Interleaver/ De-Interleaver v8.0

LogiCORE IP Product Guide

Vivado Design Suite

PG049 November 18, 2015

XILINX
ALL PROGRAMMABLE™

Table of Contents

Introduction

The interleaver/de-interleaver core is appropriate for any application that requires data to be rearranged in an interleaved fashion, including many popular communications standards such as CDMA2000 and DVB Terrestrial (T), Cable (C), and Satellite (S).

The multiple configuration mode is particularly useful for standards that require swapping between several convolutional interleavers, for example, ITU J.83 Annex B.

Features

- High-speed compact symbol interleaver/de-interleaver with AXI4-Stream interfaces
- Supports standards such as DVB and CDMA2000
- Forney Convolutional and Rectangular Block type architectures available
- Fully synchronous design using a single clock
- Symbol size from 1 to 256 bits
- Internal or external symbol RAM
- Convolutional type features - see Feature Summary
- Rectangular Block type feature- see Feature Summary

LogiCORE™ IP Facts Table	
Core Specifics	
Supported Device Family(1)	UltraScale+™ Families UltraScale™ Architecture Zynq®-7000 7 Series
Supported User Interfaces	AXI4-Stream
Resources	Performance and Resource Utilization web page
Provided with Core	
Design Files	Encrypted RTL
Example Design	Not Provided
Test Bench	VHDL
Constraints File	Not Provided
Simulation Model	Encrypted VHDL
Supported S/W Driver	N/A
Tested Design Flows(2)	
Design Entry	Vivado® Design Suite System Generator for DSP
Simulation	For supported versions of the tools, see the Xilinx Design Tools: Release Notes Guide.
Synthesis	Vivado Synthesis
Support	
Provided by Xilinx at the Xilinx Support web page	

Notes:

1. For a complete listing of supported devices, see the Vivado IP Catalog.
2. For the supported versions of the tools, see the Xilinx Design Tools: Release Notes Guide.

Overview

An interleaver is a device that rearranges the order of a sequence of input symbols. The term *symbol* is used to describe a collection of bits. In some applications, a symbol is a single bit. In others, a symbol is a bus.

The classic use of interleaving is to randomize the location of errors introduced in signal transmission. Interleaving spreads a burst of errors out so that error correction circuits have a better chance of correcting the data.

If a particular interleaver is used at the transmit end of a channel, the inverse of that interleaver must be used at the receive end to recover the original data. The inverse interleaver is referred to as a *de-interleaver*.

Two types of interleaver/de-interleavers can be generated with this core: Forney Convolutional and Rectangular Block. Although they both perform the general interleaving function of rearranging symbols, the way in which the symbols are rearranged and their methods of operation are entirely different.

For very large interleavers, it might be preferable to store the data symbols in external memory. The core provides an option to store data symbols in internal FPGA RAM or in external RAM. This is explained in more detail in External Symbol Memory, page 38.

Feature Summary

- High-speed compact symbol interleaver/ de-interleaver with AXI4 interfaces
- Supports many popular standards, such as DVB and CDMA2000
- Forney Convolutional and Rectangular Block type architectures available
- Fully synchronous design using a single clock
- Symbol size from 1 to 256 bits
- Internal or external symbol RAM
- Convolutional type features:
  - Parameterizable number of branches

  - Parameterizable branch lengths
  - Uniform and non-uniform branch length increments
  - Multiple configurations with on-the-fly swapping
- Rectangular Block type features:
  - Parameterizable, variable, or selectable numbers of rows and columns
  - Parameterizable or variable block size
  - Can change numbers of rows/columns or block size at start of each new block
  - Row and column permutations
  - Multiple permutations for selectable rows or columns
  - Input validity checking
- Use with Xilinx Vivado® Design Suite and Xilinx System Generator for DSP
- Available under terms of the Xilinx Core License Agreement (see Licensing and Ordering Information)

Applications

The interleaver/de-interleaver core is a general purpose core. It is commonly used in telecommunications to mitigate the effect of bursts of errors.

Unsupported Features

As the interleaver/de-interleaver core is general purpose, rather than designed for a particular standard, there are no unsupported features applicable.

Licensing and Ordering Information

If the IP requires a license key, the key must be verified. The Vivado design tools have several license checkpoints for gating licensed IP through the flow. If the license check succeeds, the IP can continue generation. Otherwise, generation halts with error. License checkpoints are enforced by the following tools:

- Vivado design tools: Vivado Synthesis
- Vivado Implementation
- write_bitstream (Tcl command)

IMPORTANT: *IP license level is ignored at checkpoints. The test confirms a valid license exists. It does not check IP license level.*

This Xilinx LogiCORE™ IP module is provided under the terms of the Xilinx Core License Agreement. The module is shipped as part of the Vivado Design Suite. For full access to all core functionalities in simulation and in hardware, you must purchase a license for the core. Contact your local Xilinx sales representative for information about pricing and availability.

For more information, see the Interleaver/De-Interleaver product web page.

Information about other Xilinx LogiCORE IP modules is available at the Xilinx Intellectual Property page. For information on pricing and availability of other Xilinx LogiCORE IP modules and tools, contact your local Xilinx sales representative.

To evaluate this core in hardware, generate an evaluation license, which can be accessed from the Xilinx IP Evaluation page.

After purchasing the core, you will receive instructions for registering and generating a full core license. The full license can be requested and installed from the Xilinx IP Center for use with the Vivado IP catalog. The Vivado IP catalog is bundled with the Xilinx Vivado Design Suite at no additional charge.

Product Specification

Forney Convolutional Operation

Figure 2-1 shows the operation of a Forney Convolutional Interleaver. The core operates as a series of delay line shift registers. Input symbols are presented to the input commutator arm on DIN. Output symbols are extracted from the output commutator arm on DOUT. DIN and DOUT are fields in the AXI4-Stream Data Input Channel and Data Output Channel respectively. Both commutator arms start at branch 0 and advance to the next branch after the next rising clock edge. After the last branch *(B-1)* has been reached, the commutator arms both rotate back to branch 0 and the process is repeated.

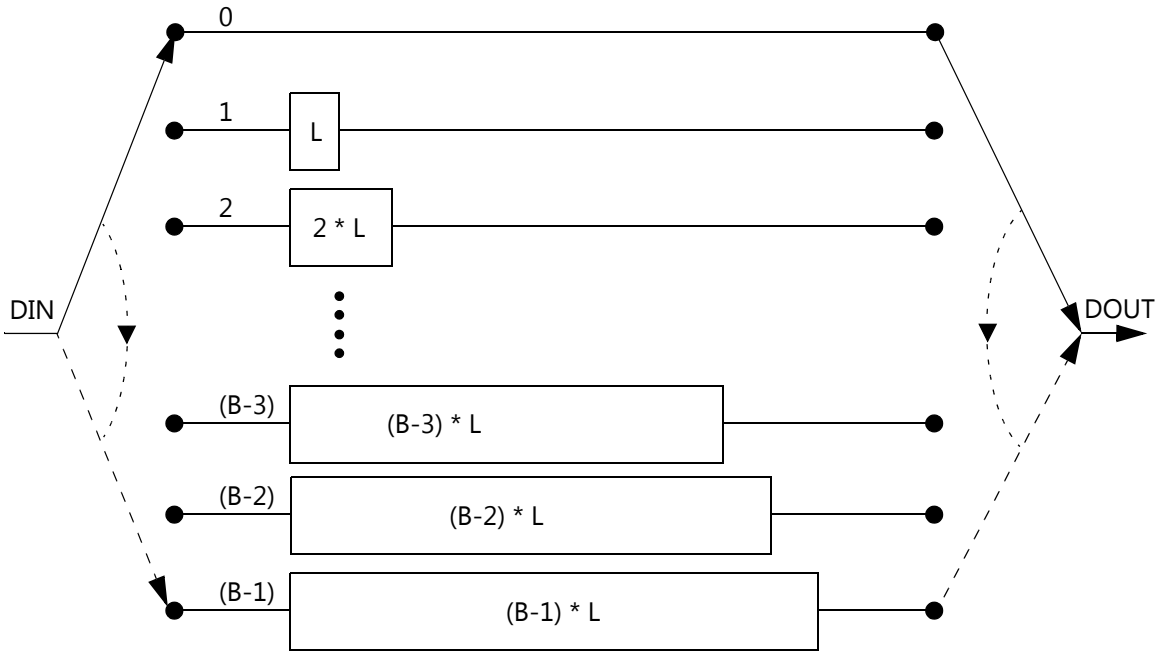


Figure 2-1: **Forney Convolutional Interleaver**

In Figure 2-1, the branches increase in length by a uniform amount, *L*. The core allows interleavers to be specified in this way, or the branch lengths can be passed in using a file, allowing each branch to be any length. Although branch 0 appears to be a zero-delay connection, there is still a delay of several clock cycles between DIN and DOUT because of the fundamental latency of the core. For clarity, this is not shown in Figure 2-1.

The only difference between an interleaver and a de-interleaver is that branch 0 is the longest in the de-interleaver and the branch length is decreased by *L* rather than increased. Branch *(B-1)* has length 0. This is shown in Figure 2-2.

If a file is used to specify the branch lengths, it is arbitrary whether the resulting core is called an interleaver or de-interleaver. All that matters is that one must be the inverse of the other. If a file is used, each branch length is individually controllable. This is shown in Figure 2-3. The file syntax is shown in Figure 4-2, page 57.

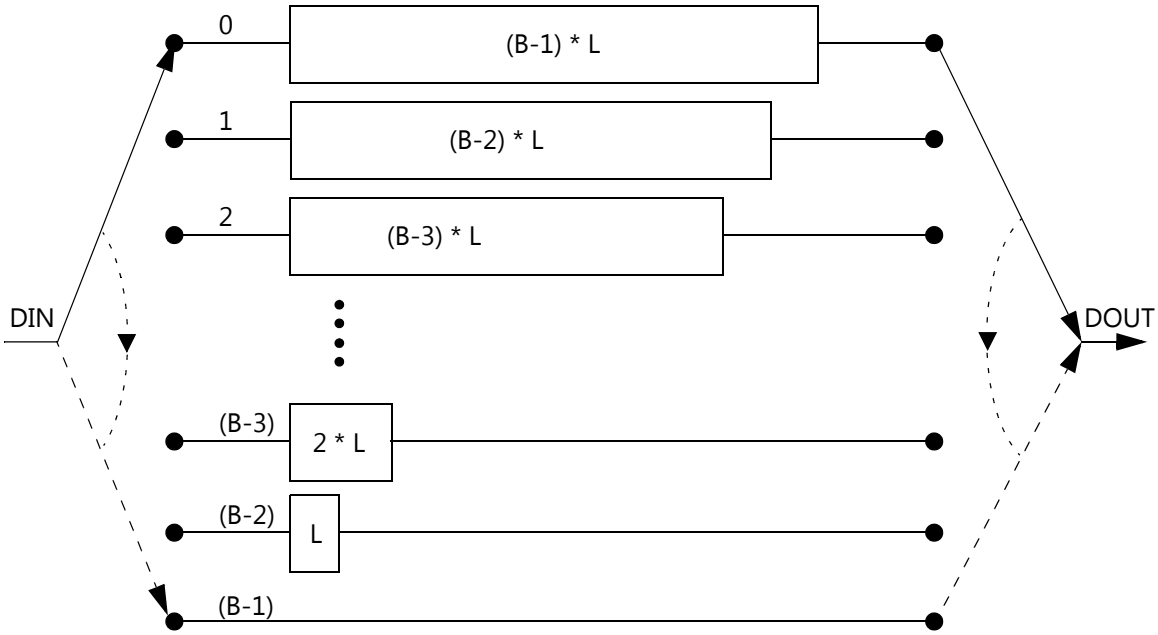


Figure 2-2: **Forney Convolutional De-interleaver**

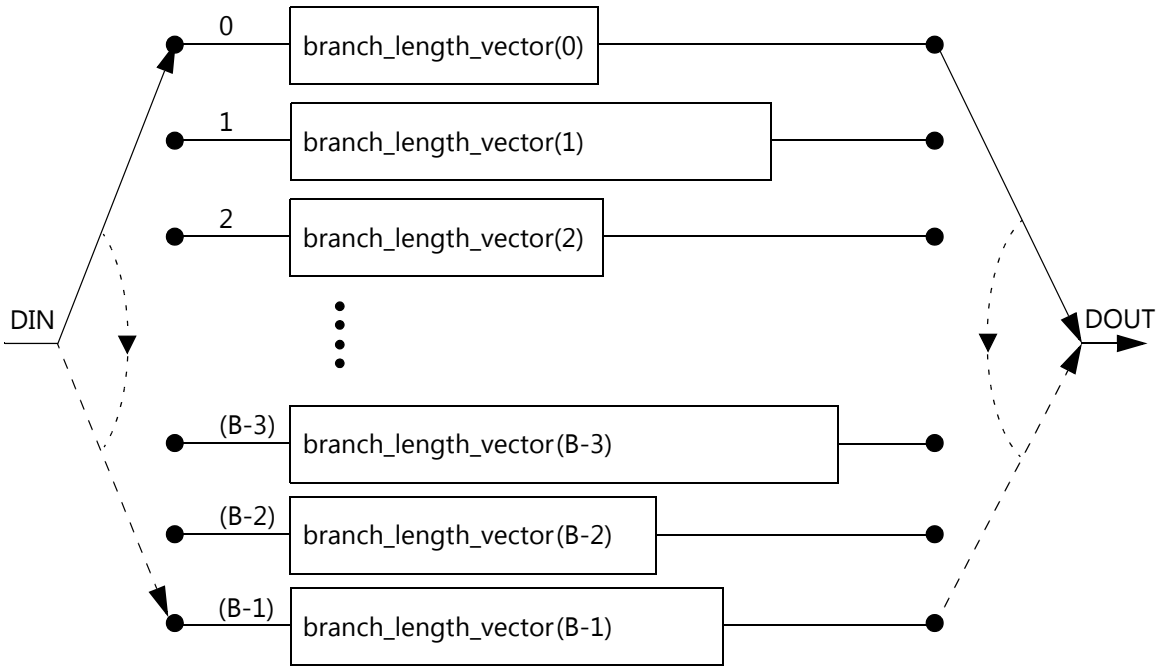


Figure 2-3: **Forney Convolutional Interleaver/De-interleaver with Branch Lengths Set by File**

Configuration Swapping

It is possible for the core to store several pre-defined configurations. Each configuration can have a different number of branches and branch length constant. It is even possible for each configuration to have every individual branch length defined by file.

The configuration can be changed at any time by sending a new CONFIG_SEL value on the AXI4-Stream Control Channel. This value takes effect when the next block starts.

The core assumes all configurations are either for an interleaver or de-interleaver, depending on what was selected in the GUI. It is possible to switch between interleaving and de-interleaving by defining the individual branch lengths for every branch of each configuration.

The details for each configuration are specified in a COE file. See COE Files for Multiple Configurations for details. The timing for a configuration swap is described in Multiple Configuration Timing.

Rectangular Block Operation

The Rectangular Block Interleaver works by writing the input data symbols into a rectangular memory array in a certain order and then reading them out in a different, mixed-up order. The input symbols must be grouped into blocks. Unlike the Convolutional Interleaver, where symbols can be continuously input, the Rectangular Block Interleaver inputs one block of symbols and then outputs that same block with the symbols rearranged.

No new inputs can be accepted while the interleaved symbols from the previous block are being output.

The rectangular memory array is composed of several rows and columns as shown in Table 2-1.

Table 2-1: **Row and Column Indexing Scheme**

Row \ Column	0	1	...	(C-2)	(C-1)
0					
1					
:					
(R-2)					
(R-1)					

The Rectangular Block Interleaver operates as follows:

1. All the input symbols in an entire block are written row-wise, left to right, starting with the top row.
2. Inter-row permutations are performed if required.
3. Inter-column permutations are performed if required.
4. The entire block is read column-wise, top to bottom, starting with the left column.

The de-interleaver operates in the reverse way:

1. All the input symbols in an entire block are written column-wise, top to bottom, starting with the left column.
2. Inter-row permutations are performed if required.
3. Inter-column permutations are performed if required.
4. The entire block is read row-wise, left to right, starting with the top row.

An example of Rectangular Block Interleaver operation is shown in Figure 2-4. This example has 3 rows, 4 columns and a block size of 12. The inter-row permutation pattern is {2, 0, 1}. This means row 0 is permuted to row 2, row 1 is permuted to row 0, and row 2 is permuted to row 1. The meaning of the permute vector differs for the de-interleaver.

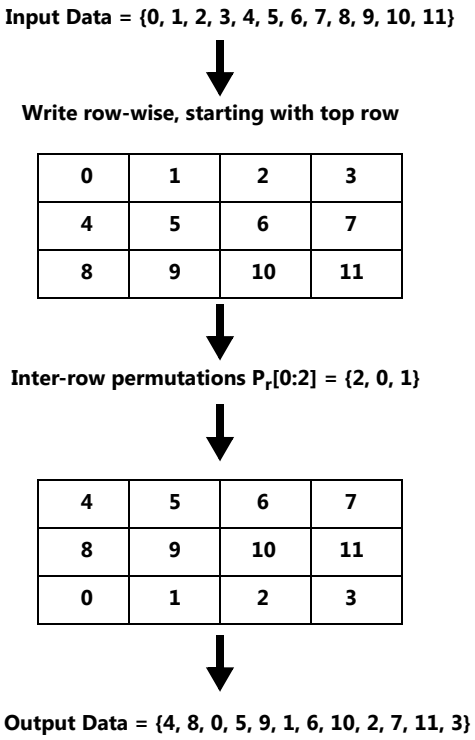


Figure 2-4: **Block Interleaving Example with Row Permutations**

Although this example shows only row permutations, it is possible to do both row and column permutations simultaneously. This is shown in Figure 2-5.

Figure 2-6 shows the output data from the interleaver of Figure 2-5 being de-interleaved. All the parameters are the same, apart from mode, which is set to de-interleaver in this case. Notice how the permute vectors are identical to those for the interleaver, but they are interpreted in a different way. This ensures that output data from the interleaver is correctly restored to the original data by the de-interleaver, as shown in Figure 2-7.

