LogiCORE IP Linear Algebra Toolkit v2.0

Product Guide for Vivado Design Suite

PG131 March 20, 2013

Table of Contents

Introduction

The Xilinx LogiCORE™ IP Linear Algebra Toolkit (LAT) v2.0 implements matrix-matrix addition, matrix-matrix subtraction, matrix-matrix multiplication, and matrix-scalar multiplication. It provides a high degree of flexibility, allowing the IP generated to be tailored to a vast range of end user applications. All functions generated are compliant with the AXI4-Stream specification, facilitating system integration.

Features

- Drop-in module for Zynq™-7000, Virtex®-7, Kintex™-7, and Artix™-7 devices
- AXI4-Stream Protocol compliant core
- Fully synchronous, single clock domain design with synchronous active-Low reset
- Fixed-point matrix-matrix addition/subtraction, matrix-scalar multiplication, matrix-matrix multiplication
- Configurable matrix dimension up to 32x32
- Real or complex data support
- Two's complement fixed-point maximum of 24-bit input and 48-bit output
- Symmetric, asymmetric, convergent, and truncation options for output rounding with output saturation support
- Clock Enable (CE) input, can pause core and resume on-the-fly by deasserting/reasserting CE
- User selectable folding factor for resource optimization

LogiCORE IP Facts Table	
Core Specifics	
Supported Device Family[1]	Zynq-7000, Virtex-7, Kintex-7, Artix-7
Supported User Interfaces	AXI4-Stream
Resources	See Table 2-4 and Table 2-5
Provided with Core	
Design Files	Encrypted RTL
Example Design	Not Provided
Test Bench	VHDL
Constraints File	Not Provided
Simulation Model	VHDL and Verilog Structural Simulation Model Bit Accurate C Model
Supported S/W Driver[2]	N/A
Tested Design Flows[2]	
Design Entry	Vivado™ Design Suite
Simulation	Mentor Graphics Questa® SIM
Synthesis	Vivado Synthesis
Support	
Provided by Xilinx @ www.xilinx.com/support	

Notes:

1. For a complete list 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

The LAT v2.0 IP core supports fixed-point matrix-matrix addition, subtraction, matrix-scalar multiplication, and matrix-matrix multiplication. These operations are represented by the following equations.

Table 1-1: **Matrix-Matrix Addition Subtraction, Matrix-Scalar Multiplication, Matrix-Matrix Multiplication Equations**

Matrix-Matrix Addition Subtraction	$C_{M \times N} = A_{M \times N} \pm B_{M \times N}$ $C_{M \times N}(i,j) = A_{M \times N}(i,j) \pm B_{M \times N}(i,j)$
Matrix-Scalar Multiplication	$C_{M \times N} = A_{M \times N} \times b$ $C_{M \times N}(i,j) = A_{M \times N}(i,j) \times b$
Matrix-Matrix Multiplication	$C_{M \times N} = A_{M \times L} \times B_{L \times N}$ $C_{M \times N}(i,j) = \sum_{k=1}^{L} A_{M \times L}(i,k) \times B_{L \times N}(k,j)$

These operations are four functional modes (`OP_MODE`) of the core. A and B (becomes scalar input b for matrix-scalar multiplication) are two input matrices, and C is the output matrix. Matrix dimensions are configurable through three dimension parameters M, N, and L. After `OP_MODE` and dimension parameters are selected, individual matrix dimensions are decided automatically as shown in Table 1-1. The next set of parameters is the data width of individual matrices, `A_WIDTH`, `B_WIDTH` and `C_WIDTH`, representing precision of elements of matrices A, B, and C, respectively. There are two AXI4-Stream compliant slave ports for input matrices and one master port for the output matrix C (port A, port B, and port C, respectively) as shown in the block diagram of the core in Figure 2-1. Data bus (TDATA) widths for all three ports are decided internally based on the data width, matrix dimension, and folding factor. Folding factor (F) is a user configurable parameter for deciding the resource reuse (mostly DSP48E1 slices) of the core. Folding factor also dictates the processing latency and streaming latency (or conversion rate) of the mode.

As this is a fully real-time streaming core, input matrices must be fed to the core in F (or less) contiguous cycles of the input clock and this constraint applies to both A and B matrices. The next batch of matrices can be fed in the immediate following clock cycle making the core fully streaming capable. Similarly, the output matrix C is made available at the output port over F (or less) contiguous cycles. The core can be paused and resumed dynamically using a clock enable (CE, active-High) signal at the input of the core. This feature can be used for flow control if needed; the CE can be tied to HIGH if not necessary. The two I/O modes, serial and parallel, are for data entry and exit to and from the core. In serial mode, matrix elements are entered serially, element-by-element and the output matrix exits the core in similar fashion. While in parallel mode, matrix elements are packed and a parallel bus is formed for each of the three ports. Packing is row-wise, the first element of the row goes to the least significant position (LSP), followed by the next element until the bus width is fully populated. Consequently, a fully packed port A (in parallel mode) has the following packing ([MSP to LSP]):

$$[A_{M\times N}(2,3)\ A_{M\times N}(2,2)\ A_{M\times N}(2,1)\ A_{M\times N}(1,5)\ A_{M\times N}(1,4)\ A_{M\times N}(1,3)\ A_{M\times N}(1,2)\ A_{M\times N}(1,1)]$$

A detailed description of the four functional modes is provided in the following sections.

