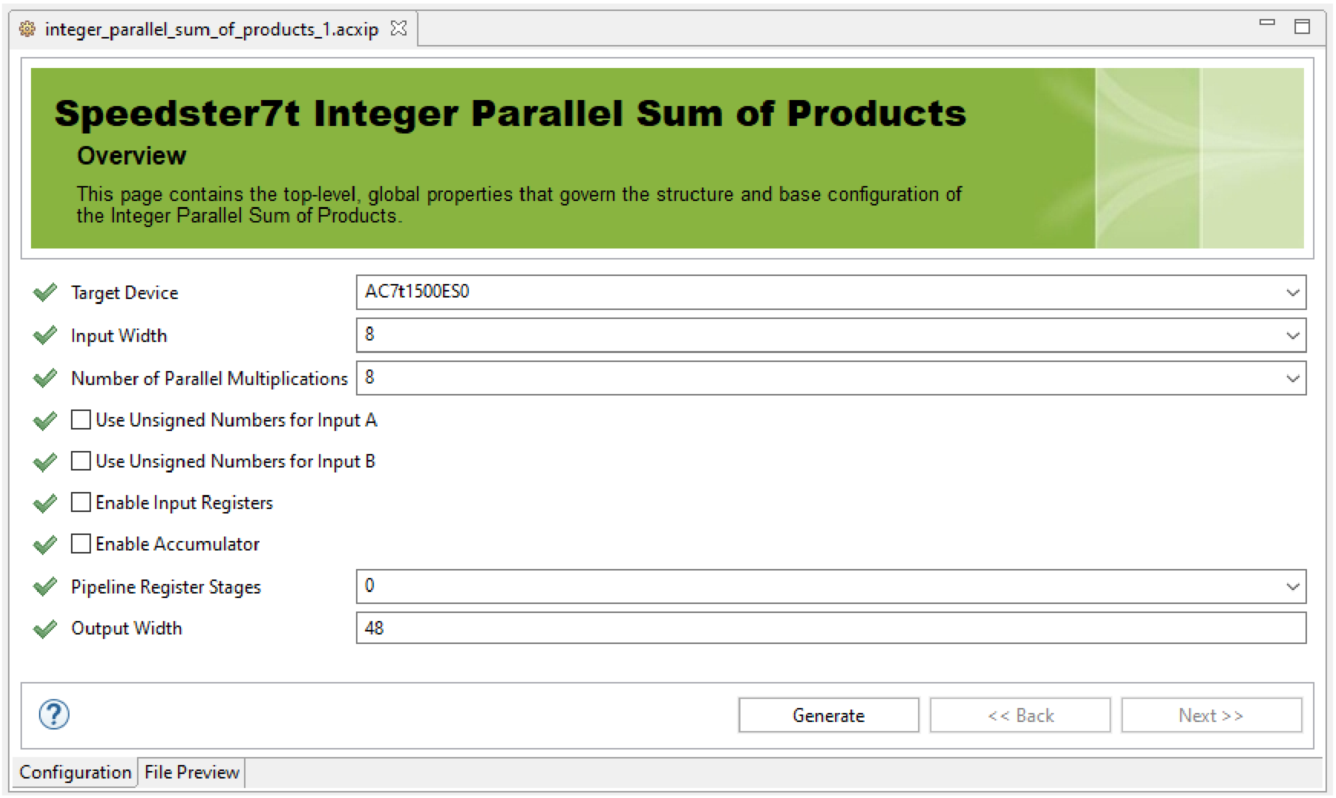
Chapter - 9: Integer Parallel Sum of Products Soft IP

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:



Speedster7t Integer Parallel Sum of Products

Overview

This page contains the top-level, global properties that govern the structure and base configuration of the Integer Parallel Sum of Products.

Target Device	AC7t1500ES0
Input Width	8
Number of Parallel Multiplications	8
☐ Use Unsigned Numbers for Input A	
☐ Use Unsigned Numbers for Input B	
☐ Enable Input Registers	
☐ Enable Accumulator	
Pipeline Register Stages	0
Output Width	48

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

Table 13: ***Configuration Options***

Name	Default	Range	Description
Target Device	AC7t1500ES0	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.

Name	Default	Range	Description
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: • `i_in_reg_a_ce` • `i_in_reg_b_ce` • `i_in_reg_rstn`
Enable Accumulator	Off	On, Off	The output is the accumulation of the result from each clock cycle. Enabling accumulation adds the input `i_load` to the resultant soft IP core. The accumulation is cleared when `i_load` is asserted.
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: • `i_pipeline_ce` • `i_pipeline_rstn`
Output Width	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. **Note** 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 minimum output width can be calculated as: (2 × **Input Width** × **Number of Parallel Multiplications**) + (number of accumulation cycles)

Table 14: ***Maximum Number of Multipliers Per Input Width***

Input Width	Maximum Number of Multipliers
3	24
4	16
5	12
6	12
7	10
8	8
16	4

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 (see page 48) 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 (see page 48) 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 registers. Present when either **Enable Accumulator** is set to **On** or **Pipeline Register Stages** is greater than 0.
`i_pipeline_rstn`	Input	(Optional) Synchronous active-low reset for pipeline registers. Present when either **Enable Accumulator** is set to **On** or **Pipeline Register Stages** is greater than 0.
`i_load`	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. **Note** This signal is internally pipelined to have the same latency as `i_din_a` and `i_din_b`.
`o_dout`[(Output Width – 1):0]	Output	Output bus consisting of the sum of products from all of the multipliers in parallel.