The inter-row permutation pattern is again {2, 0, 1}. However, for the de-interleaver this means row 2 is permuted to row 0, row 0 is permuted to row 1, and row 1 is permuted to row 2.

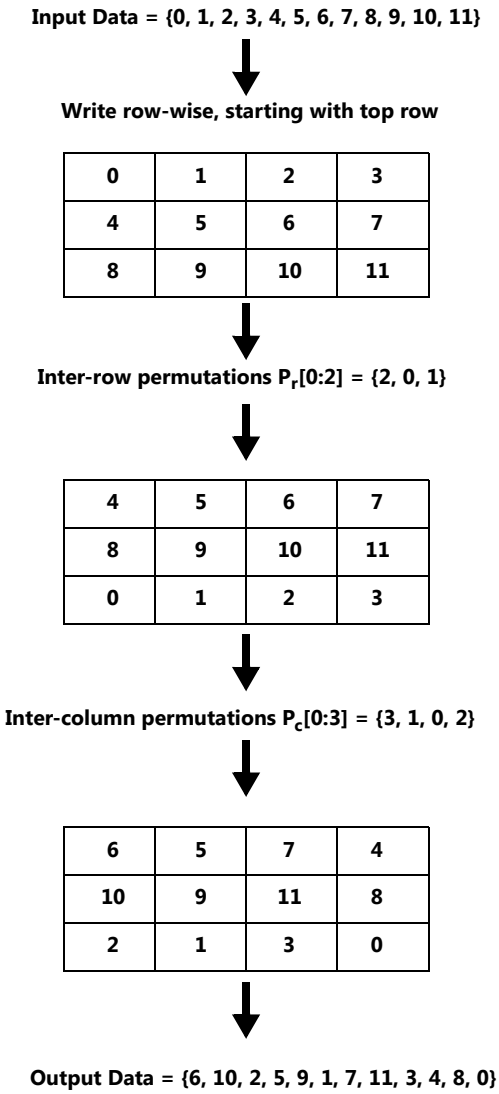


Figure 2-5: **Block Interleaving Example with Row and Column Permutations**

The core can be configured with fixed, variable, or selectable rows and columns. The block size can be fixed, variable or set to always equal R * C. If the block size is variable or less than R * C, row and column permutations are not supported. If the block size is less than R * C, the interleaver is described as *pruned*.

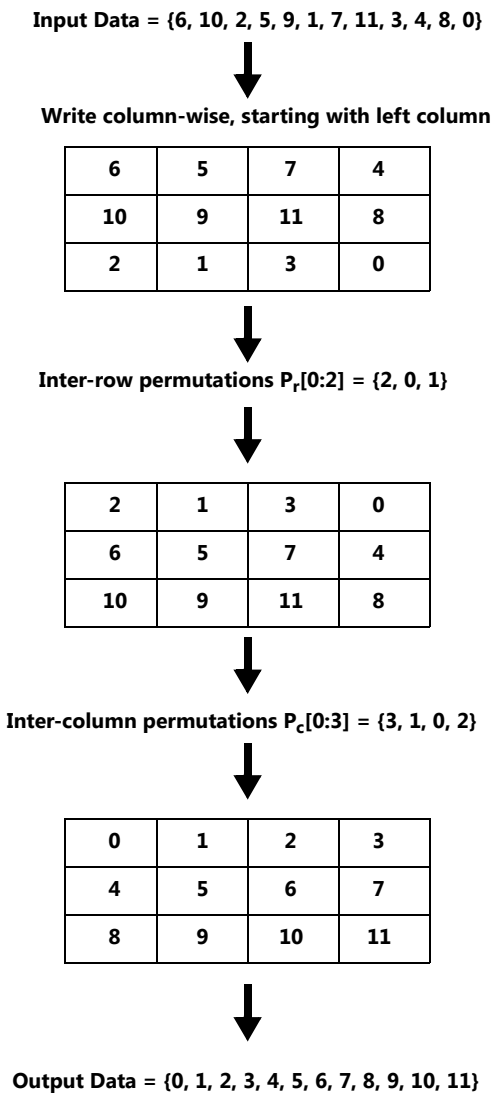


Figure 2-6: **Block De-interleaving Example with Row and Column Permutations**

Row permutations are not supported if the row type is *variable*. Column permutations are not supported if the column type is *variable*.

Selectable rows/columns can be used when the number of possible values for the number of rows or columns is known and is relatively small. In this mode, the number of rows/ columns is still run time variable but chosen from a small set of predetermined values, stored within the core. Permutations are possible when using selectable rows or columns. A different permutation vector can be stored for each row or column select value.

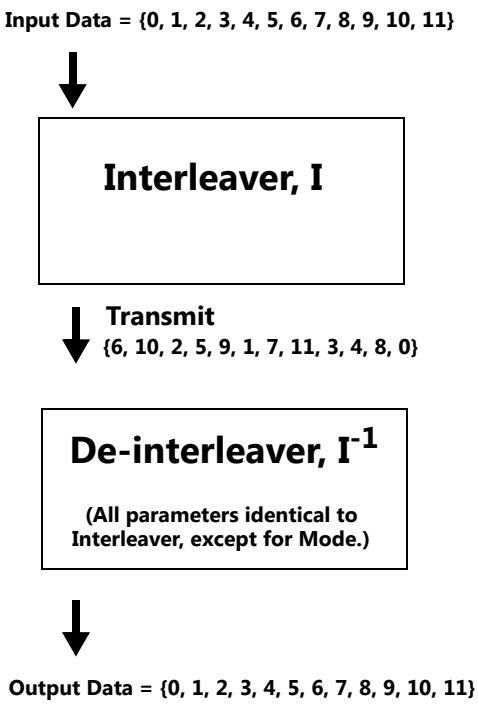


Figure 2-7: **Block Interleaver, De-interleaver Operation**

In general, the most efficient core is one with constant rows, columns, and block size. The number of optional AXI4-Stream fields should be kept to a minimum. For every optional field or extra feature, such as row or column permutations, there might be an adverse impact on area and speed.

More details on the available block size types and choosing the appropriate one are given in Block Size Type.

The core also provides outputs to check the validity of inputs, such as BLOCK_SIZE (see Control Channel for more information). These can sometimes be useful in a receiver to detect if things have gone catastrophically wrong before further error correction is attempted.

Block Interleaver Specifications

Sometimes, a specification requires a Rectangular Block Interleaver, but it is specified in the form of an equation. It is not always immediately obvious that the equation represents a block interleaver.

It might be necessary to evaluate the equation for several values to see if it can be translated into the parameters required by the block interleaver core.

For example, one way of defining a block interleaver is to specify that the symbols are all written to a memory in sequential order (0, 1, ... block_size -1) as in the write phase of Figure 2-4. The symbols are then read back from memory in an order defined by the following equation:

$$\text{Read Address}_i = 2^C(i \bmod R) + \text{BRO}_C(\text{Round Down}(i/R))$$

where i increased from 0 to block_size-1, 2^C is the number of columns and R is the number of rows. $\text{BRO}_C(x)$ is the bit-reversed c-bit value of x, for example, $\text{BRO}_3(1) = 4$.

Examination of the Read Address equation shows that the first part, $2^C(i \bmod R)$, yields the start address of each row. The second part, $\text{BRO}_C(\text{Round Down}(i/R))$, yields how far along the row to go. The BRO part produces column permutations. For example, if c=3, column 4 is permuted to column 1 and column 1 is permuted to column 4.

If c=3 and R=4, then the resultant column permute vector is {0, 4, 2, 6, 1, 5, 3, 7}.

Performance

For details about performance, visit the Performance and Resource Utilization web page.

Timing, Latency and FDO Delay

The precise definitions of latency and FDO Delay differ for the different types of interleaver/de-interleaver. Each is described separately. The following assumptions are made when defining the latency figures:

1. There are no waitstates on any AXI4-Stream channels.
2. `aclken` (if present) is always 1.
3. The core is idle.
4. If the Control Channel is present, a control word is sent before (or coincident with) the first data symbol because the Control Channel blocks the Data Input Channel otherwise.

Forney Convolutional Timing

The latency is the number of clock cycles from a new symbol being sampled on the Data Input Channel, to a new symbol appearing on the Data Output Channel. These are generally not the same symbol, as they have been interleaved or de-interleaved.

The latency is dependent on the pipelining level selected as shown in Table 2-2.

Table 2-2: **Forney Convolutional Latency**

Pipelining	Latency[1][2]
Minimum	4
Medium	5
Maximum	6

Notes:

1. Add one clock cycle to these figures if external symbol RAM is used.
2. Add two clock cycles to these figures if the Data Output Channel has a TREADY (XCO DOUT_HAS_TREADY = TRUE)

The FDO delay is the delay from the first symbol being sampled to that symbol being output on the Data Output Channel. The FDO delay is comprised of two parts: several valid symbols, including the first symbol, and the latency. The GUI reports the FDO delay as the number of new symbols that must be sampled, including the first symbol, plus *latency* clock cycles. This is when FDO is actually asserted. The number of new symbols that must be sampled is dependent on the number of branches and the length of branch 0.

An example, with a latency of four clock cycles, is shown in Figure 2-8. `s_axis_data_tready` is permanently High in this example so is not shown. When the symbol with value 1 is sampled, a new symbol (with value XX) appears on DOUT after four clock cycles. They do not have the same value because the symbols are interleaved. The value 'XX' is used to indicate symbols that are not part of the current frame (for example, values come from an un-initialized memory location or a symbol that was written in the previous frame).

The first symbol that was sampled (value = 1) finally appears on DOUT when FDO goes High.

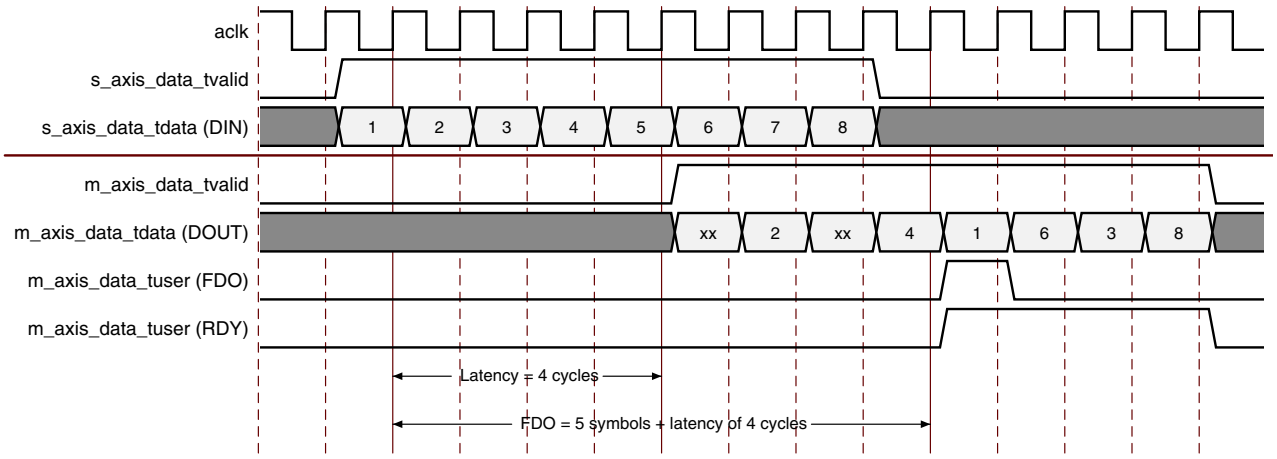


Figure 2-8: **Forney Convolutional Interleaver Interface Timing**

The latency of the core is not affected by waitstates on the Data Input channel (Figure 2-9), but the FDO delay is (Figure 2-10).

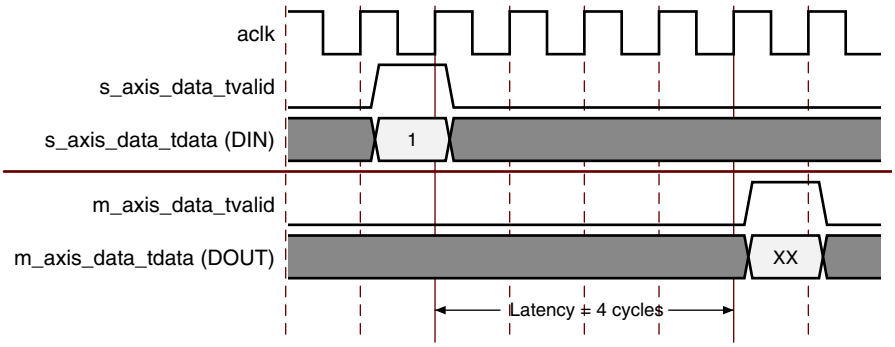


Figure 2-9: **Forney Convolutional Interleaver Interface Timing - Waitstates do not Affect Latency**

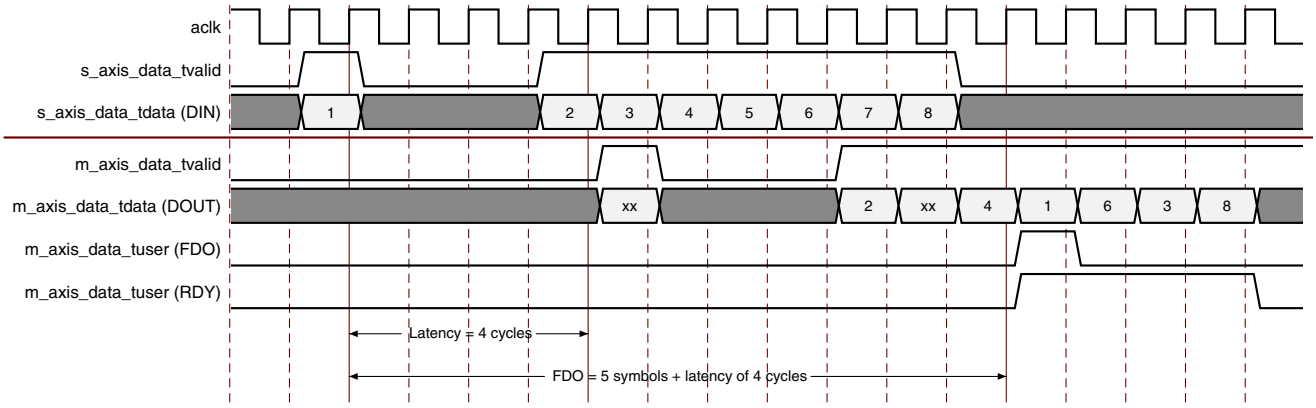


Figure 2-10: **Forney Convolutional Interleaver Interface Timing - Waitstates do Affect Delay**

Multiple Configuration Timing

Figure 2-11 shows the timing when the configuration is swapped using CONFIG_SEL. The example uses maximum pipelining and has no TREADY on the Data Output Channel. Therefore, its latency is 6 cycles. Configuration 0 has 2 branches (of length 0 and 1) and configuration 1 has 3 branches, of length 0, 2, and 4. FDO delay in both cases is 1 symbol, which is the first symbol itself. `s_axis_data_tready` and `s_axis_data_tvalid` are both 1 for the duration of this example so are not shown.

In this example, the core switches from configuration 0 to configuration 1. The new configuration value is sent on the Control Channel when DIN = 15 but the configuration is not actually changed until the DIN symbol with value 17, as this is the first symbol of a new frame.

Note: TLAST is an indication to the core that the block should end when the last branch is reached. It does not specify that the block ends immediately (unless the commutator is already on the last branch). TLAST is stored by the core and used at the next opportunity, even if it was incorrectly sent.

In this example `s_axis_tlast` is asserted one cycle early and `event_tlast_unexpected` is asserted as a result. The last symbol here has a value of 16 even though the symbol with value 15 was marked by the user as being the last one. Care must be taken to ensure that the symbol data remains in synchronisation with the branch commutator. If the user had expected symbol 15 to be the last one, the incoming data stream would no longer be synchronised to the core.

When the Symbol Interleaver/De-interleaver detects the new configuration it continues to output the configuration 0 data for Latency clock cycles, as defined previously. After this number of clock cycles, DOUT is unknown until the first configuration 1 symbol appears on DOUT. Symbols for the new frame appear after the FDO Delay plus latency as normal. Because the first branch in configuration 1 is 0, FDO for configuration 1 appears directly after the last symbol from configuration 0, and RDY remains asserted. If the first branch for configuration 1 had been greater than 0, then RDY would have deasserted after the last symbol from configuration 0 and remained deasserted until FDO asserted.

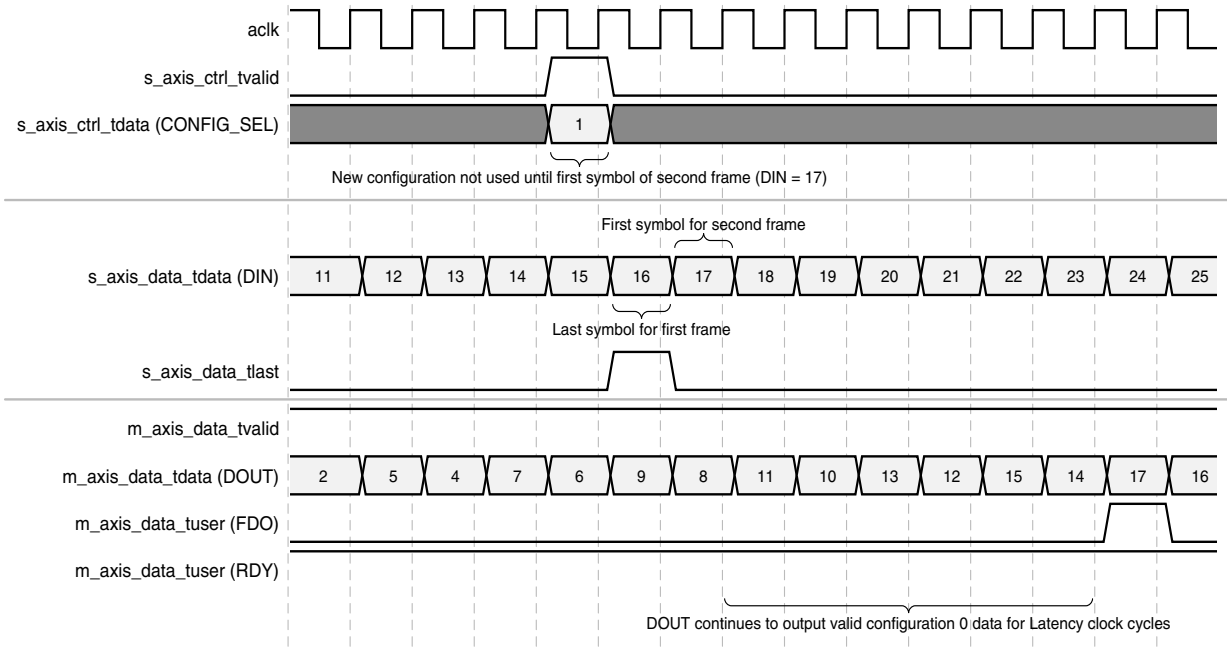


Figure 2-11: **Configuration Swapping Timing**

Rectangular Block Timing

The latency is the number of clock cycles from the last symbol in a block being sampled on the Data Input Channel to the first symbol from that block appearing on the Data Output Channel.