Matrix-Matrix Addition and Subtraction

In this mode, matrices A, B, and C have the same dimension, MxN (M is the number of rows and N is the number of columns), and the same number of elements (MN). The data widths (that is, precision) A_WIDTH, B_WIDTH, and C_WIDTH are abbreviated as A_w, B_w, and C_w, respectively. The elements of the corresponding matrix entering (or exiting for C) in one clock cycle are NE_A, NE_B, and NE_C. For this mode, NE_A=NE_B =NE_C and computed as:

$$NE_{A/B/C} = ceil\left(\frac{M\times N}{F}\right) \qquad \text{Equation 1-1}$$

in parallel I/O mode, which indicates as many elements are packed to form the TDATA bus (of the corresponding port) as elements (adder/subtracter) required for the chosen mode. The selection of $NE_{A/B/C}$ occurs inside the core when OP_MODE, dimensions, data width, folding factor, and I/O modes are selected. The core selects the optimum number of elements based on F and the real-time streaming requirement specifying that matrix elements enter (and exit) in F clock cycles or less.

TDATA bus widths for three ports are computed accordingly after $NE_{A/B/C}$ are computed. S_AXIS_A_TDATA, S_AXIS_B_TDATA and M_AXIS_C_TDATA (as shown in Figure 2-1) bus widths are denoted as $ABUS_w$, $BBUS_w$ and $CBUS_w$. The core computes these bus widths as shown in the following equations:

$$BBUS_w = NE_B \times (1 + CMPLX) \times ceil8(B_w) \qquad \text{Equation 1-2}$$

$$ABUS_w = NE_A \times (1 + CMPLX) \times ceil8(A_w) \qquad \text{Equation 1-3}$$

$$CBUS_w = NE_C \times (1 + CMPLX) \times ceil8(C_w) \qquad \text{Equation 1-4}$$

Where, CMPLX is the data type parameter to be configured to zero for real data type and to one for complex data type and *ceil8*(.) is the operator for ceiling (equal to or greater) to an integer divisible by 8. The number of clock cycles required for matrix A/B/C is determined by the following equation (meeting the real-time streaming requirement):

$$K_{A/B/C} = ceil\left(\frac{M \times N}{NE_{A/B/C}}\right) = ceil\left(\frac{M \times N}{ceil\left(\frac{M \times N}{F}\right)}\right) \le F \qquad \text{Equation 1-5}$$

Serial I/O mode is a special case of parallel I/O mode with $NE_A=NE_B=NE_C=1$, F=MN and $K_{A/B/C}$=MN, and all previous equations are true with bus widths scaling down accordingly. While entering the data to ports A and B, corresponding port TVALID is asserted High for $K_{A/B/C}$ number of contiguous clock cycles. Similarly, on the output port C, TVALID is asserted High by the core for the same number of contiguous clock cycles when the output is available. Also, TREADY is asserted HIGH (by external AXI4 slave) for these many cycles (at least) for the proper functioning of the core in this mode.

Matrix-Scalar Multiplication

This mode is similar to the matrix-matrix addition mode with the exception of port B where input is scalar; consequently, $NE_B=1$ and $K_B=1$. The rest of the parameters for ports A and C remain the same as those for addition mode. In serial mode, port B parameters are unchanged ($NE_B=1$ and $K_B=1$), while port A and B parameters scale down as in the addition case with $NE_{A/C}=1$ and $K_{A/C}$=MN.

Matrix-Matrix Multiplication

In this mode, unlike addition mode, the dimensions of matrices A, B and C can be dissimilar and folding factor F scales matrix rows only, giving matrix elements per cycle as:

$$NE_A = ceil\left(\frac{M}{F}\right) \times L \qquad \text{Equation 1-6}$$

$$NE_B = ceil\left(\frac{L}{F}\right) \times N \qquad \text{Equation 1-7}$$

$$NE_C = ceil\left(\frac{M}{F}\right) \times N \qquad \text{Equation 1-8}$$

The bus width equations (Equation 1-2, Equation 1-3, and Equation 1-4) hold true for this mode as well. However, the number of clock cycles going in to matrices A and B and out of matrix C are changed as follows:

$$K_A = ceil\left(\frac{M \times L}{NE_A}\right) = ceil\left(\frac{M \times L}{ceil\left(\frac{M}{F}\right) \times L}\right) = ceil\left(\frac{M}{ceil\left(\frac{M}{F}\right)}\right) \le F \qquad \textit{Equation 1-9}$$

$$K_B = ceil\left(\frac{L \times N}{NE_B}\right) = ceil\left(\frac{L \times N}{ceil\left(\frac{L}{F}\right) \times N}\right) = ceil\left(\frac{L}{ceil\left(\frac{L}{F}\right)}\right) \le F \qquad \textit{Equation 1-10}$$

$$K_C = ceil\left(\frac{M \times N}{NE_C}\right) = ceil\left(\frac{M \times N}{ceil\left(\frac{M}{F}\right) \times N}\right) = ceil\left(\frac{M}{ceil\left(\frac{M}{F}\right)}\right) \le F \qquad \textit{Equation 1-11}$$