Input Format

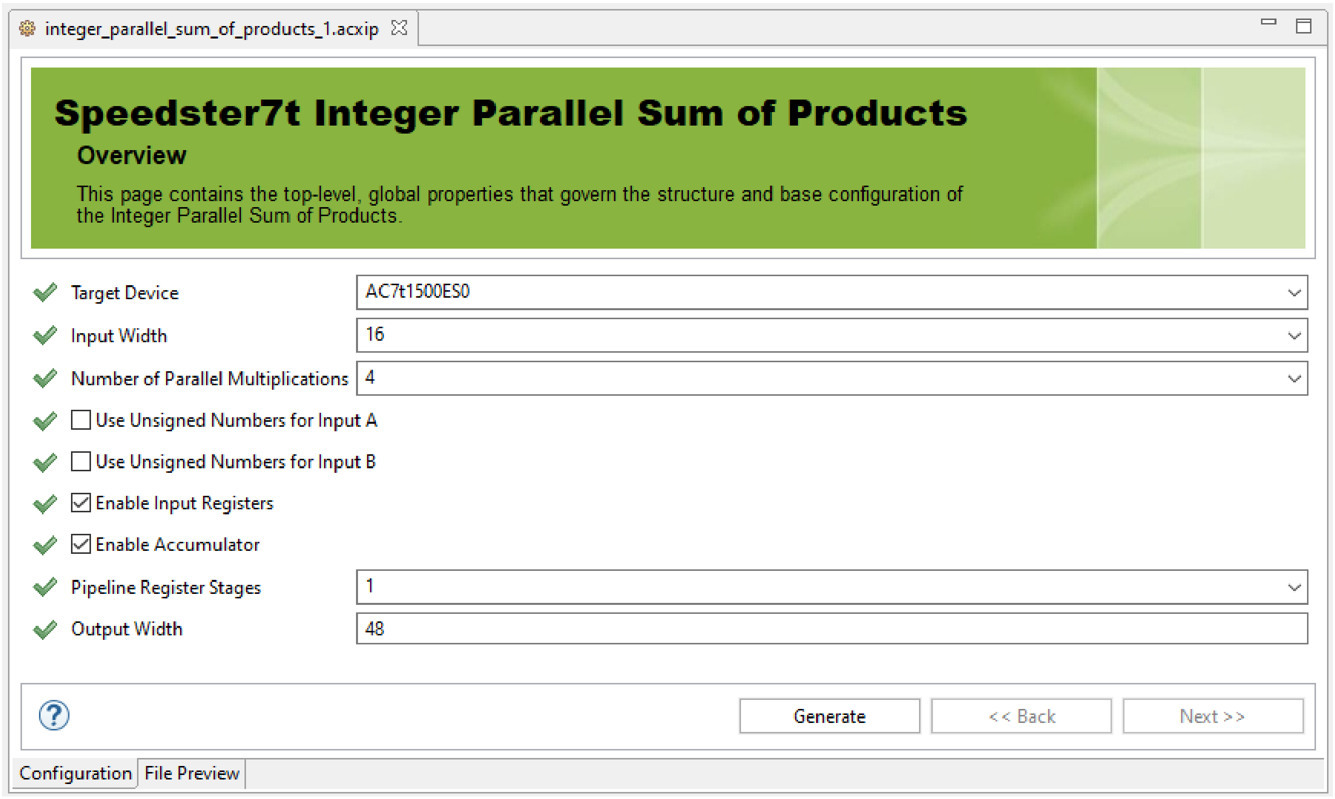
Each multiplier input is formed from an array of the individual inputs, packed in a single input vector.

Code

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

Examples

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



Speedster7t Integer Parallel Sum of Products

Overview

This page contains the top-level, global properties that govern the structure and base configuration of the Integer Parallel Sum of Products.

Target Device	AC7t1500ES0
Input Width	16
Number of Parallel Multiplications	4
☐ Use Unsigned Numbers for Input A	
☐ Use Unsigned Numbers for Input B	
☑ Enable Input Registers	
☑ Enable Accumulator	
Pipeline Register Stages	1
Output Width	48