The latency is dependent on the block size type, row and column types and whether there are any permutations. All the possible values are shown in Table 2-3.

Table 2-3: **Rectangular Block Latency**

Block Size Type	Column Type	Row Type	Row or Column Permutations	Latency (Clock Cycles)(1)
Constant	-	-	No	5
Constant	-	-	Yes	7
Row * Column	-	-	No	5
Row * Column	-	-	Yes	7
Variable	Constant	Constant	N/A	7
Variable	Not Constant	-	N/A	10
Variable	-	Not Constant	N/A	10

Notes:

1. Add two clock cycles if the Data Output Channel has a TREADY (XCO DOUT_HAS_TREADY = TRUE)

The FDO delay, or BLOCK_START delay, is the delay from the first symbol of a block being sampled to the first symbol of the same block being output on the Data Output Channel. This is a different symbol to the first one sampled due to the interleaving process. The BLOCK_START delay is composed of two parts: several valid symbol pulses, including the first symbol, and the latency. The GUI reports the BLOCK_START delay as the number of new symbols that must be sampled, including the first symbol, plus several clock cycles. This is when BLOCK_START field in the Data Output Channel is actually asserted, assuming `aclken` is High all the time. The number of new symbols that must be sampled is equal to the block size.

An example is shown in Figure 2-12. In this example, the block size is variable and the number of rows and columns is constant. There are two rows and four columns. There are no permutations. Using Table 2-3, the latency is seven clock cycles. The block size is resampled at the start of each new block.

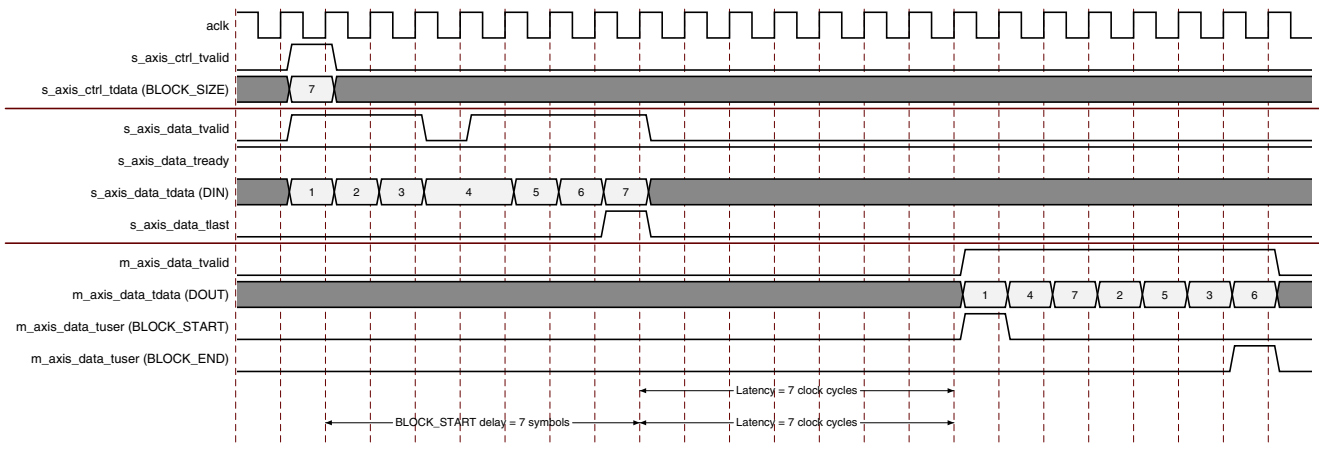


Figure 2-12: **Rectangular Block Interface Timing**

The block begins when s_axis_data_tvalid = 1 and s_axis_data_tready = 1 and a control word has been sent to the core. In this example the first symbol has the value 1. The block size is sampled as 7 at this time. If the core had ROW, COL, ROW_SEL or COL_SEL inputs, they would also be sampled at this time. The seven input symbols are sampled. The symbol with value 4 is initially ignored because s_axis_data_tvalid is 0. The symbol with value 7 is the seventh and final symbol in the block and is designated as such by the assertion of s_axis_data_tlast.

The BLOCK_START Delay is reported as seven symbols plus the seven clock cycle latency. The latency is independent of the block size; the fact that they are both 7 in this example is coincidence.

Note that s_axis_data_tvalid being Low does affect the count of the number of symbols, but it does not affect the latency. The output symbols are always output consecutively on the Data Output Channel, even if there were gaps between some of the input symbols. In the example, there was a one clock cycle gap in the input symbols between '3' and '4' because s_axis_data_tvalid was deasserted, but this has no effect on the rate at which data is output on the Data Output Channel.

It is possible to overlap the processing of two blocks. Figure 2-13 shows the loading of second block starting immediately after the first block is loaded. The first two symbols (values 11 and 12) are immediately taken by the core as there is space in the Data Input Channel for them. However, the next symbol (value 13) is stalled by the core by deasserting s_axis_data_tready. This is because the core is internally still processing the previous frame and cannot use these symbols yet. The new control word (BLOCK_SIZE = 9) has also been accepted by the core, but has not yet been used.

After a certain amount of time the core starts to accept new symbols on the Data Input Channel even though it is still unloading symbols for the previous frame. The time between the last symbol of frame N being sampled and the core starting to process the first symbol of frame N+1 is always equal to the block length +2 of frame N. When it has started processing symbols internally, it accepts new symbols on the Data Input Channel by asserting s_axis_data_tready.

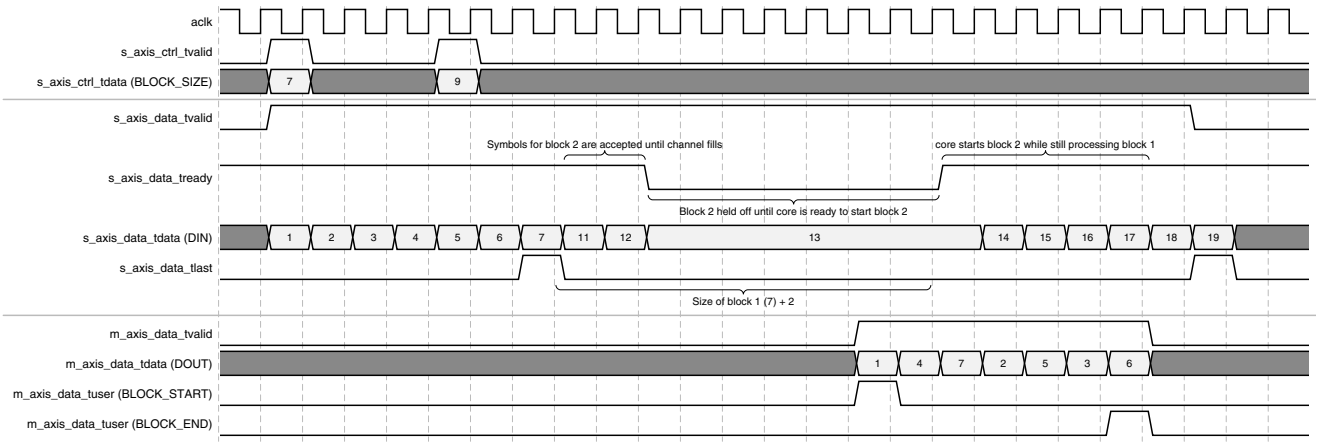


Figure 2-13: **Loading Rectangular Blocks Back to Back**

If any of the fields in the Control Channel contain illegal values, the core deasserts the appropriate `event_*_valid` signal after a delay. Using Table 3-10, you can see that the delay to the `event_block_size_valid` output changing is 7- 2 = 5 clock cycles. `event_block_size_valid` changes five clock cycles after the block starts. The block starting does not necessarily correspond to the first symbol being consumed on the Data Input Channel (see Figure 2-13). Figure 2-15 shows that the VALID Delay is not affected by waitstates. If an invalid block configuration is sampled, the core aborts the processing of the block and treats the next symbol as being the first symbol in a new frame.

RECOMMENDED: Xilinx recommends resetting the core if any of the `event_*_valid` outputs go Low.

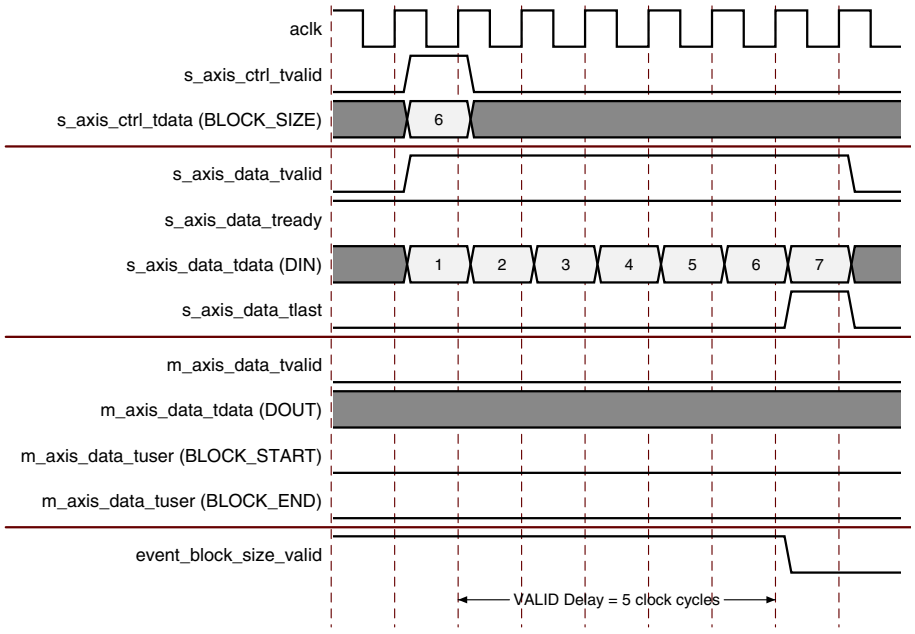


Figure 2-14: **Response to Illegal Rectangular Blocks**

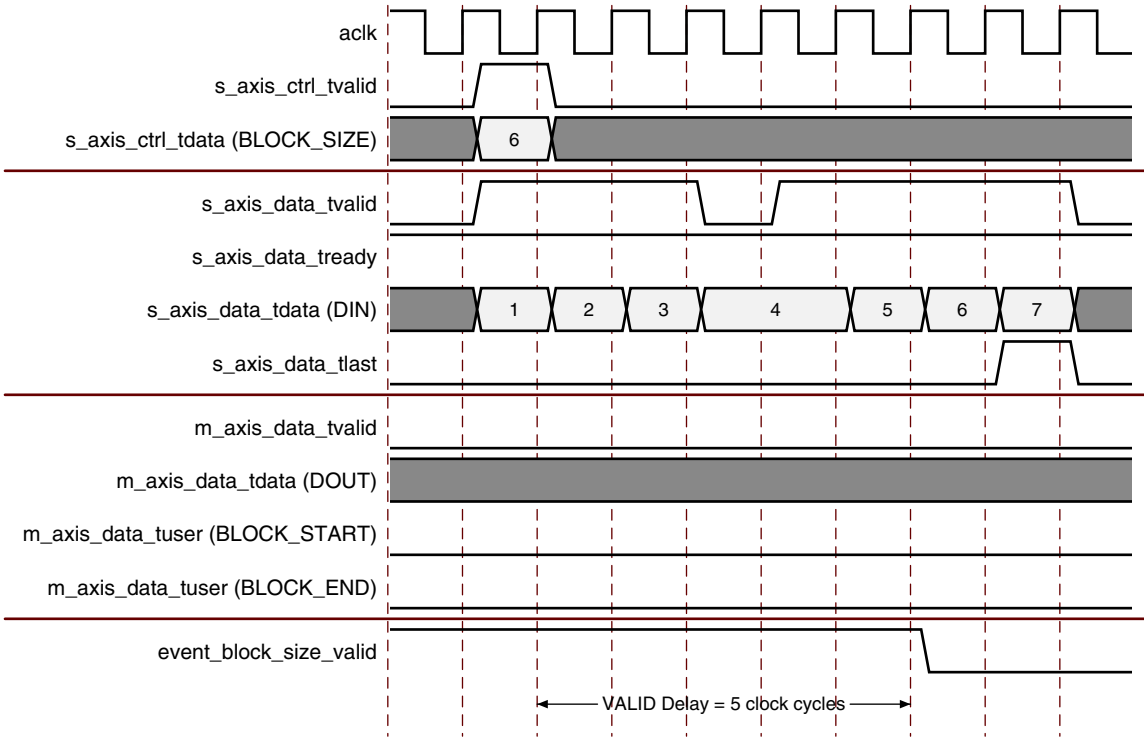


Figure 2-15: **Response to Illegal Rectangular Blocks. VALID Delay is not Affected by Waitstates**

Throughput

The Interleaver/De-interleaver core supports full throughput when in Forney mode. When in Rectangular mode, throughput is 50% because one memory is used for both block write and block read.

Power

No power measurements have been performed on this core.

Resource Utilization

For details about resource utilization, visit the Performance and Resource Utilization web page.

Port Descriptions

Some of the pins are optional. These should be selected only if they are genuinely required, as their inclusion might result in an increase in the core size.

Representative symbols for the Forney Convolutional type and Rectangular Block type are shown in Figure 2-16 and Figure 2-17, respectively.

Schematic Symbol for Forney Convolutional Type

Figure 2-16 illustrates the Forney Convolutional type schematic symbol.

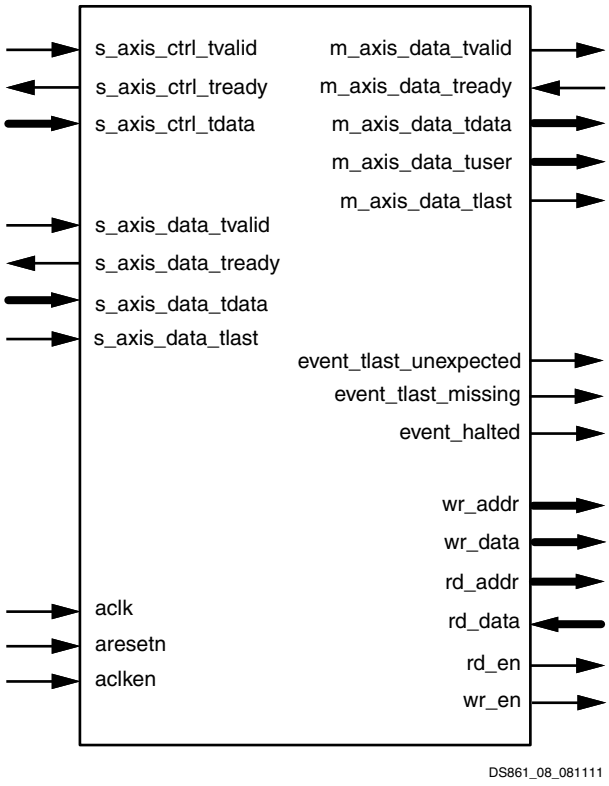


Figure 2-16: **Forney Convolutional Schematic Symbol**

Schematic Symbol for Rectangular Block Type

Figure 2-17 illustrates the Rectangular Block type schematic symbol.

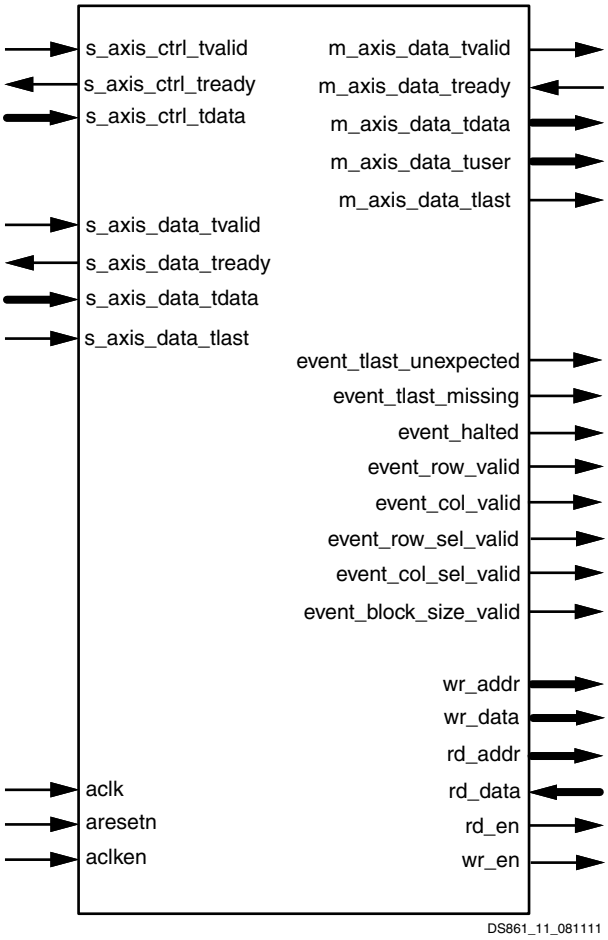


Figure 2-17: **Rectangular Block Schematic Symbol**

Table 2-4 summarizes the signal functions. They are described in more detail in AXI4-Stream Interface Considerations. Timing examples are shown in Timing, Latency and FDO Delay.