This choice of scaling matches the rates of A and C ports, that is K_A=K_C (only in parallel mode). Serial I/O mode is a bit different in this operation. While elements per cycle in serial mode are equal (NE_A=NE_B =NE_C =1), the number of cycles needed for input to the matrix and the folding factor change according to the matrix dimensions. Consequently, K_A=ML, K_B=LN and K_C =MN and the folding factor is as follows:

$$F = max(K_A, K_B, K_C) \qquad \textit{Equation 1-12}$$

In the previous four functional modes, when K_A is less than F, no data should be sent on the cycles beyond K_A in port A. The core also deasserts `S_AXIS_A_TREADY` beyond these K_A cycles. Similarly, when K_B is less than F, no data should be sent on the cycles beyond K_B. The core also deasserts `S_AXIS_B_TREADY` beyond these cycles and when K_C is less than F, no data will be sent on the cycles beyond K_C. The core will also deassert `M_AXIS_C_TVALID` beyond these cycles. Folding factor support is 1-M for parallel I/O mode. However, other folding factors 1-N can be obtained (if needed) by transposing the input matrices (C' = B'A'). Consequently, output matrix C' will also be in the transposed form.

Symmetric rounding to zero, symmetric rounding to infinity, asymmetric rounding to zero, asymmetric rounding to infinity, convergent rounding to even, convergent rounding to odd and truncation modes are supported for rounding. Saturation is performed by default. All internal computations are performed in full precision and rounding-saturation is performed only at the last stage of computation. Data grows through the internal computing stages where full data precision is maintained until the last stage assuming worst case data growth. This is handled intelligently without requiring any extra computing resources. For example, the worst case output precision $C_{w,max}$ for different `OP_MODES` (given input precisions A_w,B_w) are as follows:

`OP_MODE`: matrix-matrix addition/subtraction,

$$C_{w,\,max} = max(A_w, B_w) + 1 \qquad \text{Equation 1-13}$$

`OP_MODE`: matrix-scalar multiplication,

$$C_{w,\,max} = A_w + B_w + CMPLX \qquad \text{Equation 1-14}$$

`OP_MODE`: matrix-matrix multiplication,

$$C_{w,max} = A_w + B_w + ceil(log2(L)) + CMPLX \qquad \text{Equation 1-15}$$

In these four `OP_MODES`, output is allowed to grow up to $C_{w,max}$ (as shown previously) before rounding to the user selected C_w at the last stage of computing.

In matrix-matrix addition-subtraction mode, input data widths (A_w and B_w) can be up to a maximum of 24 bits. You have the flexibility to choose an implementation based on the FPGA logic or DSP48E1 slices. For the complex input case, when DSP slice based implementation is used, each DSP slice is configured in the SIMD mode to optimize the resource usage. Rounding and saturation are implemented on the FPGA logic.

In matrix-scalar multiplication mode, A_w can be up to 24 bits and B_w can be up to 18 bits. This limitation is from the DSP slice, which uses an internal 25x18 multiplier. The implementation is always based on DSP slices. Rounding and saturation are implemented on the FPGA logic.

In matrix-matrix multiplication mode, A_w and B_w can be up to 24 bits. However, for any given implementation, the choice of A_w and B_w should be such that unit multiplication can be accommodated within one DSP slice, which uses an internal 25x18 multiplier. Consequently, if A_w is greater than 18 bits, B_w should be less than or equal to 18 bits and vice versa. The implementation is always based on DSP48E1 slices. Rounding and saturation are implemented on the FPGA logic. Output precision C_w can be up to 48 bits. You have the flexibility to choose any C_w bits over the range [0, $C_{w,max}$-1] for output using the parameter LSB. C_w can be chosen between [2, $C_{w,max}$]. The parameter LSB indicates the least significant bit location of the output relative to the internal full precision output value, as shown in Figure 1-1. C_w number of bits, starting from the LSB are sent to the output. As a result, given C_w, LSB can vary from [0, $C_{w,max}$ - C_w].

You also have the flexibility to choose a folding factor for a given matrix operation to optimize resource utilization when operating in the parallel I/O mode. The folding factors supported are [1 to M, N and MN] for matrix addition, subtraction and scalar multiplication modes, while in matrix multiplication mode, supported folding factors are [1 to M]. However, the folding factor should be selected so it does not violate the AXI4 port TDATA width limitations. In this version, maximum port width is set to 1024 bits for all the three ports (ports A, B and C). In serial I/O mode, you do not have the flexibility to choose the folding factor, which scales to MN for matrix addition, subtraction and scalar multiplication modes, while for matrix multiplication mode, it is determined by Equation 1-12.