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

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

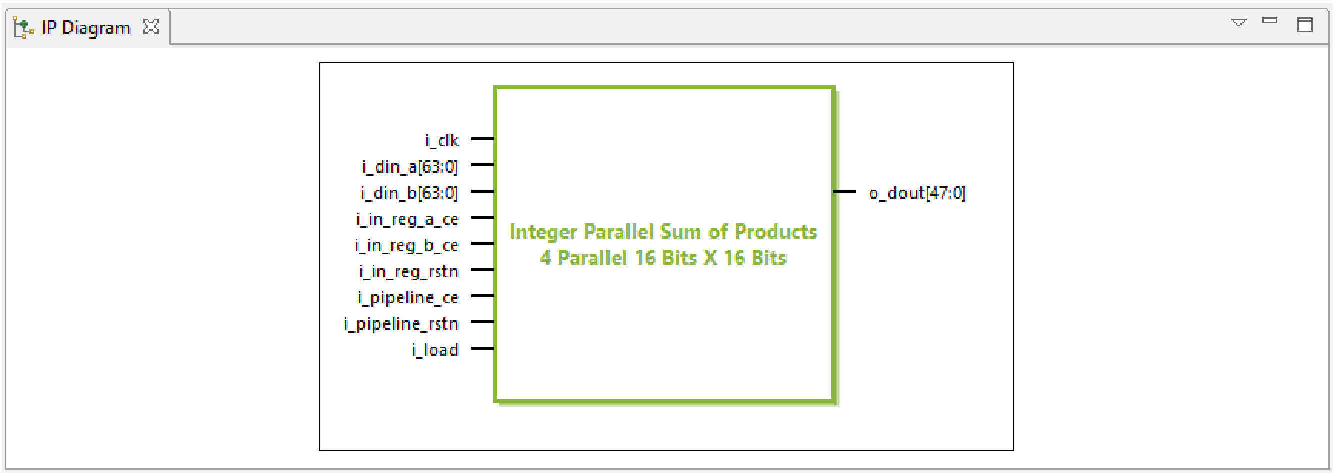


Figure 19: ***Four 16 × 16 Signed Multiplier Sum of Products I/O***

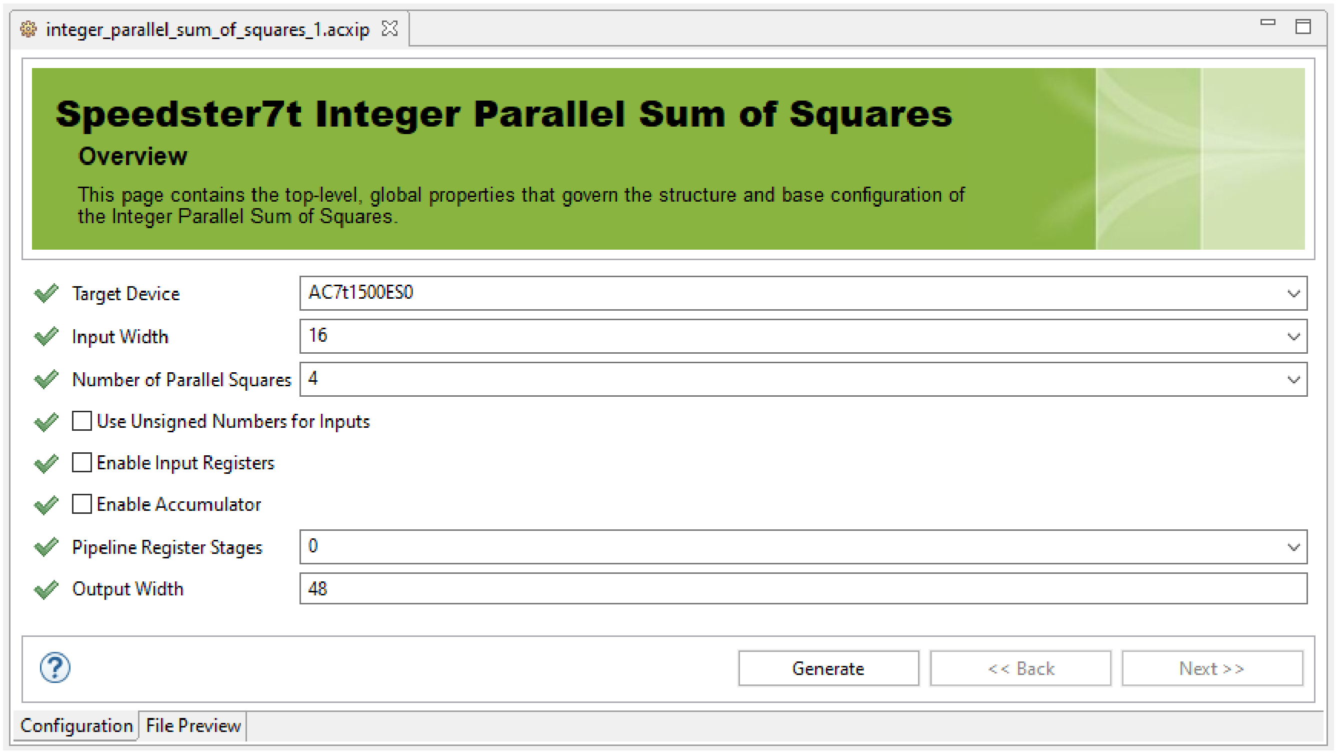
Chapter - 10: Integer Parallel Sum of Squares Soft IP

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:



Speedster7t Integer Parallel Sum of Squares

Overview

This page contains the top-level, global properties that govern the structure and base configuration of the Integer Parallel Sum of Squares.

✓ Target Device	AC7t1500ES0
✓ Input Width	16
✓ Number of Parallel Squares	4
✓ ☐ Use Unsigned Numbers for Inputs	
✓ ☐ Enable Input Registers	
✓ ☐ Enable Accumulator	
✓ Pipeline Register Stages	0
✓ Output Width	48

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

Table 16: ***Configuration Options***

Name	Default	Range	Description
Target Device	AC7t1500ES0	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 below 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.

Name	Default	Range	Description
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: • `i_in_reg_ce` • `i_in_reg_rstn`
Enable Accumulator	Off	On, Off	Output is the accumulation of the result from each clock cycle. Enabling accumulation adds these inputs to the resultant soft IP core: • `i_load` • `i_pipeline_ce` • `i_pipeline_rstn` The accumulation is cleared when `i_load` is asserted.
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: • `i_pipeline_ce` • `i_pipeline_rstn`
Output Width	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. **Note** 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 minimum output width can be calculated as: (2 × **Input Width** × **Number of Parallel Squares**) + (number of accumulation cycles)

Table 17: *Maximum Number of Multipliers Per Input Width*

Input Width	Maximum Number of Multipliers
3	32
4	32
5	16
6	16
7	16
8	16
16	4

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 (see page 54) 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 pipeline registers. Present when either **Enable Accumulator** is set to **On** or **Pipeline Register Stages** is greater than 0.
`i_pipeline_rstn`	Input	(Optional) Synchronous active-low reset for pipeline registers. Present when either **Enable Accumulator** is set to **On** or **Pipeline Register Stages** is greater than 0.
`i_load`	Input	(Optional) When asserted to 1'b1, resets the accumulator to `i_din2` ignoring the previous value. Present when **Enable Accumulator** is set to **On**. **Note** This signal is internally pipelined to have the same latency as `i_din`.
`o_dout[`(Output Width – 1):0]	Output	Output bus consisting of the sum of squares from all of the multipliers in parallel.