Table 2-4: **Core Signal Pinout**

Signal	I/O	Description
ACLK	Input	Rising edge clock.
ALCKEN	Input	Active-High clock enable (optional). If aresetn and aclken are asserted at the same time, aresetn takes precedence.
ARESETN	Input	Active-Low synchronous clear (optional). A minimum aresetn active pulse of two cycles is required. If aresetn and aclken are asserted at the same time, aresetn takes precedence.

Table 2-4: **Core Signal Pinout *(Cont'd)***

Signal	I/O	Description
S_AXIS_DATA_TVALID	Input	TVALID for the Data Input Channel. Used by the external master to signal that it is able to provide data.
S_AXIS_DATA_TREADY	Output	TREADY for the Data Input Channel. Used by the Symbol Interleaver/De-interleaver to signal that it is ready to accept data.
S_AXIS_DATA_TDATA	Input	TDATA for the Data Input Channel. Carries the unprocessed symbol data.
S_AXIS_DATA_TLAST	Input	TLAST for the Data Input Channel. In Forney mode, signals that the block ends *when the commutator reaches the last branch*. In rectangular mode, signals the last symbol in a block.
S_AXIS_CTRL_TVALID	Input	TVALID for the Control Channel. Asserted by the external master to signal that it is able to provide data. Only present when the core is configured to have control options.
S_AXIS_CTRL_TREADY	Output	TREADY for the Control Channel. Asserted by the Symbol Interleaver/De-interleaver to signal that it is ready to accept control data. Only present when the core is configured to have control options.
S_AXIS_CTRL_TDATA	Input	TDATA for the Control Channel. Can contain the following field in Forney mode: ◦ CONFIG_SEL Can contain the following fields in Rectangular mode: ◦ ROW ◦ ROW_SEL ◦ COL ◦ COL_SEL ◦ BLOCK_SIZE Only present when the core is configured to have control options.
M_AXIS_DATA_TVALID	Output	TVALID for the Data Output Channel. Asserted by the Symbol Interleaver/De-interleaver to signal that it is able to provide symbol data.
M_AXIS_DATA_TREADY	Input	TREADY for the Data Output Channel. Asserted by the external slave to signal that it is ready to accept data. Only present when the XCO parameter HAS_DOUT_TREADY is TRUE.
M_AXIS_DATA_TDATA	Output	TDATA for the Data Output Channel. Carries the processed symbol data

Table 2-4: **Core Signal Pinout *(Cont'd)***

Signal	I/O	Description
M_AXIS_DATA_TUSER	Output	TUSER for the Data Output Channel. Optional signal Can contain the following fields in Forney mode: ◦ FDO ◦ RDY Can contain the following fields in Rectangular mode: ◦ BLOCK_START ◦ BLOCK_END
M_AXIS_DATA_TLAST	Output	TLAST for the Data Output Channel. In Forney mode, asserted every time a symbol is produced from the last branch. In Rectangular mode, asserted when the last symbol is produced for a block
RD_ADDR	Output	Read address for external symbol RAM. Only present if the core is configured to have External Symbol Memory.
RD_DATA	Input	Read data value from external symbol RAM. Only present if the core is configured to have External Symbol Memory.
RD_EN	Output	Read enable for external symbol RAM. High when reading data from external symbol RAM. Only present if the core is configured to have External Symbol Memory.
WR_ADDR	Output	Write address for external symbol RAM. Only present if the core is configured to have External Symbol Memory.
WR_DATA	Output	Write data value for external symbol RAM. Only present if the core is configured to have External Symbol Memory.
WR_EN	Output	Write enable for external symbol RAM. High when writing data to external symbol RAM. Only present if the core is configured to have External Symbol Memory.
EVENT_ROW_VALID	Output	1 when the value in the Control Channel ROW field is valid. Becomes 0 if the value in invalid. See event_row_valid for more information. Only present if the core is configured to be in Rectangular mode.
EVENT_COL_VALID	Output	1 when the value in the Control Channel COL field is valid. Becomes 0 if the value in invalid. See event_col_valid for more information. Only present if the core is configured to be in Rectangular mode.

Table 2-4: **Core Signal Pinout *(Cont'd)***

Signal	I/O	Description
EVENT_ROW_SEL_VALID	Output	1 when the value in the Control Channel ROW_SEL field is valid. Becomes 0 if the value in invalid. See event_row_sel_valid for more information. Only present if the core is configured to be in Rectangular mode.
EVENT_COL_SEL_VALID	Output	1 when the value in the Control Channel COL_SEL field is valid. Becomes 0 if the value in invalid. See event_col_sel_valid for more information. Only present if the core is configured to be in Rectangular mode.
EVENT_BLOCK_SIZE_VALID	Output	1 when the value in the Control Channel BLOCK_SIZE field is valid. Becomes 0 if the value in invalid. See event_block_size_valid for more information. Only present if the core is configured to be in Rectangular mode.
EVENT_TLAST_UNEXPECTED	Output	Asserted on every clock cycle where s_axis_data_tlast is unexpectedly seen asserted; that is, asserted on a symbol that is not the last symbol. The meaning of "Last Symbol" varies between Forney and Rectangular mode: ◦ Forney - a symbol corresponding to the last branch ◦ Rectangular - the final symbol loaded for a block
EVENT_TLAST_MISSING	Output	Asserted on the last symbol of an incoming block if s_axis_data_tlast is not asserted with that symbol. This signal is only present in Rectangular mode.
EVENT_HALTED	Output	Asserted when the Symbol Interleaver/De-interleaver tries to write data to the Data Output Channel and it is unable to do so. Only present when the XCO HAS_DOUT_TREADY is TRUE.

Designing with the Core

This chapter includes guidelines and additional information to facilitate designing with the core.

Clocking

The Interleaver/De-Interleaver uses a single clock, called `aclk`. All input and output interfaces and internal state are subject to this single clock.

Resets

The Interleaver/De-Interleaver core uses a single, optional reset input called `aresetn`.

IMPORTANT: This signal is active-Low and must be active or inactive for a minimum of two clock cycles to ensure correct operation.

`aresetn` is a global synchronous reset which resets all control states in the core; all data in transit through the core is lost when `aresetn` is asserted.

Protocol Description

AXI4-Stream Interface Considerations

The conversion to AXI4-Stream interfaces brings standardization and enhances interoperability of Xilinx IP LogiCORE™ solutions. Other than general control signals such as `aclk`, `aclken` and `aresetn`, external memory interface and event signals, all inputs and outputs to the Symbol Interleaver/De-interleaver are conveyed using AXI4-Stream channels. A channel always consists of TVALID and TDATA plus additional ports (such as TREADY, TUSER and TLAST) when required and optional fields. Together, TVALID and TREADY perform a handshake to transfer a message, where the payload is TDATA, TUSER and TLAST.

For further details on AXI4-Stream interfaces, see the *Xilinx AXI Reference Guide* (UG761) [Ref 1] and the *AMBA® AXI4-Stream Protocol Specification* [Ref 2].

Basic Handshake

Figure 3-1 shows the transfer of data in an AXI4-Stream channel. TVALID is driven by the source (master) side of the channel and TREADY is driven by the receiver (slave). TVALID indicates that the values in the payload fields (TDATA, TUSER and TLAST) are valid. TREADY indicates that the slave is ready to receive data. When both TVALID and TREADY are TRUE in a cycle, a transfer occurs. The master and slave set TVALID and TREADY respectively for the next transfer appropriately.

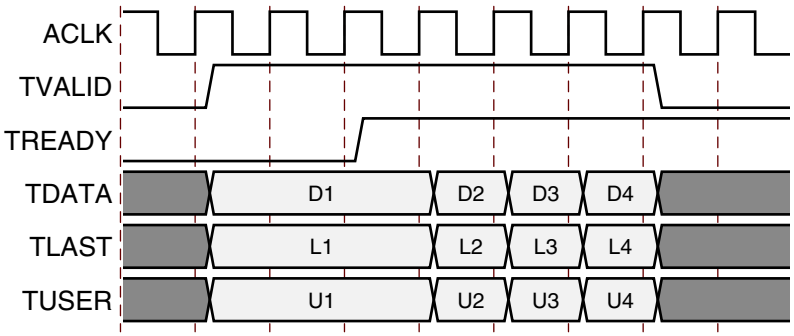


Figure 3-1: **Data Transfer in an AXI4-Stream Channel**

AXI4 Channel Rules

All of the AXI4 channels follow the same rules:

- All TDATA and TUSER fields are packed in little endian format. That is, bit 0 of a sub-field is aligned to the same side as bit 0 of TDATA or TUSER
- All TDATA and TUSER vectors are multiples of 8 bits. After all fields in a TDATA or TUSER vector have been concatenated, the overall vector is padded to bring it up to an 8 bit boundary.

Data Input Channel

The Data Input Channel carries symbol data to be interleaved or de-interleaved. The symbol data is carried in TDATA. The channel contains a two element buffer to provide some system elasticity.

The Data Input Channel is blocked by the Control Channel (if present). When a symbol in the Data In Channel represents the first symbol for a block, it is not processed until a new configuration is seen in the Control Channel. The new configuration does not have to be different from the previous configuration.

Pinout

Table 3-1: **Data Input Channel Pinout**

Port Name	Port Width	Direction	Description
s_axis_data_tdata	Variable. See the Vivado® Customize IP dialog box when configuring the Symbol Interleaver/ De-interleaver.	In	Carries the symbol data
s_axis_data_tvalid	1	In	Asserted by the upstream master to signal that it is able to provide data
s_axis_data_tlast	1	In	Asserted by the upstream master on the symbol corresponding to the last symbol in a block. In Forney mode, the last symbol of a block is deemed to be the one that enters the last branch when s_axis_data_tlast is, or has been, asserted. For example, if s_axis_data_tlast is asserted with the symbol that enters branch 3 in a 10 branch system (numbered 0 to 9), symbols for branches 4 to 9 are deemed to be part of the current block, and the block is not finished until the symbol for branch 9 is consumed. If a new control word is waiting in the Control Channel, it is not consumed until the symbol for branch 0 after s_axis_data_tlast = 1 is consumed.
s_axis_data_tready	1	Out	Used by the Symbol Interleaver/De-interleaver to signal that it is ready to accept data.

TDATA Fields

The DATA field is packed into the `s_axis_data_tdata` vector as shown in Table 3-2:

Table 3-2: **Data Input Channel TDATA Fields**

Field Name	Width	Padded	Description
DIN	1 to 256	Yes	The symbol data to be interleaved or de-interleaved.

The DIN field should be extended to the next 8-bit boundary if it does not already finish on an 8-bit boundary. The Symbol Interleaver/De-interleaver core ignores the value of the padding bits, so they can be driven to any value.

TDATA Format

The Data Input Channel TDATA vector (`s_axis_data_tdata`) has one field (DIN) which is packed as shown in Figure 3-2.

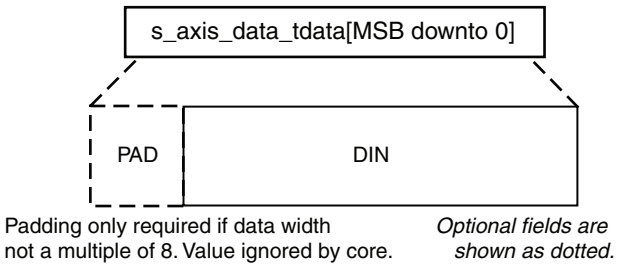


Figure 3-2: **Data Input Channel TDATA (s_axis_data_tdata) Format**

Note: In Forney mode, every symbol sampled on this channel causes a new symbol to be output on the Data Output Channel.

Data Output Channel

The Data Output Channel carries symbol data that has been interleaved or de-interleaved, and a selection of information flags. The data is carried in TDATA and the flags in TUSER. As all of these flags are individually optional, the TUSER vector might not always be present.

In Forney mode, one new symbol is output on the Data Output Channel for each input symbol sampled on the Data Input Channel. The number of clock cycles between an input symbol being sampled and a new output appearing is termed the latency of the core. This is not the number of cycles from a particular symbol being sampled on the Data Input Channel and the same symbol finally appearing on the Data Output Channel. For the first symbol, this is termed the FDO delay and is dependent on the interleaver branch lengths. This delay is different for each input symbol because of the interleaving process. The latency is always the same, regardless of branch lengths. Latency and the FDO delay are described in Timing, Latency and FDO Delay, page 16.

In Rectangular mode, the core does not start to output the symbols from the block in an interleaved/de-interleaved fashion until all the symbols from the block have been sampled. The number of clock cycles between the last input symbol in a block being sampled and the first symbol from that block appearing on DOUT is termed the latency of the core. This definition differs from the one used for the Convolutional interleaver. The number of cycles from the first symbol being sampled on the Data Input Channel to the first symbol of the same block finally appearing on the Data Output Channel is termed the FDO delay or BLOCK_START delay. This delay is dependent on the interleaver block size. The delay from an input symbol being sampled on the Data Input Channel to that same symbol finally appearing on the Data Output Channel is different for each input symbol because of the interleaving process.

Latency and the BLOCK_START delay are described in Timing, Latency and FDO Delay, page 16.

If TREADY is disabled for this channel then it is not buffered. Failure to consume symbols when TVALID is asserted means that the symbol is lost. If TREADY is enabled for this channel, the channel is buffered to store a small number of symbols.

Pinout

Table 3-3: **Data Output Channel Pinout**

Port Name	Port Width	Direction	Description
m_axis_data_tdata	Variable. See the Vivado Customize IP dialog box when configuring the Symbol Interleaver/ De-interleaver.	Out	Carries the symbol data.
m_axis_data_tuser	Variable. See the Vivado Customize IP dialog box when configuring the Symbol Interleaver/ De-interleaver.	Out	Carries additional per-symbol status. As all status fields are optional, TUSER might not always be present.
m_axis_data_tvalid	1	Out	Asserted by the Symbol Interleaver/ De-interleaver to signal that it is able to provide symbol data.
m_axis_data_tlast	1	Out	In Rectangular mode, m_axis_data_tlast is asserted when the last symbol for a processed block is transmitted. In Forney mode, m_axis_data_tlast is asserted when a symbol is transmitted from the last branch. In this mode, m_axis_data_tlast bears no similarity to s_axis_data_tlast. s_axis_data_tlast is asserted once per block but m_axis_data_tlast is asserted once every sweep of the commutator
m_axis_data_tready	1	In	Asserted by the external slave to signal that it is ready to accept data. Only available when the core is configured to have a TREADY on the Data Output Channel.

TDATA Fields

The DATA field is packed into the `m_axis_data_tdata` vector as shown in Table 3-4:

Table 3-4: **Data Output Channel TDATA Fields**

Field Name	Width	Padded	Description
DOUT	1 to 256	Yes - sign extended	The processed symbol data. This field always has the same width as DIN.

The DATA field is sign extended to the next 8 bit boundary if it does not already finish on an 8 bit boundary.

TDATA Format

The Data Output Channel TDATA vector (`m_axis_data_tdata`) has one field (DOUT) which is packed as shown in Figure 3-3:

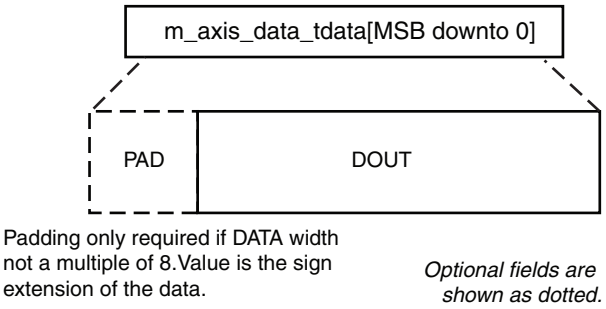


Figure 3-3: **Data Output Channel TDATA (m_axis_data_tdata) Format**

TUSER Fields

The Data Output Channel carries the fields in Table 3-5 in its TUSER vector:

Table 3-5: **Data Output Channel TUSER Fields**

Field Name	Width	Padded	Description
FDO	1	No	Forney mode only. Asserted when the first symbol in a block appears on the Data Output Channel. The FDO field is useful for synchronizing circuits downstream of the interleaver. It can be used to tell those circuits when to start a new block if the data is arranged in blocks.
RDY	1	No	Forney mode only. The RDY (Ready) field signals valid data on DOUT. It is similar to m_axis_data_tvalid, the difference being that RDY is not asserted until the first symbol sampled on the Data Input Channel finally appears on DOUT. m_axis_data_tvalid is asserted as each zero (or other symbol) that was in the branch memory prior to the first symbol pulse is flushed out. RDY is not asserted for this data. If branch 0 has length 0, RDY and m_axis_data_tvalid are identical. This is because the first symbol output on DOUT after the first symbol is sampled is the first symbol sampled.
BLOCK_START	1	No	Rectangular mode only. BLOCK_START has similar functionality to FDO in the Convolutional interleaver. It is asserted High when the first symbol of a block appears on DOUT. The BLOCK_START delay is dependent on the latency and the block size. The BLOCK_START output is useful for synchronizing circuits downstream of the interleaver. It can be used to tell those circuits when to start a new block.
BLOCK_END	1	No	Rectangular mode only. BLOCK_END is asserted High when the last symbol of a block appears on DOUT.

TUSER Format

The fields are packed into the `m_axis_data_tuser` vector in the following order (starting from the LSB):

1. FDO
2. RDY
3. BLOCK_START
4. BLOCK_END

The fields in TUSER are only present if supported by the core configuration. When a field is omitted, the fields in Figure 3-4 are moved down the vector as close to bit 0 as possible. That is, if BLOCK_START and BLOCK_END are selected, BLOCK_START is at bit 0 and BLOCK_END is at bit 1. If only BLOCK_END is selected, it is at bit 0.