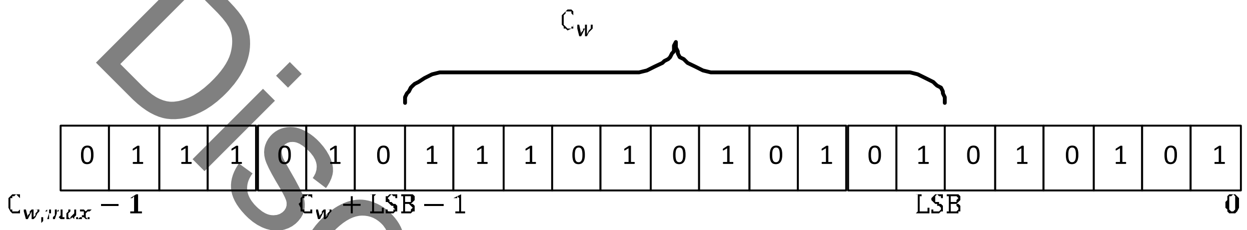


Figure 1-1: **Output Precision Format**

Feature Summary

Linear Algebra Toolkit v2.0 is a Vivado™ IP catalog core which implements matrix-matrix addition, matrix-matrix subtraction, matrix-matrix multiplication, and matrix-scalar multiplication. The core incorporates a high degree of flexibility by implementing matrix operations on a foldable and scalable systolic structure supporting a wide range of configurations. Salient features of this core are:

- Drop-in module for Zynq™-7000, Virtex®-7, Kintex™-7, and Artix™-7 devices
- AXI4-Stream Protocol compliant core
- Fully synchronous, single clock domain design with synchronous active-Low reset
- Fixed-point matrix-matrix addition/subtraction, matrix-scalar multiplication, matrix-matrix multiplication
- Configurable matrix dimension up to 32x32
- Real or complex data support
- Two's complement fixed-point maximum of 24-bit input and 48-bit output
- Symmetric, asymmetric, convergent, and truncation options for output rounding with output saturation support
- Clock Enable (CE) input, can pause core and resume on-the-fly by deasserting/reasserting CE
- User selectable folding factor for resource optimization

- Data I/O Support: serial or parallel
- Uses SIMD mode of DSP48E1 to implement complex addition on a single slice
- DSP slice or LUT based implementation for matrix-matrix addition/subtraction
- Folding factors supported for matrix size MxN are: 1, 2, 3, ..., M, N, MN
- Bit accurate C Model available
- Customer demonstration test bench
- Folding factors supported for matrix multiplication $C_{MxN} = A_{MxL} \times B_{LxN}$ are: 1, 2, 3, ..., M, MN
- Bit accurate C Model available
- Customer demonstration test bench

Applications

The following applications are supported.

- LAT base functions (addition/subtraction, scalar and matrix multiplications) IP core ideally suited for high performance streaming matrix-data processing, various types of digital signal processing needs on 1-D and 2-D data. This highly scalable and configurable core provides a wide choice of cost-performance trade-off for various types of Radar, Sonar and other horizontal signal and data processing requirements.
- Provides high-performance and well optimized base functions as building blocks for MIMO (Multiple Input Multiple Output) transmit-receive processing in wireless communication system.

Licensing and Ordering Information

This Xilinx LogiCORE™ IP module is provided at no additional cost with the Xilinx Vivado™ Design Suite under the terms of the Xilinx End User License. Information about this and other Xilinx LogiCORE IP modules is available at the Xilinx Intellectual Property page. For information about pricing and availability of other Xilinx LogiCORE IP modules and tools, contact your local Xilinx sales representative.

Chapter 2

Product Specification

Standards

The Linear Algebra Toolkit v2.0 core adheres to the AMBA® AXI4-Stream protocol for the data interface.

Latency and Performance

This section describes latency, performance and resource utilization of the core in various modes and configurations. Because the LAT core is fully real-time streaming capable, the streaming latency (minimum lag in clock cycles between two successive matrices) is always the same as the folding factor F cycles. However, initial latency of the core is a function of various core parameters.

Latency

Let T_D (in clock cycles) denote the total first-in-first-out (initial latency) delay of the core. This is the latency from the first data input to the core to the first output observed at the output of the core. The initial latency T_D has two components, one, delay (in clock cycles) T_{core_delay} of the functional-mode core itself (based on the choice of M, N and L) and the other, a fixed part (based on routing and registering stages of the core) T_f. Thus, total latency can be expressed as:

$$T_D = T_{core\_delay} + T_f$$ *Equation 2-1*

Functional mode-wise break-up of initial latency follows:

- Matrix-Matrix Addition/Subtraction

  In this mode, core delay is constant and does not scale with matrix dimension, T_{core_delay}=2 and the fixed delay T_f is between [0,4] based on implementation.

- Matrix-Scalar Multiplication

For this op-mode, the core delay parameter is a function of the number of elements of the A matrix input per cycle (NE_A) (as shown in the following table) and the fixed delay T_f is between [0,4] as in the case of addition/subtraction.