Input Packing

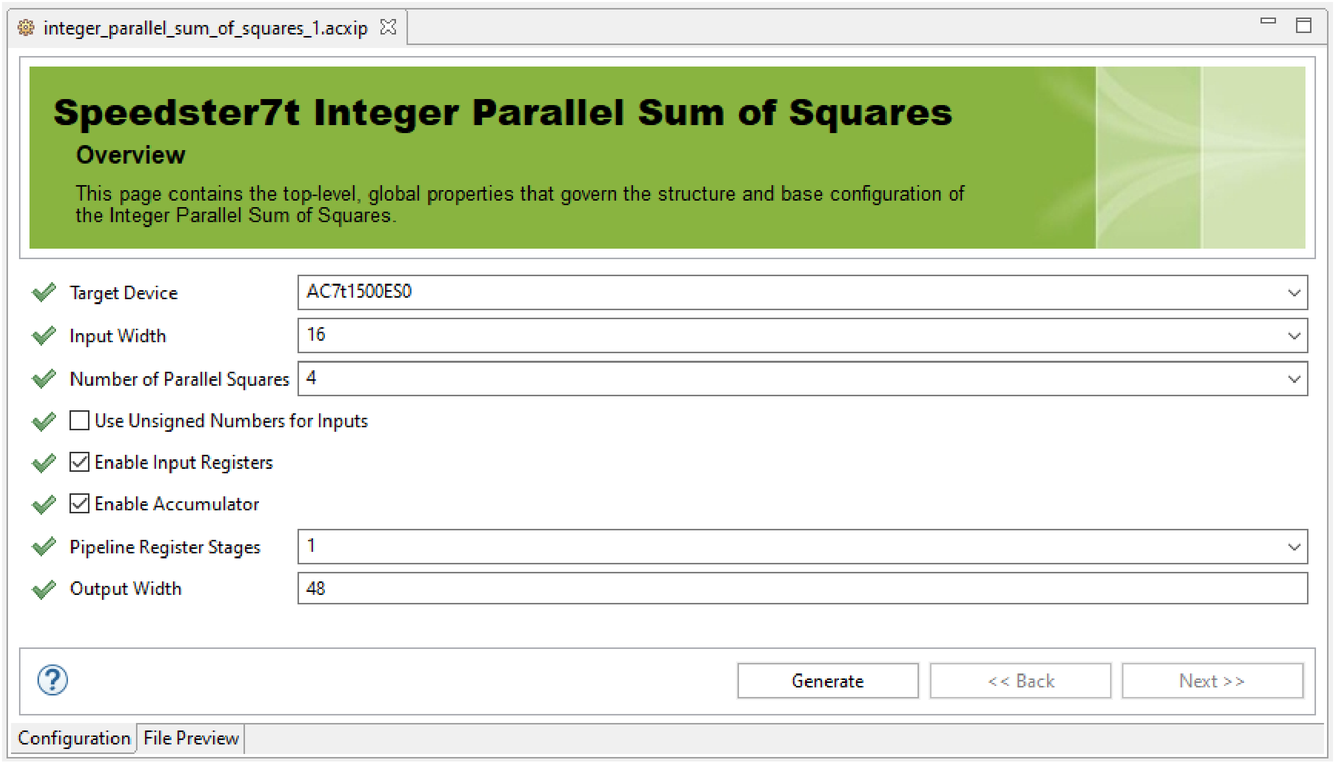
Inputs are packed in a single input vector.