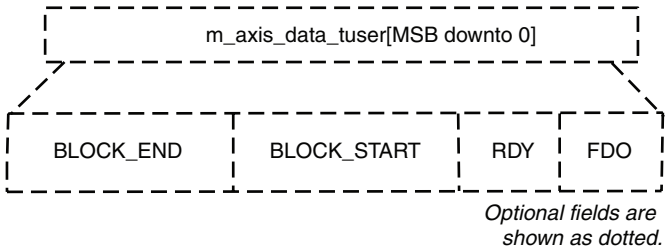


Figure 3-4: **Data Output Channel TUSER (m_axis_data_tuser) Format**

Control Channel

The Control Channel (`s_axis_ctrl`) contains the information needed to configure the interleaving or de-interleaving process. The Control Channel blocks the first symbol for a block in the Data Input Channel. That is, if the core has a Control Channel, the first symbol for a block remains in the Data Input Channel until a control word is received in the Control Channel.

Pinout

Table 3-6: **Control Channel Pinout**

Port Name	Port Width	Direction	Description
s_axis_ctrl_tdata	Variable. See the Vivado Customize IP dialog box when configuring the Symbol Interleaver/ De-interleaver	In	Can contain the following field in Forney mode: ◦ CONFIG_SEL Can contain the following fields in Rectangular mode: ◦ ROW ◦ ROW_SEL ◦ COL ◦ COL_SEL ◦ BLOCK_SIZE
s_axis_ctrl_tvalid	1	In	Asserted by the external master to signal that it is able to provide data.
s_axis_ctrl_tready	1	Out	Asserted by the Symbol Interleaver/ De-interleaver to signal that it is able to accept data.

TDATA Fields

In Forney mode the TDATA vector consists of one field (CONFIG_SEL). This is optional, and when it is not present the entire control channel is removed from the core interface.

In Rectangular mode, it can contain ROW, ROW_SEL, COL, COL_SEL and BLOCK_SIZE, although not all at the same time. These fields are all optional, and some are mutually exclusive (ROW with ROW_SEL, and COL with COL_SEL). When a field is not needed for a particular core configuration, it is not included in the Control Channel. In the case where no fields are present, the entire Control Channel is removed from the core interface.

Table 3-7: **Control Channel TDATA Fields**

Field Name	Description
CONFIG_SEL	If the core is configured to have more than one configuration then CONFIG_SEL selects which configuration to use for the current block. CONFIG_SEL is sampled when the first symbol is sampled. More details on the use of multiple configurations are given in Configuration Swapping, page 10.
ROW	This optional field is automatically selected if the variable number of rows option is chosen in the GUI. The width of the input is also entered in the GUI. Care should be taken not to make this field any wider than necessary, as doing so increases the size of the core. ROW is resampled at the start of each new block when the first symbol is sampled. The number sampled from the ROW input tells the core how many rows are in the block. Illegal values on ROW are indicated on the event_row_valid output.

Table 3-7: **Control Channel TDATA Fields *(Cont'd)***

Field Name	Description
ROW_SEL	This optional field is automatically selected if the selectable number of rows option is chosen in the GUI. The width of the input is determined from the number of selectable rows entered in the GUI. Care should be taken not to make this any larger than necessary, as doing so increases the size of the core. ROW_SEL is resampled at the start of each new block when the first symbol is sampled. The number sampled from the ROW_SEL input tells the core which value to use from the COE file. 0 means use the first value, 1 the second, and so on. An illegal value on ROW_SEL is indicated on the event_row_sel_valid output. An example of an illegal ROW_SEL value would be three possible numbers of rows being defined in the COE file. These would be selected using a two-bit ROW_SEL bus. '00,' '01' and '10' would all be valid ROW_SEL inputs, but '11' would be illegal because there is no corresponding row number for that ROW_SEL value in the COE file.
COL	This optional field is automatically selected if the variable number of columns option is chosen in the GUI. The width of the input is also entered in the GUI. Care should be taken not to make this field any wider than necessary, as this increases the size of the core. COL is resampled at the start of each new block when the first symbol is sampled. The number sampled from the COL input tells the core how many columns there are in the block. Illegal values on COL are indicated on the event_col_valid output.
COL_SEL	This optional field is automatically selected if the selectable number of columns option is chosen in the GUI. The width of the input is determined from the number of selectable columns entered in the GUI. Care should be taken not to make this any larger than necessary, as this increases the size of the core. COL_SEL is resampled at the start of each new block when the first symbol is sampled. The number sampled from the COL_SEL input tells the core which value to use from the COE file. 0 means use the first value, 1 the second, and so on. The value sampled on COL_SEL must correspond to a predefined number of columns in the COE file. An illegal value on COL_SEL is indicated on the event_col_sel_valid output. An example of an illegal COL_SEL value would be three possible numbers of columns being defined in the COE file. These would be selected using a two-bit COL_SEL bus. '00', '01' and '10' would all be valid COL_SEL inputs, but '11' would be illegal because there is no corresponding column number for that COL_SEL value in the COE file.
BLOCK_SIZE	This optional field is automatically selected if variable block sizes are chosen in the GUI. The width of the input is also entered in the GUI. Take care not to make this field any wider than necessary, as this increases the size of the core. BLOCK_SIZE is resampled at the start of each new block when the first symbol is sampled. Illegal values on BLOCK_SIZE are indicated on the event_block_size_valid output. See Block Size Type, page 52, for details of illegal BLOCK_SIZE values.

All fields should be 0 extended to the next 8 bit boundary if they do not already finish on an 8 bit boundary.

All field widths are variable. See the CORE GUI when configuring the Symbol Interleaver/ De-interleaver.

TDATA Format

The Control Channel TDATA vector is structured as shown in Figure 3-5 (starting from the LSB):

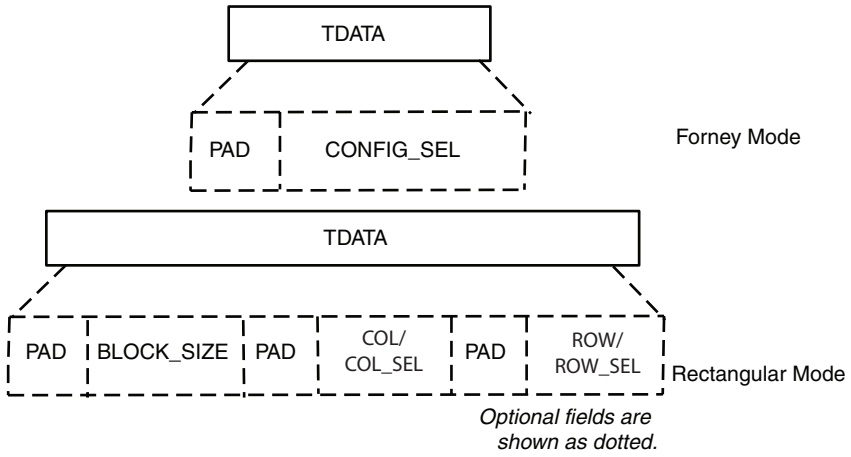


Figure 3-5: **Control Channel TDATA (s_axis_ctrl_tdata) Format**

External Symbol Memory

The core can use internal or external memory to store the data symbols. External memory might be necessary if there is insufficient block RAM left in the FPGA to implement a large interleaver.

The Convolutional Interleaver requires a dual-port RAM, with separate addresses for read and write ports. The Rectangular Block interleaver requires only a single-port RAM.

The connections for a Convolutional Interleaver example are shown in Figure 3-6. It is assumed the dual-port RAM behaves in the same way as a standard Xilinx block RAM. The ports shown are the standard Xilinx block RAM ports. The ENA/ENB ports act as clock enables for operations on the A and B ports respectively. In this case the A port is used for write operations and the B port for read operations.

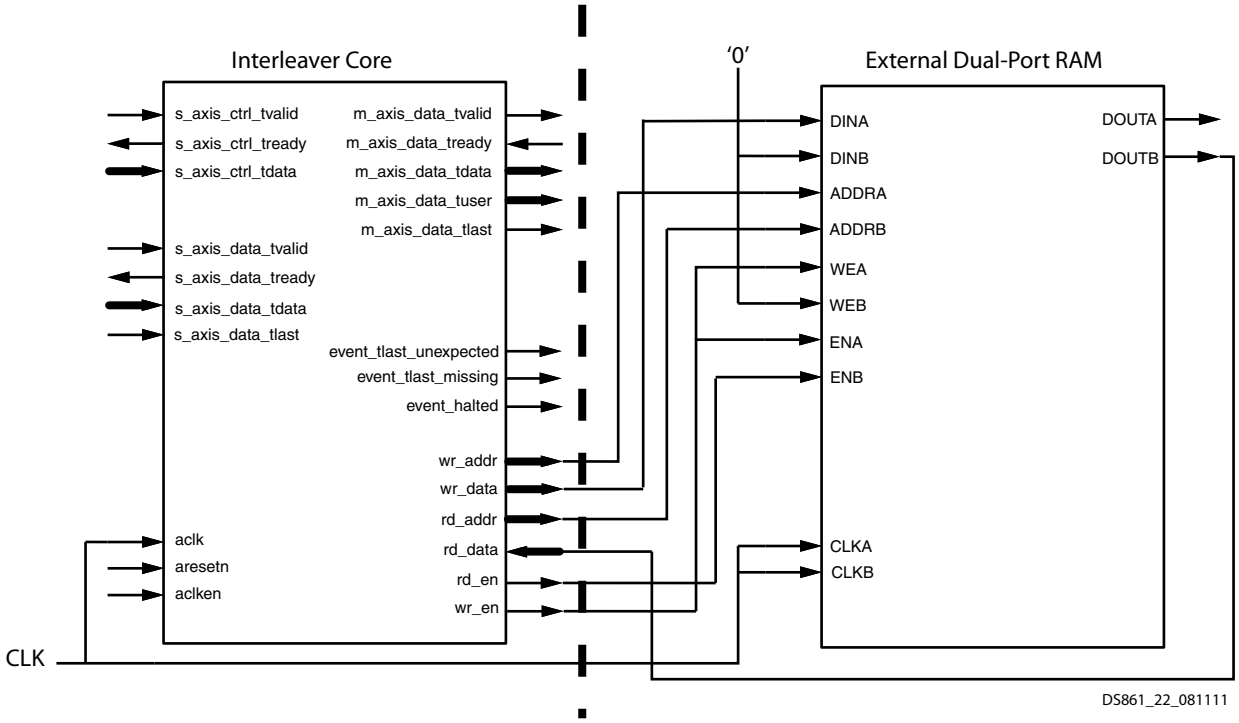


Figure 3-6: **Connections to External Symbol RAM for Convolutional Interleaver**

Table 3-8: **External Memory Pinout Table for Forney Cores**

Signal	Direction	Description
RD_ADDR	Output	This is the read address for the external symbol RAM. RD_ADDR should be connected to the RAM address bus corresponding to the port used by RD_DATA. RD_ADDR must be exactly wide enough to address the required symbol memory depth. The Vivado IP catalog calculates this automatically and generates a core with appropriately sized address buses. The required address bus width is also displayed on the last page of the core GUI.
RD_DATA	Input	This is the read data bus from the external symbol RAM. It has the same width as the DIN field in the Data Input Channel (ignoring padding).
RD_EN	Output	This is the read enable for the RAM. The core asserts the RD_EN output High when there is a valid read address on RD_ADDR and it needs to read the contents of that RAM location.
WR_ADDR	Output	This is the write address for the external symbol RAM. WR_ADDR should be connected to the RAM address bus corresponding to the port used by WR_DATA. WR_ADDR must be the same width as RD_ADDR.
WR_DATA	Output	This is the write data bus to the external symbol RAM. It has the same width as the DIN field in the Data Input Channel (ignoring padding).
WR_EN	Output	This is the write enable for the external symbol RAM. The core asserts the WR_EN output High when there is a valid write address on WR_ADDR and it needs to write to that RAM location.

The connections for a Rectangular Block Interleaver are shown in Figure 3-7. This is similar to the dual-port case; however, the Rectangular Block Interleaver does not perform simultaneous reads and writes so only a single port is required. The EN input acts as a clock

enable for all operations. If the core has an `aclken` input, then the RAM should be enabled with the same signal. The clock enable could also be sourced from inside the FPGA.

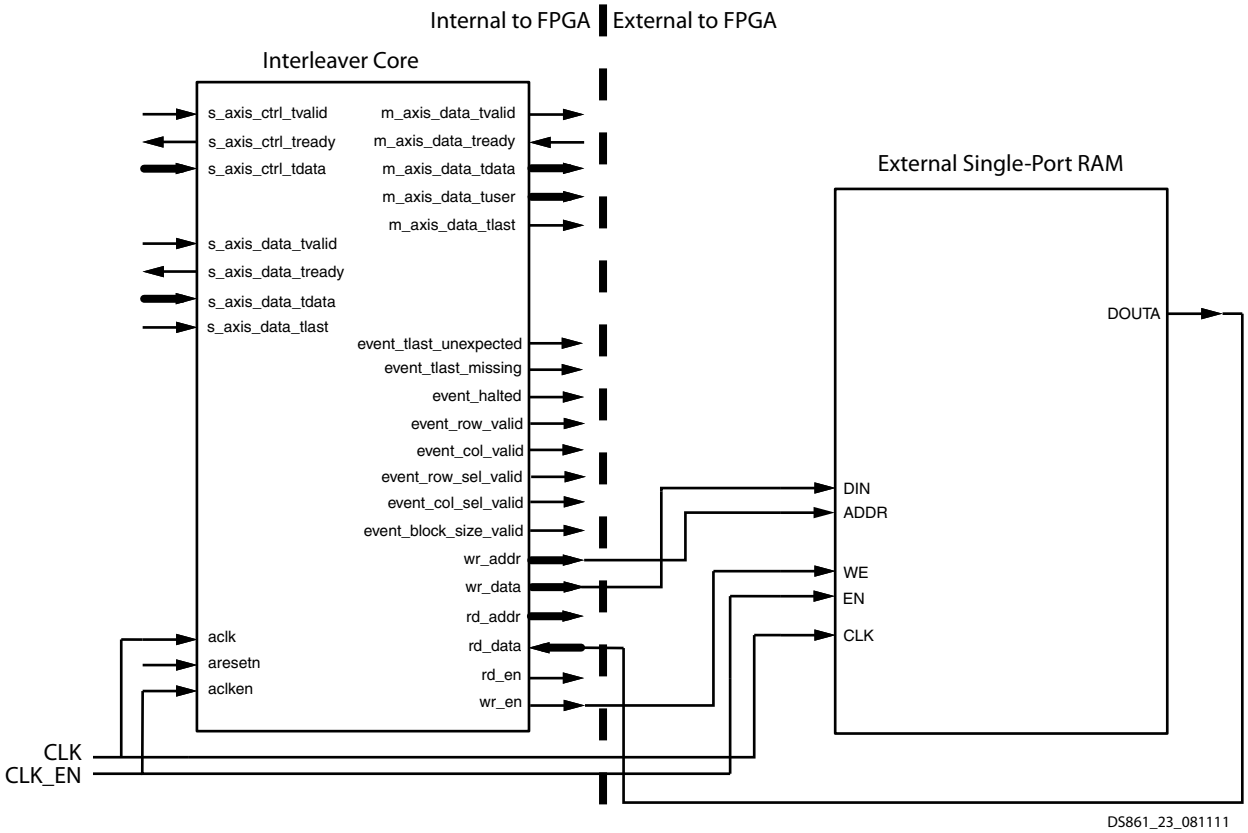


Figure 3-7: **Connections to External Symbol RAM for Rectangular Interleaver**

Table 3-9: **External Memory Pinout Table for Rectangular Cores**

Signal	Direction	Description
RD_ADDR	Output	This is the address for the external symbol RAM. As the RAM is only single-port for the Rectangular Block Interleaver, there only needs to be one address bus. RD_ADDR and WR_ADDR are actually identical for the Rectangular Block Interleaver.
RD_DATA	Input	This is the read data bus from the external symbol RAM. It has the same width as the DIN field in the Data Input Channel (ignoring padding).
RD_EN	Output	This is the read enable for the RAM. The Rectangular Block Interleaver uses only a single port symbol RAM and RD_EN is driven High all the time. It is High even when aclken is Low, so care must be taken if using it to drive an enable input external to the core. It is necessary to AND it with aclken.
WR_ADDR	Output	This is the write address and is just a duplicate of RD_ADDR.
WR_DATA	Output	This is the write data bus to the external symbol RAM. It has the same width as the DIN field in the Data Input Channel (ignoring padding).
WR_EN	Output	This is the write enable for the external symbol RAM. The core asserts the WR_EN output High when there is a valid write address on WR_ADDR and it needs to write to that RAM location.

All the outputs for the external RAM are registered in the core. To achieve predictable timing when accessing the external RAM, these registers should be mapped into IOBs using the appropriate mapper options. It is possible to insert additional registers on the external memory interface if required to meet timing. The same number of registers must be added to every output signal, but they do not have to match the number on rd_data. The delay in clock cycles added by the registers on the outbound path and the inbound path must be specified in the GUI "External Memory Latency" edit box. See Figure 3-8 for an example. In this case, the outbound latency is 3 and the inbound latency is 1, so EXTERNAL_MEMORY_LATENCY should be set to 4. This increases the latency of the interleaver core by the number of additional registers inserted.

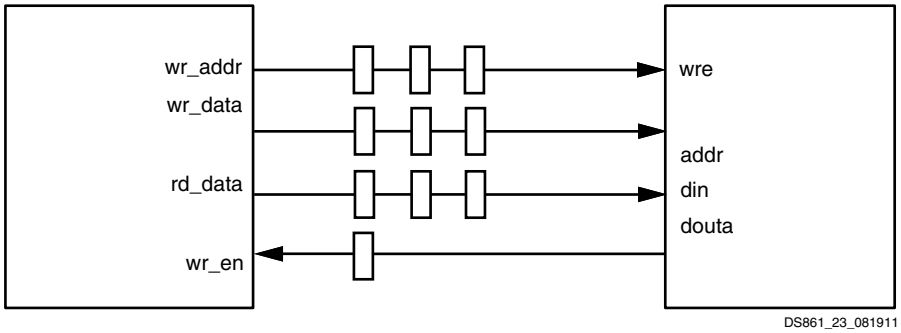


Figure 3-8: **Calculating EXTERNAL_MEMORY_LATENCY**

The Vivado IP catalog feature automatically determines the required address bus width. This value is also displayed on the final page of the core GUI.