Table 2-1: **Matrix-Scalar Mode Core Delay**

NE_A	T_{core_delay}
≤ 4	4 + 2 x CMPLX
≤ 8	5 + 2 x CMPLX
≤ 16	6 + 2 x CMPLX
≤ 32	7 + 2 x CMPLX
> 32	8 + 2 x CMPLX

- Matrix-Matrix Multiplication

  In this op-mode, the folding factor directly affects T_{core_delay} because the core waits for the input matrices to get filled up completely before starting to process them. Consequently, for parallel I/O:

$$T_{core\_delay} = F + (CMPLX + 1) \times L + N + ceil\left(\frac{M}{F}\right) \quad \textit{Equation 2-2}$$

  and for the serial I/O case, this equation is simplified to:

$$T_{core\_delay} = F + (CMPLX + 1) \times L \quad \textit{Equation 2-3}$$

  The maximum value of T_f is 11 and 17 clock cycles in parallel and serial modes, respectively.

General Notes on Characterization

While obtaining the maximum frequency numbers for different configurations and op-modes for the core, the general guideline is to use default settings of Vivado™ Design Suite Synthesis and Implementation tools to make the results independent of tool versions. The specific settings used for obtaining resource utilization and maximum frequency of operation are shown in Table 2-2.

Table 2-2: **Tool Options For Core Performance**

Tool	Options
Vivado Synthesis	Specify "-mode out_of_context" under More Options in Vivado Synthesis settings.
Vivado Implementation	Enable option - phys_opt_design

The maximum achievable clock frequency and the resource counts can be affected by other tool options, additional logic in the device, using a different version of Xilinx tools, and other factors.

IMPORTANT: *Clock frequency does not take clock jitter into account and should be derated by an amount appropriate to the clock source jitter specification. Relatively higher matrix dimensions with low folding factors or high values of I/O precision (bit widths) results in increased routing congestion especially for the case of matrix-matrix multiplication. This can result in poorer maximum achievable frequency.*

Rounding Modes

Characterization was performed by configuring the core to the default "convergent rounding to even" mode.

Memory Usage

FIFOs are used in matrix-matrix multiplication mode, which can either be mapped to Block RAM or Distributed RAM. Selection of Distributed RAM mode realizes FIFOs using LUT/FFs on the FPGA logic and gives better maximum frequency, while choice of Block RAM provides better resource utilization at the cost of some reduction in maximum achievable frequency.

Resource Utilization

Resources required for the Linear Algebra toolkit core have been estimated for the Virtex®-7 and Kintex™-7 FPGAs (Table 2-4 and Table 2-5). These values were generated using the Vivado IP Catalog. The maximum frequency of operation is obtained from post PAR timing report while the resource usage is obtained from post placement utilization report.

This section provides data on the timing performance and resource utilization of the core. Performance has been obtained on one representative device from the Virtex-7 and Kintex-7 families of FPGAs. The following table lists the devices used for characterization.

Table 2-3: **Devices Used For Characterization**

Virtex-7	Kintex-7
xc7vx330t	xc7k70t

All devices were used with speed grade -1.

Table 2-4: **Virtex-7 Maximum Frequency and Resource Utilization, Device: xc7vx330t-ffg41157-1, Speed File: -1 ADVANCED 1.08a 2013-02-09**

OPMODE	Configuration								DSP48E1 Slices	Block RAMs	Slices	LUT-FF Pairs	Fmax (MHZ)	Fmax[2] (MHz)
	M	L	N	Folding Factor	Data Type	A Width	B Width	C Width						
Matrix-Matrix Addition (DSP48E1 based)	6	NA	5	1	Complex	16	16	16	30	0	1088	1183/4216	470	470
	8	NA	5	2	Complex	24	24	24	20	0	1063	1150/4256	470	470
	8	NA	10	4	Complex	24	24	24	20	0	1067	1174/4262	470	470
	8	NA	10	80[1]	Complex	24	24	24	1	0	78	85/244	470	470
Matrix-Matrix Addition (LUT based)	6	NA	5	1	Complex	16	16	16	0	0	1636	2162/7156	470	470
	8	NA	5	2	Complex	24	24	24	0	0	1598	2076/7016	470	470
	8	NA	10	4	Complex	24	24	24	0	0	1476	2076/7022	470	470
	8	NA	10	80[1]	Complex	24	24	24	0	0	91	137/382	470	470
Matrix-Scalar Multiplication	2	NA	5	1	Complex	16	18	16	40	0	639	454/2657	470	470
	4	NA	5	2	Complex	24	18	24	40	0	819	602/3620	470	470
	8	NA	10	4	Complex	24	18	24	80	0	1943	1224/8178	470	470
	8	NA	10	80[1]	Complex	24	18	24	4	0	82	99/312	470	470
Matrix-Matrix Multiplication	8	4	1	1	Real	16	16	16	32	0	1171	1647/4088	470	470
	8	8	1	2	Real	16	16	16	32	0	983	1596/3662	460	470
	4	4	4	4	Real	16	16	16	16	0	509	909/1783	470	470
	8	8	8	64[1]	Real	16	16	16	8	0	534	1478/1593	470	470
	8	4	1	8	Complex	16	16	16	16	0	594	1281/2074	470	470
	4	4	4	4	Real	16	16	16	16	24	323	258/1079	400	400