Code

```
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:



integer_parallel_sum_of_squares_1.acxip

Speedster7t Integer Parallel Sum of Squares

Overview

This page contains the top-level, global properties that govern the structure and base configuration of the Integer Parallel Sum of Squares.

Target Device	AC7t1500ES0
Input Width	16
Number of Parallel Squares	4
☐ Use Unsigned Numbers for Inputs	
☑ Enable Input Registers	
☑ Enable Accumulator	
Pipeline Register Stages	1
Output Width	48

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

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

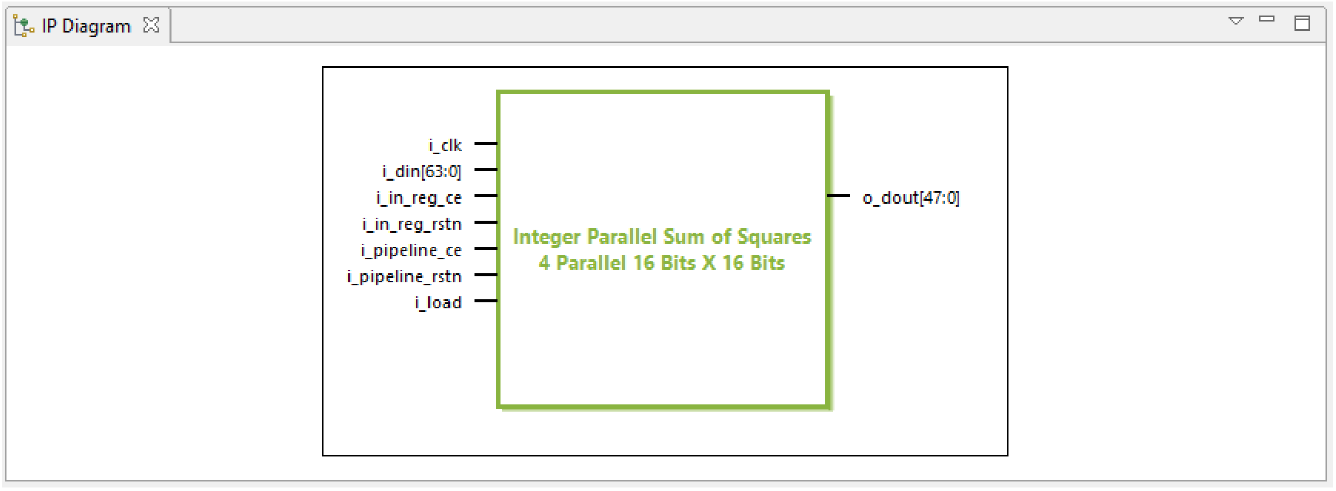


Figure 22: *Four 16 × 16 Signed Multiplier Sum of Squares I/O*

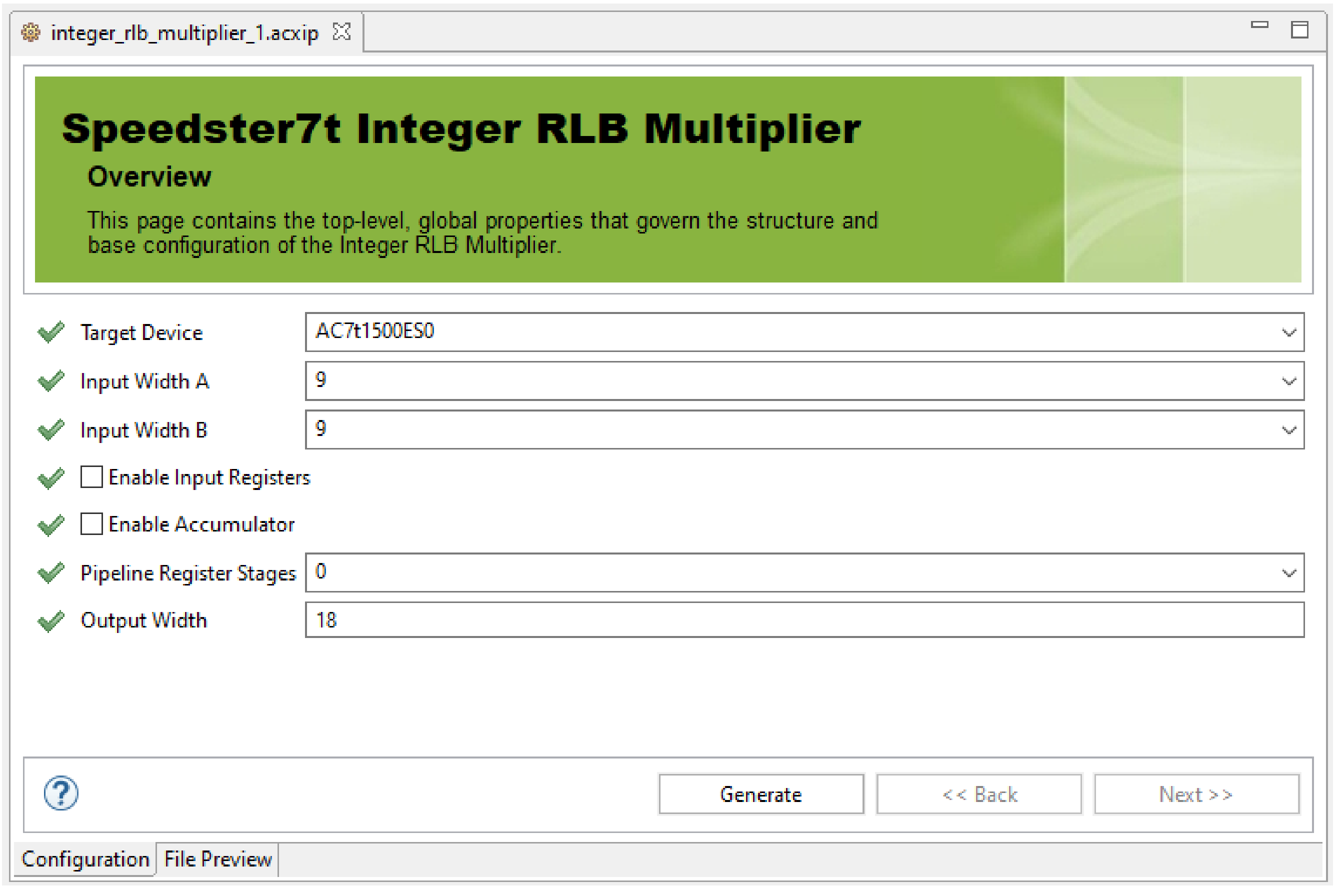
Chapter - 11: Integer RLB Multiplier Soft IP

Description

The Integer 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:


integer_rlb_multiplier_1.acxip

Speedster7t Integer RLB Multiplier

Overview

This page contains the top-level, global properties that govern the structure and base configuration of the Integer RLB Multiplier.

Target Device	AC7t1500ES0
Input Width A	9
Input Width B	9
☐ Enable Input Registers	
☐ Enable Accumulator	
Pipeline Register Stages	0
Output Width	18

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

Table 19: *Configuration Options*

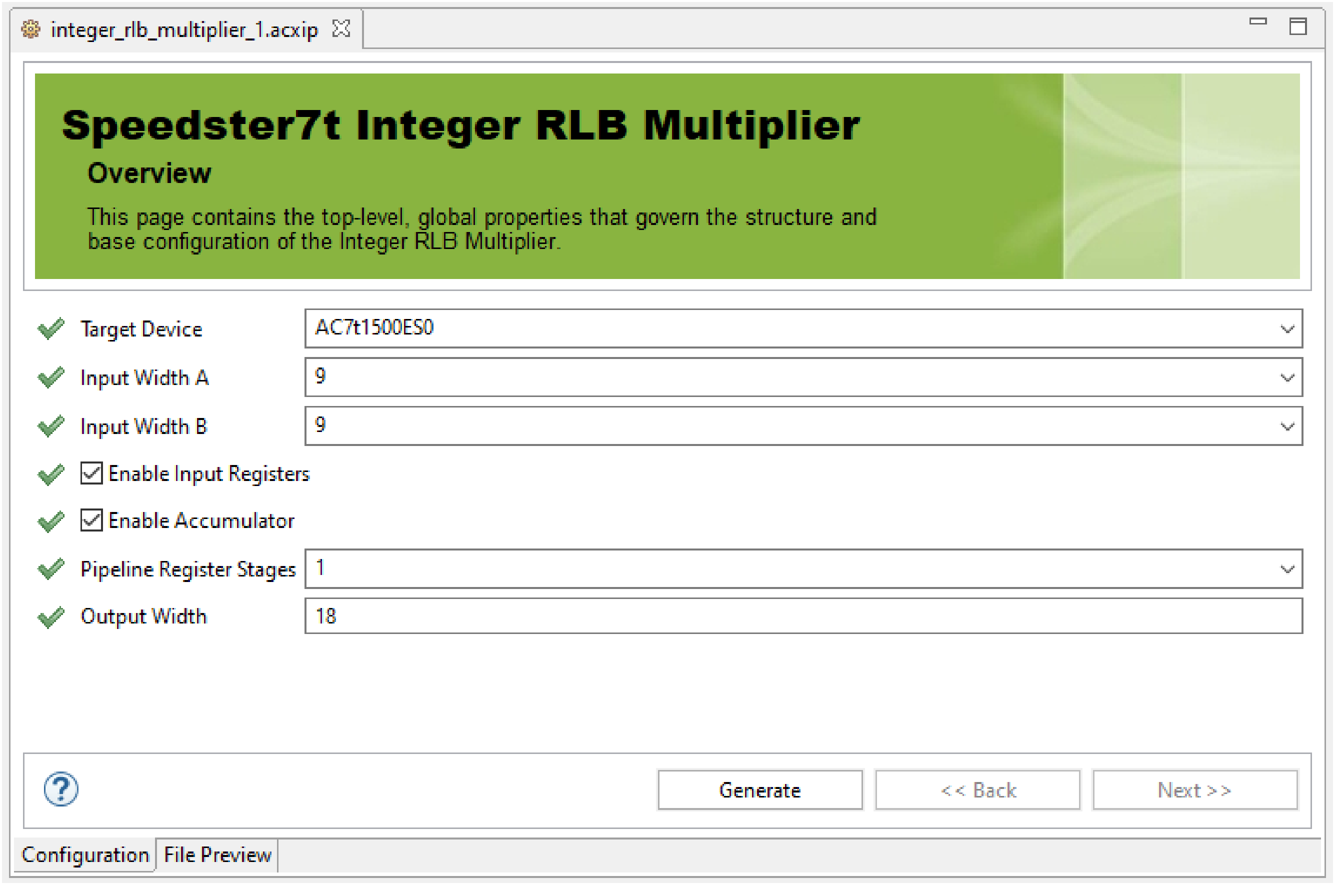
Name	Default	Range	Description
Target Device	AC7t1500ES0	All Speedster7t devices	Set to match the target device of the project.