The timing for accessing the external symbol RAM is identical to the timing for accessing a synchronous dual-port block RAM inside the FPGA.

Figure 3-9 shows the timing of the dual-port external RAM interface for a Convolutional Interleaver example. The first write to external memory occurs at t2 when 1 is written to address 0. The first read occurs at t3 when address 1 is read. This causes the first real symbol to appear on RD_DATA at t4 and as the EXTERNAL_MEMORY_LATENCY is 0 in this example, and because there is no TREADY on the Data Output Channel, the symbol is passed straight through to `m_axis_data_tdata`. If TREADY had been enabled on this channel, it would take a further 2 clock cycles for the data to appear. The value of this symbol is shown XX as because this is a residual value left over from the previous interleave operation.

No write occurs at t4 because `s_axis_data_tvalid` was Low at t1. Similarly RD_EN is Low at t5, so no read occurs at that time and `m_axis_data_tvalid` is 0 at t6.

The timing diagram does not show realistic clock-to-output delays, as this is a function of where the final output registers are placed in the FPGA (preferably in the IOBs) and the PCB layout.

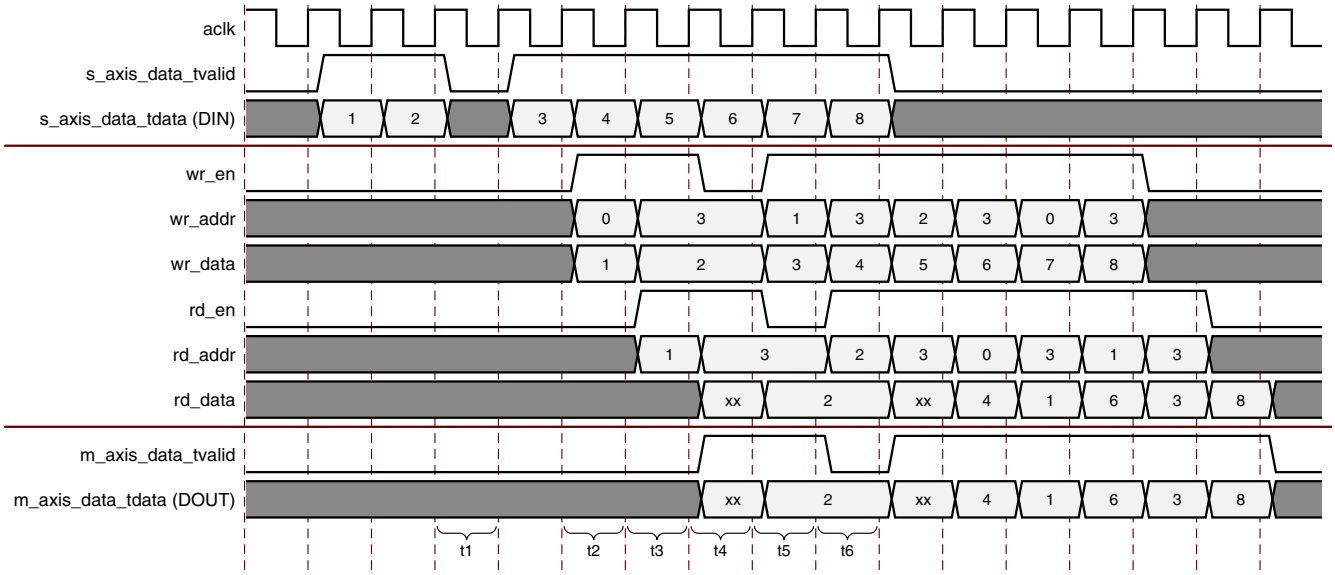


Figure 3-9: **External RAM Interface Timing when EXTERNAL_MEMORY_LATENCY = 0**

Figure 3-10 shows the same design but with an EXTERNAL_MEMORY_LATENCY of 3. The effect of this is to delay the symbols appearing on the Data Output Channel.

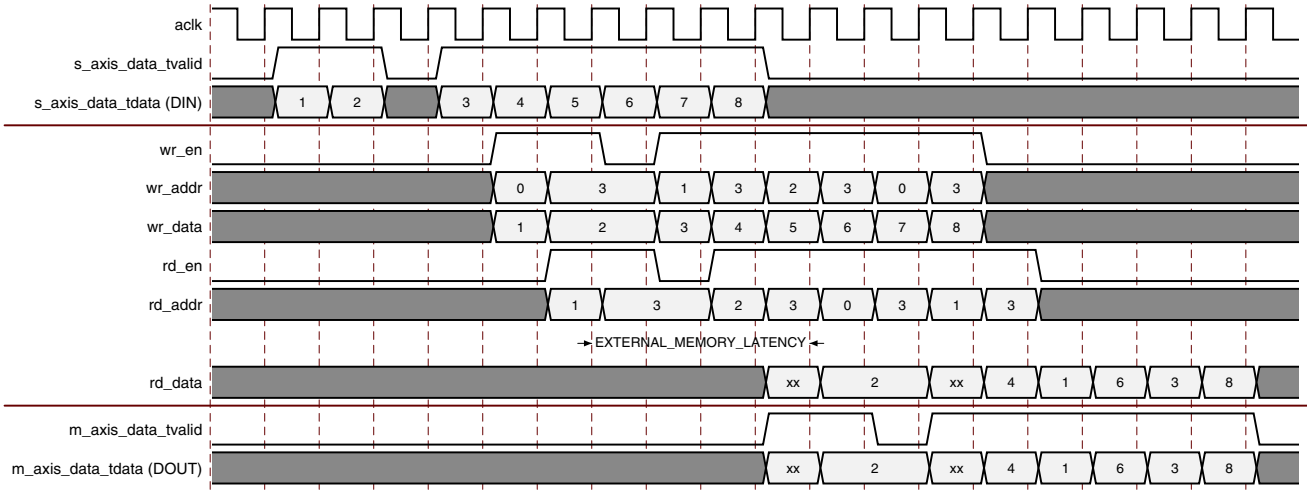


Figure 3-10: **External RAM Interface Timing when EXTERNAL_MEMORY_LATENCY = 3**

Event Signals

The Symbol Interleaver/De-interleaver core provides some real-time non-AXI signals to report information about the core status. These event signals are intended for use by reactive components such as interrupt controllers. These signals are not optionally configurable from the GUI, but are removed by synthesis tools if left unconnected.

event_tlast_missing

This event signal is asserted for a single clock cycle on the last symbol of an incoming block if `s_axis_data_tlast` is not seen asserted with that symbol. This event is only present in Rectangular mode.

event_tlast_unexpected

This event signal is asserted on every clock cycle where `s_axis_data_tlast` is unexpectedly asserted. That is, asserted on a symbol that is not the last symbol. The meaning of "Last Symbol" varies between Forney and Rectangular mode:

- Forney - the symbol corresponding to the last branch
- Rectangular - the final symbol loaded for a block

If there are multiple unexpected highs on `s_axis_data_tlast`, then this is asserted for each of them.

event_halted

This event is asserted on every cycle where the Symbol Interleaver/De-interleaver needs to write data to the Data Output Channel but cannot because the buffers in the channel are full. When this occurs, the core is halted and all activity stops until space is available in the channel buffers. The event pin is only available when HAS_DOUT_TREADY is TRUE.

event_block_size_valid

This output is available when the block size is not constant. That is, if the block size type is either variable or equal to R * C.

If the block size type is variable, `event_block_size_valid` signals whether a legal or illegal value is sampled on the BLOCK_SIZE field of the Control Channel. If an illegal value is sampled, `event_block_size_valid` goes Low a predefined number of clock cycles later.

If the block size type is R * C, `event_block_size_valid` signals that the block size, generated from the values sampled on ROW or ROW_SEL, or on COL or COL_SEL, is legal.

Table 3-10 lists the number of clock cycles from the first symbol being sampled to `event_block_size_valid` (or any of the other *_VALID outputs) changing. After a *_VALID output has gone Low, it remains Low until "Valid Delay" clock cycles after the next block is started. The latency is defined in Table 2-3 and always equals a constant. Thus, the Valid Delay is always constant.

Table 3-10: **Clock Cycles from First Symbol**

Block Size Type	Selectable Rows or Columns	Valid Delay (clock cycles)
Variable	-	latency-2
Row x Column	No	3
Row x Column	Yes	5

Regardless of the block size type chosen, the block size must never go below the absolute minimum value given in Table 4-2.

event_col_valid

This optional output is available when a variable number of columns is selected. If an illegal value is sampled on the COL field of the Control Channel, `event_col_valid` goes Low a predefined number of clock cycles later. Table 3-10 lists the number of clock cycles from the first symbol being sampled High to `event_col_valid` changing.

See Table 4-2 for details of illegal COL values.

event_col_sel_valid

This optional output is available when a selectable number of columns is chosen. If an illegal value is sampled on the COL_SEL field of the Control Channel, `event_col_sel_valid` goes Low a predefined number of clock cycles later. Table 3-10 lists the number of clock cycles from the first symbol being sampled High to `event_col_sel_valid` changing.

See Table 3-7 for an explanation of an illegal COL_SEL value.

event_row_valid

This optional output is available when a variable number of rows is selected. If an illegal value is sampled on the ROW field of the Control Channel, `event_row_valid` goes Low a predefined number of clock cycles later. Table 3-10 lists the number of clock cycles from the first symbol being sampled High to `event_row_valid` changing.

See Table 4-2 for details of illegal ROW values.

event_row_sel_valid

This optional output is available when a selectable number of rows is chosen. If an illegal value is sampled on the ROW_SEL field of the Control Channel, `event_row_sel_valid` goes Low a predefined number of clock cycles later. Table 3-10 lists the number of clock cycles from the first symbol being sampled High to `event_row_sel_valid` changing.

See Table 3-7 for an explanation of an illegal `ROW_SEL` value.

Design Flow Steps

This chapter describes customizing and generating the core, constraining the core, and the simulation, synthesis and implementation steps that are specific to this IP core. More detailed information about the standard Vivado® design flows and the IP integrator can be found in the following Vivado Design Suite user guides:

- *Vivado Design Suite User Guide: Designing IP Subsystems using IP integrator* (UG994) [Ref 3]
- *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 4]
- *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 5]
- *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 7]

Customizing and Generating the Core

This chapter includes information on using Xilinx tools to customize and generate the Symbol Interleaver/De-interleaver core.

If you are customizing and generating the core in the Vivado IP integrator, see the *Vivado Design Suite User Guide: Designing IP Subsystems using IP integrator* (UG994) [Ref 3] for detailed information. IP integrator might auto-compute certain configuration values when validating or generating the design. To check whether the values do change, see the description of the parameter in this chapter. To view the parameter value you can run the `validate_bd_design` command in the Tcl Console.

Vivado Integrated Design Environment

You can customize the IP for use in your design by specifying values for the various parameters associated with the IP core using the following steps:

1. Select the IP from the IP catalog.
2. Double-click the selected IP or select the Customize IP command from the toolbar or right-click menu.

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 4] and the "Working with the Vivado IDE" section in the *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 5].

Core Parameters

The Vivado® Customize IP dialog box for the Symbol Interleaver/De-interleaver core uses several screens for setting core parameters. To move between screens, click **Next** or **Back**. After selecting the desired parameters for type of core you want, click **Generate** to generate the core when you reach the final screen.

Some parameters are relevant to both types of interleavers, and some are specific to only one type.

Component Name

Used as the base name of the output files generated for the core. Names must begin with a letter and must be composed of the following characters: a to z, 0 to 9 and "_".

Memory Style

The following options are available:

- **Distributed.** The core should not use any block memories, if possible. This is useful if they are required elsewhere in the design.
- **Block.** The core should use block memories wherever possible. This keeps the number of slices used to a minimum, but might waste block memory.
- **Automatic.** Allow the core to use the most appropriate style of memory for each case, based on required memory depth.

Symbol Width

The number of bits in the symbols to be processed. In IP integrator, this parameter is auto-updated.

Symbol Memory

Allows the symbol memory to be specified as internal or external. If external is selected, then all the optional pins required for external memory access are automatically added. When external memory is selected the 'External Symbol Memory Latency' edit box becomes enabled. The value entered here must match the total number of cycles of latency that have been added to the external memory interface *in the system*. This includes both the outbound and inbound signals. See External Symbol Memory for more details.

Type

Determines whether the core is to be an interleaver or de-interleaver.

For the Forney Convolutional type, the branch lengths are increased from branch 0, as shown in Figure 2-1, or decreased from branch 0, as shown in Figure 2-2. If the branch lengths are specified in a file, the mode is irrelevant.

For the Rectangular Block type, this determines whether a write rows, read columns or write columns, read rows operation is performed.

Pipelining

Three levels of pipelining are available. Select **Maximum** if speed is important. This might result in a slight increase in area. The latency of the Convolutional type also increases slightly.

RECOMMENDED: *In general, it is recommended that **Maximum** pipelining is used.*

Medium and **Maximum** pipelining have the effect of adding registers to certain inputs in the Convolutional type. **Maximum** pipelining results in some additional internal registering within the core compared to **Medium** pipelining.

In the Rectangular Block type, several internal circuits are pipelined to improve performance. **Medium** and **Maximum** pipelining are actually identical for the Rectangular Block type.

Optional Pins

Check the boxes of the optional pins that are required. Each selected pin can result in more FPGA resources being used and can result in a reduced maximum operating frequency.

Control Signals

ACLKEN Input

The `aclken` (Clock Enable) input is an optional input pin. When `aclken` is deasserted (Low), all the synchronous inputs are ignored and the core remains in its current state. If `aresetn` and `aclken` are asserted at the same time, `aresetn` takes precedence.

ARESETN Input

When `aresetn` (active-Low reset) is asserted (Low), all the core flip-flops are synchronously initialized. The core remains in this state until `aresetn` is deasserted. If `aresetn` and `aclken` are asserted at the same time, `aresetn` takes precedence. The synchronous initialization resets only the core control logic. The symbol memory itself is not cleared. `ARESETn` is an optional pin.

User Parameters

Table 4-1 shows the relationship between the GUI fields in the Vivado IDE (described in Vivado Integrated Design Environment) and the User Parameters (which can be viewed in the Tcl console).

Table 4-1: **GUI Parameter to User Parameter Relationship**

GUI Parameter/Value[1]	User Parameter/Value[1]	Default Value
Memory Style	memory_style	Automatic
Automatic	automatic	
Block	block	
Distributed	distributed	
Symbol Width	symbol_width	1
Type	sid_type	Forney
Forney Convolutional	forney	
Rectangular Block	rectangular	
Mode	mode	Interleaver
Deinterleaver	deinterleaver	
Interleaver	interleaver	
Symbol Memory	symbol_memory	Internal
External	external	
Internal	internal	
Number of Branches	number_of_branches	16
Number of Configurations	number_of_configurations	1
Branch Length Type	branch_length_type	constant_difference_between_ consecutive_branches
Constant Difference Between Consecutive Branches	constant_difference_between_ consecutive_branches	
Use COE file to define branch length constants for each configuration	coe_file_defines_branch_length_ constant_for_each_configuration	
Use COE file to define Branch Lengths	use_coe_file_to_define_branch_ lengths	
Use COE file to define individual branch lengths for every branch in each configuration	coe_file_defines_individual_ branch_lengths_for_every_branch_ in_each_configuration	
Architecture	architecture	Rom_based
Logic-based	logic_based	
ROM-based	rom_based	
Length of Branches: Value	branch_length_constant	1

Table 4-1: **GUI Parameter to User Parameter Relationship *(Cont'd)***

GUI Parameter/Value[1]	User Parameter/Value[1]	Default Value
Length of Branches: COE File	branch_length_coe_file_name	No_coe_file_loaded
Number of Rows	number_of_rows	Constant
Constant	constant	
Selectable	selectable	
Variable	variable	
Rows Definition: Value	number_of_rows_constant_value	15
Row Port Width	row_port_width	4
Number of Columns	number_of_columns	Constant
Constant	constant	
Selectable	selectable	
Variable	variable	
Block Size	block_size_type	Rows_columns
Constant	constant	
Rows/Columns	rows_columns	
Variable	variable	
Minimum Number of Rows	minimum_rows	15
Number of Values	number_of_rows_ selectable_value	4
Row Permutations	row_permutations	None
None	none	
Use COE file to define row permutations	use_coe_file_to_define_row_ permutations	
Column Definition: Value	number_of_columns_constant_value	15
COL Port Width	col_port_width	4
Minimum Number of Columns	minimum_columns	15
Number of Values	number_of_columns_selectable_ value	4
Column Permutations	column_permutations	none
None	none	
Use COE file to define row permutations	use_coe_file_to_define_row_ permutations	
Block Size: Value	block_size_constant_value	225
Block Size: COE File	coe_file	No_coe_file_loaded
BLOCK_SIZE Port Width	block_size_port_width	3

Table 4-1: **GUI Parameter to User Parameter Relationship *(Cont'd)***

GUI Parameter/Value[1]	User Parameter/Value[1]	Default Value
ROW_VALID	has_row_valid	False
ROW_SEL_VALID	has_row_sel_valid	False
COL_VALID	has_col_valid	False
COL_SEL_VALID	has_col_sel_valid	False
BLOCK_SIZE_VALID	has_block_size_valid	False
BLOCK_START	has_block_start	False
BLOCK_END	has_block_end	False
ACLKEN	has_aclken	False
ARESETn	has_aresetn	False
RDY	has_rdy	False
FDO	has_fdo	False
Pipelining	pipelining	maximum
Maximum	maximum	
Medium	medium	
Minimum	minimum	
TREADY	has_dout_tready	False
External Memory Latency	external_memory_latency	0

1. When the GUI parameter value differs from the user parameter value, those parameter values are indented below the associated parameter.

Forney Convolutional Specific Parameters

Number of Branches

The value for the variable *B* in Figure 2-1, Figure 2-2, and Figure 2-3. The value must always be specified, even when using a file to define branch lengths.

Length of Branches

Either a constant difference between branch lengths, as in Figure 2-1 and Figure 2-2, or the branch lengths specified from a file, as in Figure 2-3. Figure 4-2 provides an example of the file syntax in the latter case.