1. Serial I/O
2. Without clock enable

Table 2-5: **Kintex-7 Maximum Frequency and Resource Utilization, Device: xc7k70t-fbg484-1, Speed File: -1 PRODUCTION 1.08 2013-02-09**

OPMODE	Configuration								DSP48E1 Slices	Block RAMs	Slices	LUT-FF Pairs	Fmax (MHZ)	Fmax[2] (MHz)
	M	L	N	Folding Factor	Data Type	A Width	B Width	C Width						
Matrix-Matrix Addition (DSP48E1 based)	6	NA	5	1	Complex	16	16	16	30	0	1054	1176/4216	470	470
	8	NA	5	2	Complex	24	24	24	20	0	1066	1166/4256	470	470
	8	NA	10	4	Complex	24	24	24	20	0	1024	1166/4262	470	470
	8	NA	10	80[1]	Complex	24	24	24	1	0	74	88/244	470	470
Matrix-Matrix Addition (LUT based)	6	NA	5	1	Complex	16	16	16	0	0	1519	2134/7156	470	470
	8	NA	5	2	Complex	24	24	24	0	0	1419	2075/7016	470	470
	8	NA	10	4	Complex	24	24	24	0	0	1383	2082/7022	470	470
	8	NA	10	80[1]	Complex	24	24	24	0	0	91	138/382	470	470
Matrix-Scalar Multiplication	2	NA	5	1	Complex	16	18	16	40	0	684	436/2657	470	470
	4	NA	5	2	Complex	24	18	24	40	0	875	599/3620	470	470
	8	NA	10	4	Complex	24	18	24	80	0	1976	1203/8178	465	470
	8	NA	10	80[1]	Complex	24	18	24	4	0	77	99/312	470	470
Matrix-Matrix Multiplication	8	4	1	1	Real	16	16	16	32	0	1215	1647/4100	445	470
	8	8	1	2	Real	16	16	16	32	0	1086	1596/3662	455	470
	4	4	4	4	Real	16	16	16	16	0	518	909/1783	465	470
	8	8	8	64[1]	Real	16	16	16	8	0	548	1482/1593	455	470
	8	4	1	8	Complex	16	16	16	16	0	576	1281/2074	470	470
	4	4	4	4	Real	16	16	16	16	24	363	259/1079	400	400

1. Serial I/O
2. Without clock enable

Core Symbol and Port Definitions

The block diagram in Table 2-1 provides the input/output signals for the core. Signals are categorized into Control Signals (clock, reset and clock enable), Port A Signals, Port B Signals and Port C Signals. Table 2-6 provides detailed information for the signals.