Input Width A	9	3 to 9	Width of 'A' inputs.
Input Width B	9	3 to 9	Width of 'B' inputs.
Enable Input Registers	Off	On, Off	When set, enables a register stage for both 'A' and 'B' inputs. Adds a cycle of latency to all results. Enabling the input registers adds these inputs to the resultant soft IP core: • `i_in_reg_a_ce` • `i_in_reg_b_ce` • `i_in_reg_rstn`
Enable Accumulator	Off	On, Off	Output is the accumulation of the result from each clock cycle. Enabling accumulation adds the input `i_load` to the resultant soft IP core. The accumulation is cleared when `i_load` is asserted. The output is reset to `i_din_a` * `i_din_b`.
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: • `i_pipeline_ce` • `i_pipeline_rstn`
Output Width	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. **Note** 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 minimum output width can be calculated as: (2 × **Input Width**) + (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 either **Enable Accumulator** is set to **On** or **Pipeline Register Stages** is greater than 0.
`i_pipeline_rstn`	Input	(Optional) Synchronous active-low reset for the pipeline and accumulator registers. Present when either **Enable Accumulator** is set to **On** or **Pipeline Register Stages** is greater than 0.
`i_load`	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**. **Note** This signal is internally pipelined to have the same latency as `i_din_a` and `i_din_b`.