Number of Configurations

If greater than 1, the core is generated with an AXI4-Stream Control Channel containing the CONFIG_SEL field. The parameters for each configuration are defined in a COE File Format.

The number of parameters defined must exactly match the number of configurations specified.

Architecture

Controls whether look-up table ROMs or logic circuits are used to compute some of the internal results in the core. Which option is best depends on the other core parameters. It is recommended that both options are tried. This parameter has no effect on the core behavior.

Rectangular Block-Specific Parameters

Block Size Type

There are three possibilities:

1. **Constant**. Block size never changes. The block can be pruned (block size < row * col). The block size must be chosen so that the last symbol is on the last row. An unpruned interleaver uses a smaller quantity of FPGA resources than a pruned one, so pruning should be used only if necessary.

   Figure 4-1 provides an example with three rows and four columns. Using the preceding rule, the only legal block sizes are 9, 10, 11 and 12. This block size type can be used only if the row and column types are also set to constant. Row and column permutations are not supported for pruned block sizes.

2. **Rows * Columns**. If the number of rows and columns is constant, selecting this option has the same effect as setting the block size type to constant and entering a value of rows * columns for the block size.

   If the number of rows or columns is not constant, selecting this option means the core calculates the block size automatically whenever a new row or column value is sampled. Pruning is impossible with this block size type.

3. **Variable**. Block size is sampled from the BLOCK_SIZE field in the Control Channel at the beginning of every block. The value sampled on BLOCK_SIZE must be such that the last symbol falls on the last row, as previously described.

   If the block size is already available external to the core, selecting this option is usually more efficient than selecting “rows * columns” for the block size type. Row and column permutations are not supported for the variable block size type.

0	1	2	3
4	5	6	7
8	9	X	X

Block Size = 10 - Legal

0	1	2	3
4	5	6	X
X	X	X	X

Block Size = 7 - Illegal

Figure 4-1: **Legal and Illegal Block Sizes**

Block Size Constant Value

This parameter is relevant only if constant block size type is selected. It must meet the constraints described in the Block Size Type.

BLOCK_SIZE Field Width

This parameter is relevant only if variable block size type is selected. It sets the width of the BLOCK_SIZE field in the Control Channel (ignoring padding). The smallest possible value should be used to keep the core as small as possible.

Column Type

There are three possibilities:

1. **Constant**. The number of columns is always equal to the Column Constant Value parameter.
2. **Variable**. The number of columns is sampled from the COL field in the Control Channel at the start of each new block. Column permutations are not supported for the variable column type.
3. **Selectable**. The COL_SEL field in the Control Channel is sampled at the start of each new block. This value is then used to select from one of the possible values for number of columns provided in the COE file.

Column Constant Value

This parameter is relevant only if constant column type is selected. The number of columns is fixed at this value.

COL Field Width

This parameter is relevant only if variable column type is selected. It sets the width of the COL field in the Control Channel (ignoring padding). The smallest possible value should be used to keep the core as small as possible.

Minimum Number of Columns

This parameter is relevant only if variable column type is selected. In this case, the core has to cope potentially with a wide range of possible values for the number of columns. If the smallest value that can actually occur is known, then the amount of logic in the core can sometimes be reduced. The largest possible value should be used for this parameter to keep the core as small as possible.

Number of Selectable Columns

If the selectable column type has been chosen, this parameter defines how many valid selection values have been defined in the COE file. Only add select values you need.

Use Column Permute File

This tells the Vivado IP catalog that a column permute vector exists in the COE file and column permutations are to be performed. Remember this is only possible for unpruned interleaver/de-interleavers.

Row Type

There are three possibilities:

1. **Constant**. The number of rows is always equal to the Row Constant Value parameter.
2. **Variable**. The number of rows is sampled from the ROW field in the Control Channel at the start of each new block. Row permutations are not supported for the variable row type.
3. **Selectable**. The ROW_SEL field in the Control Channel is sampled at the start of each new block. This value is then used to select from one of the possible values for number of rows provided in the COE file.

Row Constant Value

This parameter is relevant only if constant row type is selected. The number of rows is fixed at this value.

ROW Field Width

This parameter is relevant only if variable row type is selected. It sets the width of the ROW field in the Control Channel (ignoring padding). The smallest possible value should be used to keep the core as small as possible.

Number of Selectable Rows

If the selectable row type has been chosen, this parameter defines how many valid selection values have been defined in the COE file. Only add select values you need.

Minimum Number of Rows

This parameter is relevant only if variable row type is selected. In this case the core has to potentially cope with a wide range of possible values for the number of rows. If the smallest value that can actually occur is known, then the amount of logic in the core can sometimes be reduced. The largest possible value should be used for this parameter to keep the core as small as possible.

Use Row Permute File

This tells the Vivado IP catalog that a row permute vector exists in the COE file, and row permutations are to be performed. Remember this is possible only for unpruned interleaver/de-interleavers.

Parameter Ranges

Valid ranges for the parameters are provided in Table 4-2.

Table 4-2: **Parameter Ranges**

Parameter	Min	Max
Symbol Width	1	256
Forney Convolutional:		
Number of Configurations	1	256
Number of Branches(1)	2	256
Branch Length Constant(1)(2)	1	
Branch Lengths(1)	1	
Rectangular Block:		
Block Size Constant(1)(3)	6	65025
Block Size Width(1)	3	16
Column Constant(1)(4)	2	255
Column Width(1)	2	8

Table 4-2: **Parameter Ranges *(Cont'd)***

Parameter	Min	Max
Minimum Number of Columns[(1)(4)]	2	255
Number of Selectable Columns	2	32
Row Constant[(1)(4)]	1	255
Row Width[(1)]	1	8
Minimum Number of Rows[(1)(4)]	1	255
Number of Selectable Rows	2	32

Notes:

1. This parameter is limited such that the maximum depth of individual memories within the core do not exceed certain limits. The GUI detects if these limits have been exceeded. This can mean the maximum value allowed by the GUI appears to be less than the absolute maximum value given in Table 4-2. In reality, these parameters are limited by the maximum size of device available.
2. The branch length constant is the value entered as the constant difference between consecutive branches. The GUI displays the range of legal values, based on the restrictions mentioned in Note 1.
3. Block Size Constant must be within the following range: (R-1) * C < Block Size Constant <= R * C, where R = number of rows and C = number of columns. If there is only a single row, then Block Size Constant must equal the number of columns.
4. The resulting block size must be within the absolute limits for Block SIze Constant given in this table.

Using the Symbol Interleaver/De-interleaver IP Core

The Vivado Customize IP dialog box performs error-checking on all input parameters. Resource estimation and latency information are also available.

Several files are produced when a core is generated, and customized instantiation templates for Verilog and VHDL design flows are provided in the `.veo` and `.vho` files, respectively. For detailed instructions, see the Vivado Design Suite documentation.

COE File Format

In certain cases, some parameter values are passed to the Vivado IP catalog using a COE (COEfficient) file. This is an ASCII text file with a single radix header followed by several vectors. The radix can be 2, 10, or 16. Each vector must be terminated by a semi-colon.

The GUI reads the COE file and writes out one or more MIF files when the core is generated. The VHDL and Verilog behavioral simulation models for the core rely on these MIF files. For correct operation when using MIF files, they must be copied to the directory in which the simulation is to be run. The vectors used in the COE file differ depending on whether the core is a Forney Convolutional or Rectangular Block type.

Forney Convolutional Type

If the branch lengths are defined in a file, then it must be a correctly formatted COE file. The length of all the branches is defined in a comma-separated list in a branch_length_vector.

Figure 4-2 shows an example COE file for a Forney Convolution interleaver with eight branches. The number of branches must also be set to 8 in the GUI.

```
radix=10;
branch_length_vector=
3,
10,
20,
40,
80,
160,
320,
640;
```

Figure 4-2: **Example Convolutional COE File**

The branch length values can also be placed on a single line as shown in Figure 4-3.

```
radix=10;
branch_length_vector=3,10,20,40,80,160,320,640;
```

Figure 4-3: **Example Convolutional COE File Values on a Single Line**

COE Files for Multiple Configurations

The COE file is used to specify the branch length constant and number of branches for each configuration.

Figure 4-4 shows a COE file for an example with 16 configurations. If the CONFIG_SEL field = 0 then the interleaver has 128 branches and a branch length constant of 1. If the CONFIG_SEL field = 3 then the interleaver has 64 branches and a branch length constant of 2.

```
radix=10;
number_of_branches_vector=
128,128,128,64,128,32,128,16,128,
8,128,128,128,128,128,128;
branch_length_constant_vector=
1,1,2,2,3,4,4,8,5,16,6,1,7,1,8,1;
```

Figure 4-4: **ITU J.83 Annex B COE File**

The number of elements in the `number_of_branches_vector` and the `branch_length_constant_vector` must equal the number of configurations.

If the number of configurations is not a power of two, then out-of-range values on the CONFIG_SEL field input results in the core selecting configuration 0.

It is possible to define the individual branch length for every branch in each configuration. If this option is selected, then the `branch_length_constant_vector` must be replaced with a `branch_length_vector`. The number of elements in this vector must be the exact sum of all the elements of the `number_of_branches_vector`. An example is shown in Figure 4-5. In this example, if the CONFIG_SEL field = 0, then an interleaver with branches of lengths 1, 2, 3, and 4 is selected. If the CONFIG_SEL field = 1, an interleaver with branches of 4, 3, 2, and 1 is selected. This is one way of having a single core switch between interleaving and de-interleaving. If the CONFIG_SEL field = 2, an interleaver with branches of 1, 4 and 5 is selected.

```
radix=10;
number_of_branches_vector=
4,4,3;
branch_length_vector=
1,2,3,4,4,3,2,1,1,4,5;
```

Figure 4-5: **Multiple Configuration COE File Defining Each Individual Branch Length**

Rectangular Block Type

If row or column permutations are to be used, then the row and/or column permutation vectors are passed to the Vivado IP catalog using a COE file. Figure 4-6 shows an example COE file for the permutations used in Figure 2-5.

```
radix=10;
row_permute_vector=
2,0,1;
col_permute_vector=
3,1,0,2;
```

Figure 4-6: **Example Rectangular COE File**

If the row or column type is "selectable," the row and/or column select vectors are also passed in using the COE file. These tell the core how to map the value sampled on the ROW_SEL and COL_SEL fields to a particular number of rows or columns. If row or column permutations are to be used in conjunction with selectable rows or columns, then it is possible to have a different permute vector for every row and column select value. For example, in Figure 4-7 there are three selectable row values. If the ROW_SEL field = '00' the interleaver has three rows. '01' gives four rows and '10' gives five rows. '11' is an illegal ROW_SEL value because a fourth value is not defined in the `row_select_vector`. Also, if the ROW_SEL field ='00' the row permute vector is [2, 0, 1]. If the ROW_SEL field='01' the

vector is [3, 2, 0, 1], and if the ROW_SEL field='10' the vector is [0, 1, 2, 3, 4] (that is, no row permutations). The `col_select_vector` and `col_permute` vector work in the same way.

The supplied COE file must be compatible with the other parameters entered in the GUI, such as number of selectable rows.

```
radix=10;
row_select_vector=
3,4,5;
col_select_vector=
4,6,5;
row_permute_vector=
2,0,1,
3,2,0,1,
0,1,2,3,4;
col_permute_vector=
3,1,0,2,
3,1,0,2,4,5,
2,1,3,0,4;
```

Figure 4-7: **Example Rectangular COE File with Selectable Rows and Columns**

Parameter Values

Table 4-3 defines valid entries for the parameters. Parameters are not case sensitive. Default values are displayed in bold.

Xilinx strongly recommends that parameters are not manually edited in the XCI file; instead, use the Vivado GUI to configure the core and perform range and parameter value checking.

Table 4-3: **Parameters**

XCI Parameter	Valid Values
architecture	**rom_based**, logic_based
has_block_end	true, **false**
block_size_constant_value	Integer in the range defined in Table 4-2 (default value is **15**)
block_size_port_width	Integer in the range defined in Table 4-2 (default value is **4**)
block_size_type	constant, **rows_columns**, variable
has_block_size_valid	true, **false**
has_block_start	true, **false**
branch_length_constant	Integer in the range defined in Table 4-2 (default value is **1**)

Table 4-3: **Parameters *(Cont'd)***

XCI Parameter	Valid Values
branch_length_type	**constant_difference_between_consecutive_branches** use_coe_file_to_define_branch_lengths coe_file_defines_branch_length_constant_for_each_configuration coe_file_defines_individual_branch_lengths_for_every_branch_in_each_configuration
has_ce	true, **false**
coefficient_file	Path of coe file if coe file required (default is blank)
column_permutations	**none**, use_coe_file_to_define_column_permutations
col_port_width	Integer in the range defined in Table 4-2 (default value is **4**)
has_col_sel_valid	true, **false**
has_col_valid	true, **false**
component_name	ASCII text starting with a letter and based upon the following character set: a..z, 0..9 and _ (default is blank)
has_fdo	true, **false**
memory_style	**automatic**, distributed, block
minimum_columns	Integer in the range defined in Table 4-2 (default value is **15**)
minimum_rows	Integer in the range defined in Table 4-2 (default value is **15**)
mode	**interleaver**, deinterleaver
number_of_branches	Integer in the range defined in Table 4-2 (default value is **16**)
number_of_columns	**constant**, selectable, variable
number_of_columns_constant_value	Integer in the range defined in Table 4-2 (default value is **15**)
number_of_columns_selectable_value	Integer in the range defined in Table 4-2 (default value is **4**)
number_of_configurations	Integer in the range defined in Table 4-2 (default values is **1**)
number_of_rows	**constant**, selectable, variable
number_of_rows_constant_value	Integer in the range defined in Table 4-2 (default value is **15**)
number_of_rows_selectable_value	Integer in the range defined in Table 4-2 (default value is **4**)
pipelining	minimum, **maximum**, medium
has_rdy	true, **false**
row_permutations	**none**, use_coe_file_to_define_row_permutations
row_port_width	Integer in the range defined in Table 4-2 (default value is **4**)
has_row_sel_valid	true, **false**
has_row_valid	true, **false**
has_aresetn	true, **false**

Table 4-3: **Parameters *(Cont'd)***

XCI Parameter	Valid Values
symbol_memory	**internal**, external
symbol_width	Integer in the range defined in Table 4-2 (default value is **1**)
type	**forney**, rectangular
has_dout_tready	true, **false**
external_memory_latency	**0** to 6

System Generator for DSP Graphical User Interface

The Symbol Interleaver/De-interleaver core is available through Xilinx System Generator for DSP, a design tool that enables the use of the model-based design environment Simulink® for FPGA design. The Symbol Interleaver/De-interleaver core is one of the DSP building blocks provided in the Xilinx blockset for Simulink. The core can be found in the Xilinx Blockset in the Communication section. The block is called "Interleaver/De-interleaver 7.1." See the System Generator User Manual for more information.

The controls in the System Generator GUI work identically to those in the Vivado IP catalog GUI, although the layout has changed slightly. See Core Parameters for detailed information about all other parameters.

Output Generation

For details, see "Generating IP Output Products" in the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 4].

Constraining the Core

There are no constraints associated with this core.

Simulation

For comprehensive information about Vivado® simulation components, as well as information about using supported third-party tools, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 7].

Synthesis and Implementation

For details about synthesis and implementation, see "Synthesizing IP" and "Implementing IP" in the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 4].

Example Design

No example design is provided for this core.

Test Bench

When the core is generated using the Vivado® IP catalog, a demonstration test bench is created. This is a simple VHDL test bench that exercises the core.

The demonstration test bench source code is one VHDL file: `demo_tb/tb_<component_name>.vhd` in the Vivado IP catalog output directory. The source code is comprehensively commented.

Using the Demonstration Test Bench

The demonstration test bench instantiates the generated Symbol Interleaver/De-interleaver core.

Compile the netlist and the demonstration test bench into the work library (see your simulator documentation for more information on how to do this). Then simulate the demonstration test bench. View the test bench signals in your simulator waveform viewer to see the operations of the test bench.

The Demonstration Test Bench in Detail

The demonstration test bench performs the following tasks:

- Instantiate the core
- Generate a clock signal
- Drive the core input signals to demonstrate core features (see following sections for details)
- Provide signals showing the separate fields of AXI4 TDATA and TUSER signals

The demonstration test bench drives the core input signals to demonstrate the features and modes of operation of the core. The test bench drives two blocks of incremental data into the Symbol Interleaver/De-interleaver core. The output of the core shows the same data but in an interleaved (or de-interleaved) manner. Alias signals are used to decode the AXI4 channels and allow easy viewing of the input and output data. The test bench does not manipulate `aresetn` or `aclken` even if they are enabled.

The operations performed by the demonstration test bench are appropriate for the configuration of the generated core, and are a subset of the following operations:

1. Sends control information to the core if relevant. The first control word is sent after the data, showing the Control Channel blocking the Data Input Channel. The second control word is sent immediately after the first, showing the Control Channel not blocking the Data Input Channel.
2. Sends the first block of symbols to the Symbol Interleaver/De-interleaver core with no waitstates.
3. Consumes the first block of symbols from the core with no waitstates
4. Sends the second block of symbols to the Symbol Interleaver/De-interleaver core with waitstates. The upstream master adds random waitstates to the input symbols by de-asserting `s_axis_data_tvalid`.
5. Consumes data from the core with waitstates (waitstates only when HAS_DOUT_TREADY = TRUE). The downstream slave adds random waitstates to the output symbols by de-asserting `m_axis_data_tready`.

Customizing the Demonstration Test Bench

It is possible to modify the demonstration test bench to drive the core's inputs with different data or to perform different operations. The stimuli is configured in the 'proc_stimuli_manager' process using configuration objects which are then used by other processes to control the AXI4-Stream channels. The data is sent incrementally starting from 1 for each block. This is hardwired in the 'proc_usdm' process (Upstream Data Master) but is can be changed. The clock frequency of the core can be modified by changing the CLK_PERIOD constant.

Migrating and Upgrading and Updating

This appendix contains information about migrating a design from ISE® to the Vivado® Design Suite, and for upgrading to a more recent version of the IP core. For customers upgrading in the Vivado Design Suite, important details (where applicable) about any port changes and other impact to user logic are included.