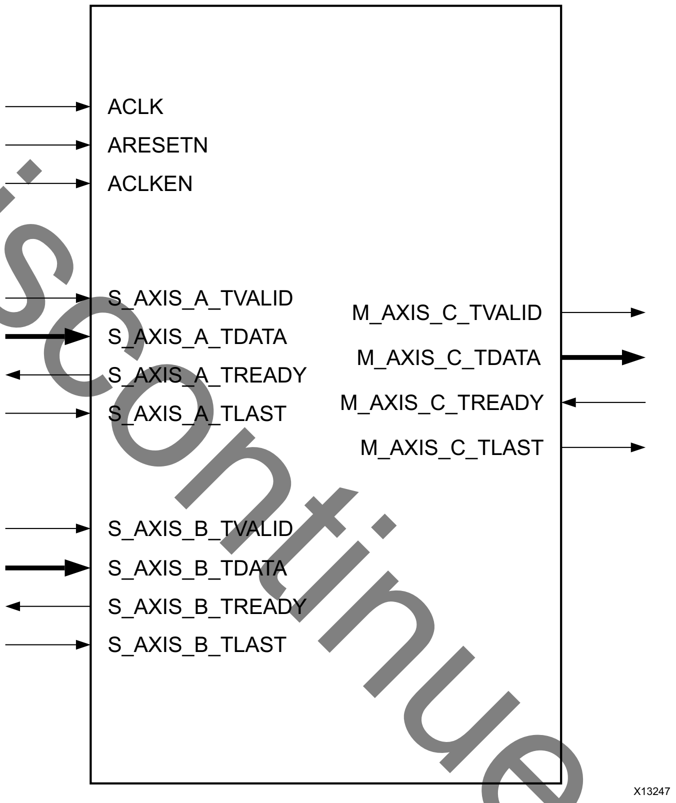


Figure 2-1: **LAT IP Core Block Diagram**

Table 2-6: **LAT Interface Signals**

Signal	Bit Width	I/O	Description
Control Signals			
ACLK	1	I	Global clock for the AXI4 interface. The same clock is used internal to the core as single clock domain fully synchronous design. The whole design operates only at the positive edge of this clock.
ARESETN	1	I	Active-low synchronous reset. When ARESETN is asserted (ARESETN = 0), the interface is forced into the reset state irrespective of ACLKEN. When ARESETN is deasserted, the interface comes out of reset state. ARESETN is synchronous to ACLK. ARESETN has to be asserted for at least 4 clock cycles for the core to get reset. ARESETN takes precedence over ACLKEN.
ACLKEN	1	I	Clock enable for the IP. This is an optional signal. ACLKEN is active-high and is synchronous to ACLK. All activity is suspended when ACLKEN is deasserted. This signal to be used for flow control for the core, which is designed for real time streaming.
Port A Signals (for input Matrix A)			
S_AXIS_A_TVALID	1	I	AXI4 data valid signal for port A. Active-High signal synchronous to ACLK, to be held high for $K_A$contiguous cycles by external master.
S_AXIS_A_TDATA	[0: $ABUS_w$ - 1]	I	Data port for Matrix A (max width 1024 bits). Data comes synchronously w.r.t ACLK over K_A contiguous cycles.
S_AXIS_A_TREADY	1	O	Core ready signal for port A, synchronous to ACLK, stays asserted for K_A contiguous cycles and gets deasserted for F-K_A cycles. Active-High signal.
S_AXIS_A_TLAST	1	I	Indicates the last cycle of K_A contiguous cycles, synchronous to ACLK, gets asserted for the last cycle of K_A contiguous cycle. Active-High signal.
Port B Signals (for input Matrix B)			
S_AXIS_B_TVALID	1	I	AXI4 data valid signal for port B. Active-High signal synchronous to ACLK, to be held high for K_B contiguous cycles by external master.
S_AXIS_B_TDATA	[0: $BBUS_w$-1]	I	Data port for Matrix B or scalar b (max width 1024 bits). Data comes synchronously w.r.t ACLK over K_B contiguous cycles.
S_AXIS_B_TREADY	1	O	Core ready signal for port B, synchronous to ACLK, stays asserted for K_B contiguous cycles and gets deasserted for F-K_B cycles. Active-High signal.
S_AXIS_B_TLAST	1	I	Indicates the last cycle of K_B contiguous cycles, synchronous to ACLK, gets asserted for the last cycle of K_B contiguous cycles. Active-High signal.

Table 2-6: **LAT Interface Signals**

Signal	Bit Width	I/O	Description
Port C Signals (for output Matrix C)			
M_AXIS_C_TVALID	1	O	AXI4 data valid signal for port C. Active-High signal synchronous to ACLK, held high for K_C contiguous cycles by the core.
M_AXIS_C_TDATA	[0: $CBUS_w$-1]	O	Data port for Matrix C (max width 1024 bits). Data comes out synchronously w.r.t ACLK over K_C contiguous cycles from the core after initial delay of T_D cycles.
M_AXIS_C_TREADY	1	I	AXI4 streaming ready signal for port C from external slave, synchronous to ACLK, must stay asserted for K_C (at least) contiguous cycles from the beginning of M_AXIS_C_TVALID assertion. Active-High signal.
M_AXIS_C_TLAST	1	O	Indicates the last cycle of K_C contiguous cycles, synchronous to ACLK, gets asserted for the last cycle of K_C contiguous cycles. Active-High signal.

Designing with the Core

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

Using the LAT Core

The Vivado™ Design Suite performs error-checking on all input parameters. Resource estimation and implementation details 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 tools user documentation.

Simulation Model

The core has two options for simulation models:

- VHDL UNISIM structural model
- Verilog UNISIM structural model

The models required can be selected in the Vivado Design Suite project options.

RECOMMENDED: *Xilinx recommends that simulations utilizing UNISIM-based structural models are run using a resolution of 1 ps. Some Xilinx library components require a 1 ps resolution to work properly in either functional or timing simulation. The UNISIM-based structural models might produce incorrect results if simulation with a resolution other than 1 ps. See the "Register Transfer Level (RTL) Simulation Using Xilinx Libraries" section in the Synthesis and Simulation Design Guide for more information.*

More details on running UNISIM-based structural models can be found in Chapter 7, Detailed Example Design.

AXI4-Stream Considerations

Conversion to AXI4-Stream interfaces brings standardization and enhances interoperability of Xilinx IP LogiCore™ IP solutions. Other than general control signals such as `ALCK`, `ACLKEN` and `ARESETN`, all inputs and outputs to the LAT core are conveyed through AXI4-Stream channels. A channel consists of `TVALID` and `TDATA` always, plus several optional ports and fields. In the LAT core, the optional ports supported are `TREADY`, `TLAST`. Together, `TVALID` and `TREADY` perform a handshake to transfer a message, where the payload is `TDATA` and `TLAST`. The LAT core operates on the operands contained in the `TDATA` fields (Port A and Port B) and outputs the result in the `TDATA` field (Port C) of the output channel. This facility is expected to ease use of the LAT core in a system. The field `TLAST` indicates last cycle of packetized data, asserted by the AXI4-Stream master for the last cycle.

TIP: *This is not a mandatory requirement for the core and can be left connected to LOW (for ports A and B) if not used (or available with upstream master).*

For further details on AXI4-Stream interfaces, see the *AXI Design Reference Guide* (UG761) and the AMBA® 4 AXI4-Stream Protocol Version: 1.0 Specification.

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 value in the payload fields (`TDATA` and `TLAST`) is valid. `TREADY` indicates that the slave is ready to receive data. When both `TVALID` and `TREADY` are true (asserted) in a cycle, a transfer occurs. The master and slave will set `TVALID` and `TREADY` respectively for the next transfer appropriately. Reset needs to be asserted at least for four clock cycles.