`o_dout`[(Output Width – 1):0]	Output	Result of multiplication and accumulation.

Examples

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



Speedster7t Integer RLB Multiplier

Overview

This page contains the top-level, global properties that govern the structure and base configuration of the Integer RLB Multiplier.

Target Device	AC7t1500ES0
Input Width A	9
Input Width B	9
Enable Input Registers	✓
Enable Accumulator	✓
Pipeline Register Stages	1
Output Width	18

Figure 24: ***9 × 9 Signed Integer RL Multiplier Configuration***

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

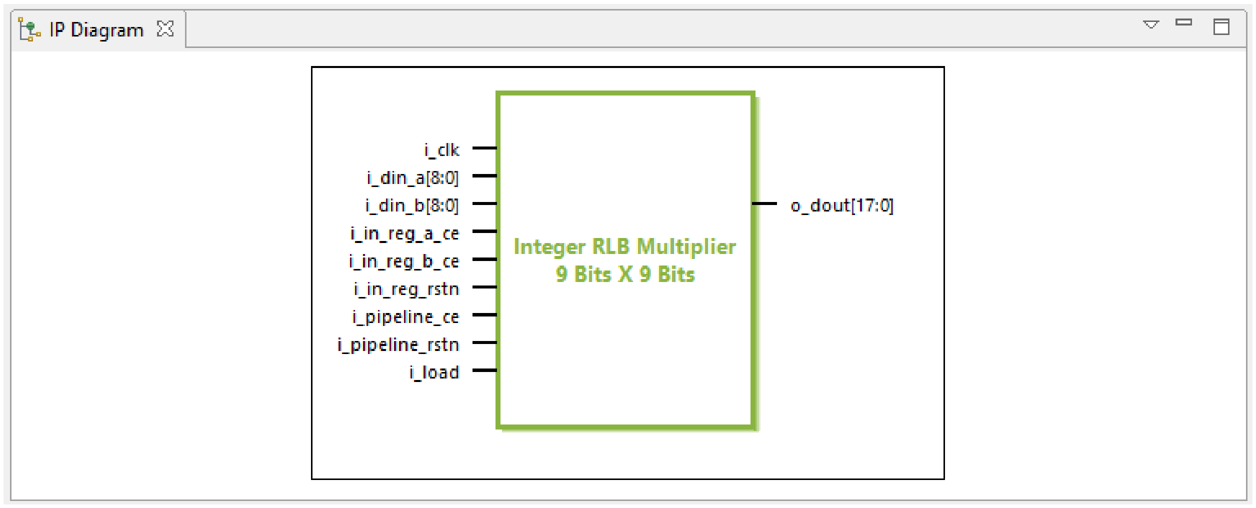


Figure 25: ***9 × 9 Signed Integer RLB Multiplier I/O***

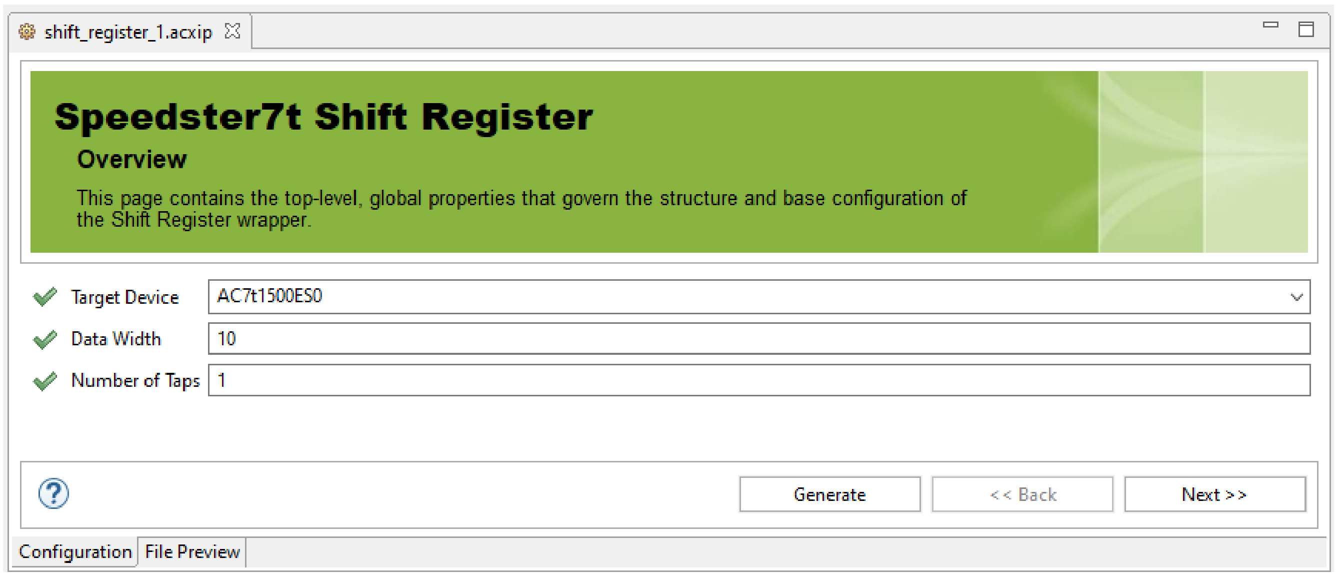
Chapter - 12: Shift Register Soft IP

Description

The Shift Register soft IP core implements a shift register using the fabric flip-flops. The soft IP core 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 soft IP configurator has the following options:


shift_register_1.acxip

Speedster7t Shift Register

Overview

This page contains the top-level, global properties that govern the structure and base configuration of the Shift Register wrapper.

✓ Target Device	AC7t1500ES0
✓ Data Width	10
✓ Number of Taps	1

Figure 26: ***Shift Register Configurator***

Table 21: ***Configuration Options***

Name	Default	Range	Description
Target Device	AC7t1500ES0	All Speedster7t devices	Set to match the target device of the project.