This appendix contains information about migrating a design from the ISE® Design Suite to the Vivado® Design Suite, and for upgrading to a more recent version of the IP core. For customers upgrading in the Vivado Design Suite, important details (where applicable) about any port changes and other impact to user logic are included.

Migrating to the Vivado Design Suite

For information about migrating to the Vivado Design Suite, see the *ISE to Vivado Design Suite Migration Guide* (UG911) [Ref X].

Upgrading in the Vivado Design Suite

This section provides information about any changes to the user logic or port designations that take place when you upgrade to a more current version of this IP core in the Vivado Design Suite.

Migrating to the Vivado Design Suite

The Vivado Design Suite IP update feature can be used to update an existing file from v6.0, v7.0 and v7.1 to Symbol Interleaver/De-interleaver v8.0. The core can then be regenerated to create a new netlist.

IMPORTANT: *For v6.0 the update mechanism alone does not create a core compatible with v8.0. See Instructions for Minimum Change Migration.*

For information about migrating to the Vivado Design Suite, see the *ISE to Vivado Design Suite Migration Guide* (UG911) [Ref 6].

Upgrading in the Vivado Design Suite

This section provides information about any changes to the user logic or port designations that take place when you upgrade to a more current version of this IP core in the Vivado Design Suite.

Parameter Changes

Symbol Interleaver/De-interleaver v8.0 has parameters additional to v6.0 for AXI-Stream support. Table A-1 shows the changes to the parameters from version 6.0 to version 8.0. There are no parameter, port nor behavior differences between versions 7.0, 7.1 and v8.0.

Table A-1: **Parameter Changes from v6.0 to v8.0**

Version 6.0	Version 8.0	Notes
architecture	architecture	Unchanged
block_end	has_block_end	Rename
block_size_constant_value	block_size_constant_value	Unchanged
block_size_port_width	block_size_port_width	Unchanged
block_size_type	block_size_type	Unchanged
block_size_valid	has_block_size_valid	Rename
block_start	has_block_start	Rename
branch_length_constant	branch_length_constant	Unchanged
branch_length_type	branch_length_type	Unchanged
ce	has_aclken	Rename
coefficient_file	coefficient_file	Unchanged
column_permutations	column_permutations	Unchanged
col_port_width	col_port_width	Unchanged

Table A-1: **Parameter Changes from v6.0 to v8.0 *(Cont'd)***

Version 6.0	Version 8.0	Notes
col_sel_valid	has_col_sel_valid	Rename
col_valid	has_col_valid	Rename
component_name	component_name	Unchanged
fdo	has_fdo	Rename
memory_style	memory_style	Unchanged
minimum_columns	minimum_columns	Unchanged
minimum_rows	minimum_rows	Unchanged
mode	mode	Unchanged
nd		Obsolete
ndo		Obsolete
number_of_branches	number_of_branches	Unchanged
number_of_columns	number_of_columns	Unchanged
number_of_columns_constant_value	number_of_columns_constant_value	Unchanged
number_of_columns_selectable_value	number_of_columns_selectable_value	Unchanged
number_of_configurations	number_of_configurations	Unchanged
number_of_rows	number_of_rows	Unchanged
number_of_rows_constant_value	number_of_rows_constant_value	Unchanged
number_of_rows_selectable_value	number_of_rows_selectable_value	Unchanged
pipelining	pipelining	Unchanged
rdy	has_rdy	Rename
rfd		Obsolete
rffd		Obsolete
row_permutations	row_permutations	Unchanged
row_port_width	row_port_width	Unchanged
row_sel_valid	has_row_sel_valid	Rename
row_valid	has_row_valid	Rename
sclr	has_aresetn	Rename
symbol_memory	symbol_memory	Unchanged
symbol_width	symbol_width	Unchanged
type	type	Unchanged
	has_dout_tready	New AXI4-Stream option
	external_memory_latency	New Option. Valid range from 0 to 6

Port Changes

Table A-2 details the changes to port naming, additional or deprecated ports and polarity changes from v6.0 and v7.1 to v8.0. There are no port differences between v7.0, v7.1 and v8.0.

Table A-2: **Port Changes from Version 6.0 and v7.1 to Version 8.0**

Version 6.0	Version 8.0	Notes
CLK	ACLK	Rename only
CE	ALCKEN	Rename only
SCLR	ARESETN	Renamed. Polarity change (now active-Low). Minimum length now two clock cycles.
DIN	S_AXIS_DATA_TDATA	Renamed. See Data Input Channel.
ND	S_AXIS_DATA_TVALID	Renamed
RFD	S_AXIS_DATA_TREADY	No longer optional
FD		Removed
RFFD		Removed
ROW	S_AXIS_CTRL_TDATA	Renamed. See Control Channel.
ROW_SEL	S_AXIS_CTRL_TDATA	Renamed. See Control Channel.
COL	S_AXIS_CTRL_TDATA	Renamed. See Control Channel.
COL_SEL	S_AXIS_CTRL_TDATA	Renamed. See Control Channel.
BLOCK_SIZE	S_AXIS_CTRL_TDATA	Renamed. See Control Channel.
CONFIG_SEL	S_AXIS_CTRL_TDATA	Renamed. See Control Channel.
NEW_CONFIG	S_AXIS_CTRL_TVALID	Renamed.
	S_AXIS_CTRL_TREADY	New signal
DOUT	M_AXIS_DATA_TDATA	Renamed. See Data Output Channel.
NDO	M_AXIS_DATA_TVALID	Renamed.
	M_AXIS_DATA_TREADY	New signal. Optional signal
RDY	M_AXIS_DATA_TUSER in Forney mode. M_AXIS_DATA_TVALID in Rectangular mode.	Renamed. See Data Output Channel.
FDO	M_AXIS_DATA_TUSER	Renamed. See Data Output Channel.
BLOCK_START	M_AXIS_DATA_TUSER	Renamed. See Data Output Channel.
BLOCK_END	M_AXIS_DATA_TUSER	Renamed. See Data Output Channel.
RD_DATA	RD_DATA	No change
RD_EN	RD_EN	No change
WR_EN	WR_EN	No change
RD_ADDR	RD_ADDR	No change

Table A-2: **Port Changes from Version 6.0 and v7.1 to Version 8.0 *(Cont'd)***

Version 6.0	Version 8.0	Notes
WR_ADDR	WR_ADDR	No change
WR_DATA	WR_DATA	No change
ROW_VALID	EVENT_ROW_VALID	Rename
COL_VALID	EVENT_COL_VALID	Rename
ROW_SEL_VALID	EVENT_ROW_SEL_VALID	Rename
COL_SEL_VALID	EVENT_COL_SEL_VALID	Rename
BLOCK_SIZE_VALID	EVENT_BLOCK_SIZE_VALID	Rename
	EVENT_TLAST_UNEXPECTED	New signal
	EVENT_TLAST_MISSING	New signal
	EVENT_HALTED	New signal

Latency Changes

There are no latency changes between v7.0, v7.1 and v8.0.

The latency of Symbol Interleaver/De-interleaver v8.0 is different compared to v6.0 in general. The update process cannot account for this and guarantee equivalent performance.

HAS_DOUT_TREADY = 0

The latency of the core increases by 1 clock cycle to that of the equivalent configuration of v6.0.

HAS_DOUT_TREADY = 1

The latency of the core is variable, so that only the minimum possible latency can be determined. The latency is a minimum of 3 cycles longer than for the equivalent configuration of v6.0. The update process cannot account for this and guarantee equivalent performance.

Instructions for Minimum Change Migration

Configuring the Symbol Interleaver/De-interleaver v8.0 to most closely mimic the behavior of v6.0 depends on the features used in v6.0. The first step is to turn off the TREADY on the Output Data Channel (set `has_dout_tready` = FALSE). Then:

- Handle newly mandatory features
  - If you use external memory and have inserted registers between that and the core, set `external_memory_latency` to the appropriate value
  - ND (now `s_axis_data_tvalid`) is no longer optional and has to be driven to 1 to pass symbol data to the core.

- RFD (now `s_axis_data_tready`) is no longer optional and has to be used in the transfer of data to the core. Asserting `s_axis_data_tvalid` is not enough to transfer symbols to the core. If `s_axis_data_tready` is 0, these symbols are ignored
- RDY (now `m_axis_data_tvalid`) in Rectangular mode is no longer optional. It must be used to control the transfer of data from the core. It is no longer completely possible to just rely on latency to calculate when symbols appear.
- NDO (now `m_axis_data_tvalid`) in Forney mode is no longer optional. It must be used to control the transfer of data from the core. It is no longer completely possible to just rely on latency to calculate when symbols appear.

- Handle Changed Features
  - If you previously had SCLR set to TRUE then remember that the reset pulse is now active-Low and must be a minimum of two clock cycles long.
  - If you previously had SCLR set to TRUE and CE set to TRUE, note that reset now overrides clock enable. In v6.0 asserting both at the same time would not result in a reset. In v8.0, asserting both at the same time results in a reset.
  - Illegal blocks now are always aborted in Rectangular mode. When the block is recognized as illegal, the core treats subsequent symbols as belonging to a new block. This can lead to synchronisation issues between the core and the system.

RECOMMENDED: *Xilinx recommends either not injecting illegal blocks, or resetting the core using* `aresetn` *when an* `event_*_valid` *signal is seen as 0.*

- Handle Obsolete features
  - FD is no longer available. The core starts a block when:
    - The first symbol is seen after a reset.
    - When the first symbol is seen after the end of a block in Rectangular mode.
    - When the first symbol is seen after the end of a block in Forney mode. This is when the commutator reaches branch 0 after `s_axis_data_tlast` has been asserted
  - FD abort is no longer available. Enough symbol data has to be supplied to bring a block to a natural conclusion, or `aresetn` has to be used to reset the core.

In Forney mode, this means blocks must now be an integer multiple of the number of branches in use.

Functionality Changes

There are no functionality changes from version 7.0 and 7.1 to v8.0. The changes in functionality from version 6.0 to version 8.0 are that the interfaces have been changed to be AXI4-Stream.

Simulation

Starting with Interleaver/De-Interleaver v8.0 (2013.3 version), behavioral simulation models have been replaced with IEEE P1735 Encrypted VHDL. The resulting model is bit and cycle accurate with the final netlist. For more information on simulation, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 7].

Debugging

This appendix includes details about resources available on the Xilinx Support website and debugging tools. In addition, this appendix provides a step-by-step debugging process and a flow diagram to guide you through debugging the Interleaver/De-Interleaver core.

Finding Help on Xilinx.com

To help in the design and debug process when using the Interleaver/De-Interleaver, the Xilinx Support web page contains key resources such as product documentation, release notes, answer records, information about known issues, and links for obtaining further product support.

Documentation

This product guide is the main document associated with the Interleaver/De-Interleaver. This guide, along with documentation related to all products that aid in the design process, can be found on the Xilinx Support web page or by using the Xilinx Documentation Navigator.

Download the Xilinx Documentation Navigator from the Downloads page. For more information about this tool and the features available, open the online help after installation.

Answer Records

Answer Records include information about commonly encountered problems, helpful information on how to resolve these problems, and any known issues with a Xilinx product. Answer Records are created and maintained daily ensuring that users have access to the most accurate information available.

Answer Records for this core are listed below, and can be located by using the Search Support box on the main Xilinx support web page. To maximize your search results, use proper keywords such as

- Product name
- Tool message(s)

- Summary of the issue encountered

A filter search is available after results are returned to further target the results.

Master Answer Record for the Interleaver/De-Interleaver

AR 54503

Technical Support

Xilinx provides technical support in the Xilinx Support web page for this LogiCORE™ IP product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support if you do any of the following:

- Implement the solution in devices that are not defined in the documentation.
- Customize the solution beyond that allowed in the product documentation.
- Change any section of the design labeled DO NOT MODIFY.

To contact Xilinx Technical Support, navigate to the Xilinx Support web page.

Debug Tools

There are tools available to address Interleaver/De-Interleaver design issues. It is important to know which tools are useful for debugging various situations.

Test Bench

The Interleaver/De-Interleaver is delivered with an test bench that can be simulated. Information about the test bench can be found in *Chapter 5, Example Design*.

Vivado Design Suite Debug Feature

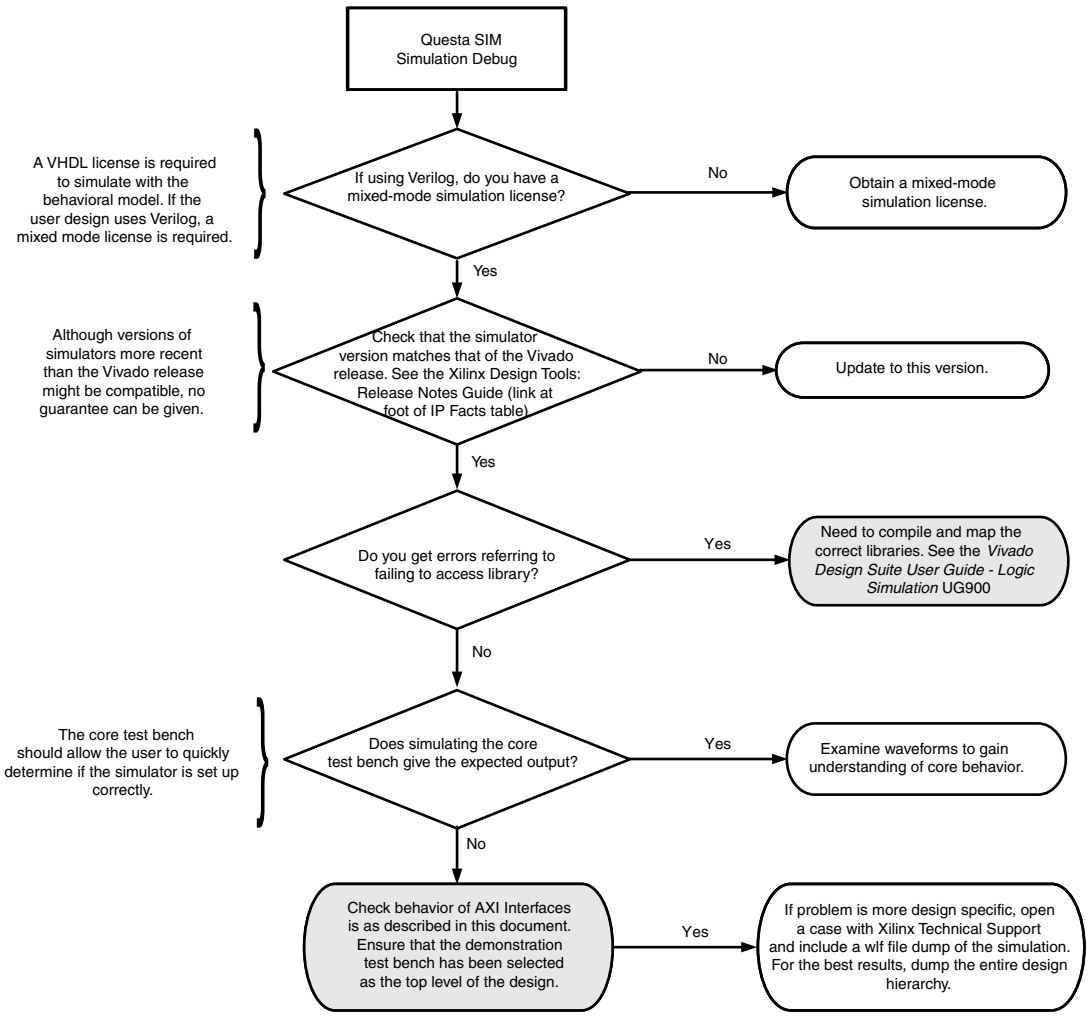
The Vivado® Design Suite debug feature inserts logic analyzer and virtual I/O cores directly into your design. The debug feature also allows you to set trigger conditions to capture application and integrated block port signals in hardware. Captured signals can then be analyzed. This feature represents the functionality in the Vivado IDE that is used for logic debugging and validation of a design running in Xilinx FPGAs in hardware.

The Vivado logic analyzer is used to interact with the logic debug LogiCORE IP cores, including:

- ILA 2.0 (and later versions)
- VIO 2.0 (and later versions)

Simulation Debug

The simulation debug flow for Questa® SIM is shown in Figure B-1. A similar approach can be used with other simulators.



Interface Debug

AXI4-Stream Interfaces

If data is not being transmitted or received, check the following conditions:

- If transmit `<interface_name>_tready` is stuck Low following the `<interface_name>_tvalid` input being asserted, the core cannot send data.

- If the receive <*interface_name*>_tvalid is stuck Low, the core is not receiving data.
- Check that the ACLK inputs are connected and toggling.
- Check that the AXI4-Stream waveforms are being followed (See Figure 3-1, Figure 3-9 and Figure 3-10).
- Check core configuration.

Appendix C

Additional Resources and Legal Notices

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see Xilinx Support.

References

These documents provide supplemental material useful with this user guide:

1. *Xilinx AXI Reference Guide* (UG761)
2. AMBA® AXI4-Stream Protocol Specification (ARM IHI 0051A)
3. *Vivado Design Suite User Guide: Designing IP Subsystems using IP integrator* (UG994)
4. *Vivado Design Suite User Guide: Designing with IP* (UG896)
5. *Vivado Design Suite User Guide: Getting Started* (UG910)
6. *Vivado Design Suite User Guide Migration Methodology Guide* (UG911)
7. *Vivado Design Suite User Guide: Logic Simulation* (UG900)
8. *Vivado Design Suite User Guide, Designing with IP* (UG896)

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
11/18/2015	8.0	Added support for UltraScale+ families.
04/02/2014	8.0	Updated Resource Utilization and Performance information.

Date	Version	Revision
12/18/2013	8.0	• Added UltraScale™ architecture support information. • Added Simulation, Synthesis, Example Design, and Test Bench chapters. • Updated Migrating appendix.
10/02/2013	8.0	Minor updates to IP Facts table and Migrating appendix. Document version number advanced to match the core version number.
03/20/2013	2.0	Updated core version for Vivado. Updated Debugging appendix.
07/25/2012	1.0	This product guide replaces DS861.

Please Read: Important Legal Notices