Data Width	10	1 to 65,536	Width of the input and output data bus.
Number of Taps	1	1 to 256	Number of taps in the shift register. All intermediate taps are output from the soft IP.

Note

The number of flip-flops used is calculated by:

Data Width × Number of Taps

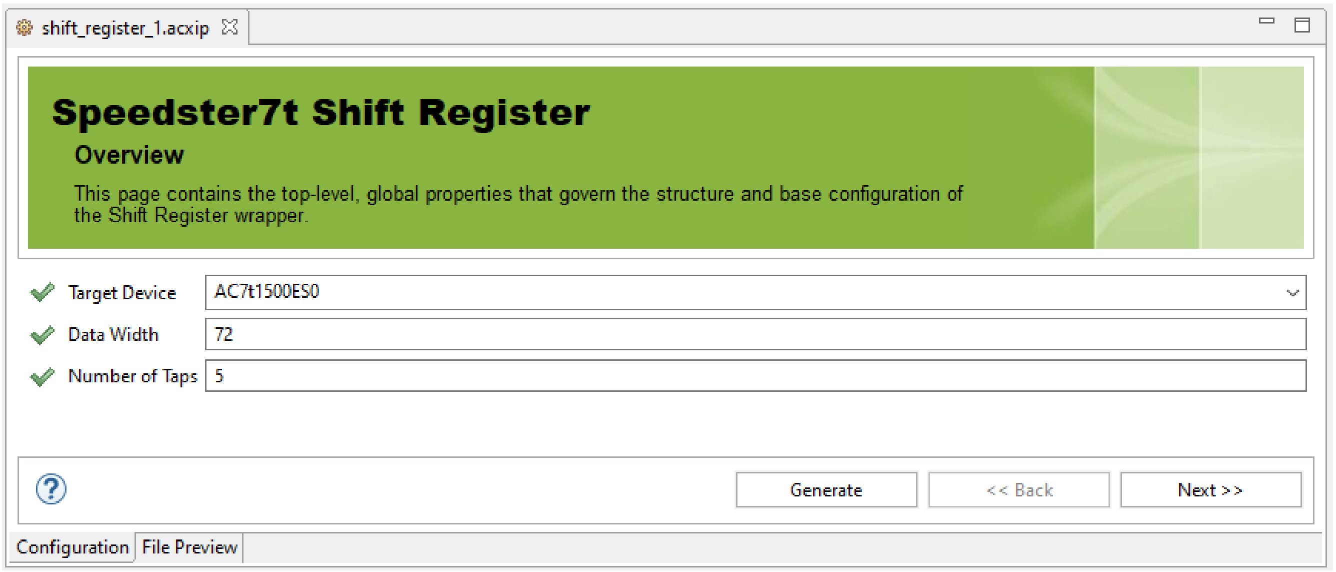
For very wide or deep shift registers, a large number of flip-flops may be used which might result in sub-optimal timing closure. In these circumstances, it is recommended to use a BRAM or LRAM to implement the shift register function. However, it should be noted that a BRAM or LRAM implemented shift register only provides access to the end tap of the shift register, and not the intermediate stages.

Table 22: ***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: `en` = 1'b0: Shift register is stopped. No data is input. Current output is maintained. `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 5 stages deep:


shift_register_1.acxip

Speedster7t Shift Register

Overview

This page contains the top-level, global properties that govern the structure and base configuration of the Shift Register wrapper.

Target Device	AC7t1500ES0
Data Width	72
Number of Taps	5

Figure 27: ***72 Bit, 5 Stage Shift Register Configuration***

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

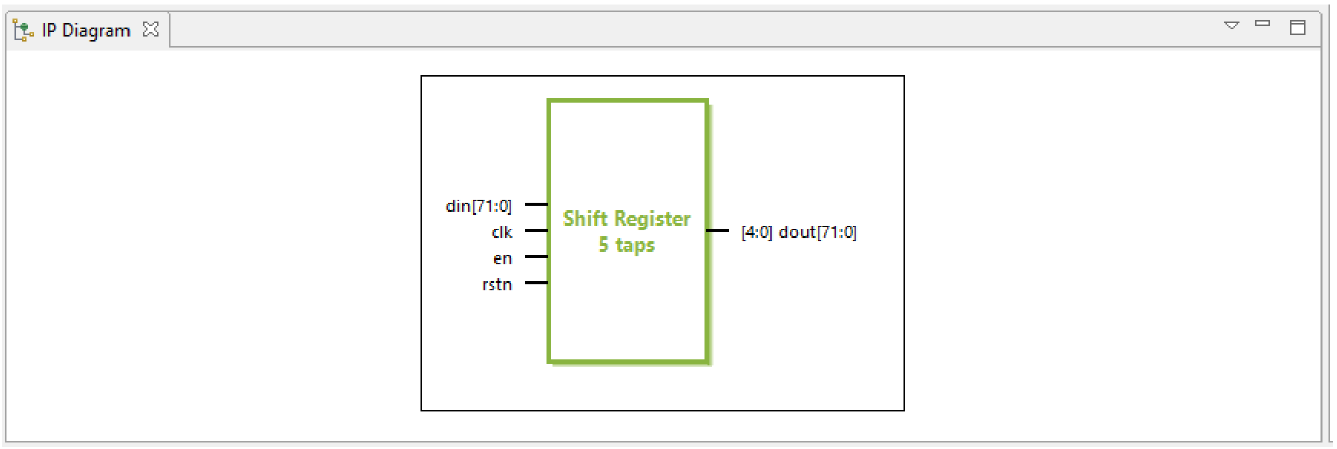


Figure 28: ***72 bit, 5 stage Shift Register I/O***

Chapter - 13: Revision History

Version	Date	Description
1.0	13 Sep 2021	• Initial release.