Note: Because this is a streaming core, it does not need reset when it is done after powering up.

In matrix-matrix multiplication mode, resetting the core also clears the pipeline (that is, matrices in the output pipeline will be cleared). Avoid resetting the core in the middle of a transfer (as this is a real-time core). Reset needs to be asserted at least for F cycles (only for matrix-matrix multiplication mode) if the core has to be reset during a transfer. As shown in the following timing diagram, the clock enable signal must stay asserted when the reset is issued.

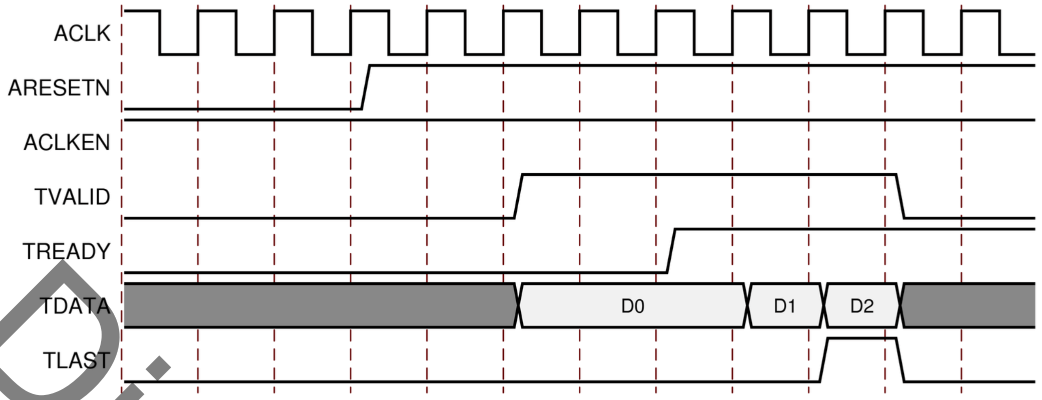


Figure 3-1: **Data Transfer in an AXI4-Stream Channel**

TDATA Packing

The `TDATA` field of the AXI4-Stream interface carries packed data. In complex data types, the real component is in the least significant position and the imaginary component follows, each are separate subfields. To ease interoperability with byte-oriented protocols, each subfield within `TDATA` that could be used independently is first extended to fit a bit field that is a multiple of 8 bits. For example, say the LAT core is configured to have an A operand width of 11 bits. Each of the real and imaginary components of A is 11 bits wide. The real component would occupy bits 10 down to 0. Bits 15 down to 11 would be ignored. Bits 26 down to 16 would hold the imaginary component and bits 31 down to 27 would also be ignored. Sub-field packing and byte-rounding are shown in Figure 3-2.

Note that the bits added by byte orientation are ignored by the core and do not result in additional resource use.

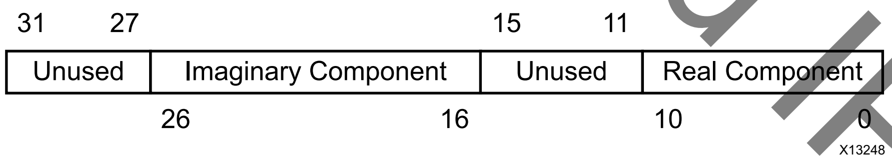


Figure 3-2: **TDATA Packing for A, B and C Ports**

Input ports A, B, and output port C carry packed and byte-rounded data in their respective `TDATA` fields. In parallel I/O mode, matrix elements (after byte-rounding and real-imaginary packing for complex data) are also packed row-wise in the `TDATA` field. Consequently, for a 4x5 matrix fully packed, `TDATA` looks ([MSP to LSP]) as follows:

$$[A_{4x5}(2,3)\ A_{4x5}(2,2)\ A_{4x5}(2,1)\ A_{4x5}(1,5)\ A_{4x5}(1,4)\ A_{4x5}(1,3)\ A_{4x5}(1,2)\ A_{4x5}(1,1)]$$

This packing is subjected to the condition that the maximum width support for `TDATA` is 1024 for this core.

Control, Interface, and Timing

The LAT v2.0 core has fully AXI4-Stream compliant interfaces for inputs and output. The core has two AXI4 slave interfaces for inputs (port A and port B) and one master interface (port C) for the output; which must be connected to external masters (for ports A and B) and slave (for port C), respectively. This core is designed for real time streaming data with the assumption that data is continuous without any interruptions when initiated. However, flow control can be achieved by the clock enable signal (`ACLKEN`), which pauses the core when deasserted externally (by a master).

Figure 3-3 shows the timing diagram for control and port A, B, C signals for matrix addition operation. As shown in the diagram, matrices A and B are input over data buses `S_AXIS_A_TDATA` and `S_AXIS_B_TDATA` and corresponding `S_AXIS_A_TVALID` and `S_AXIS_B_TVALID` signals are held High by the external AXI4-Stream master for K_A and K_B cycles (both are equal in the addition case). Corresponding ready signals (`S_AXIS_A_TREADY` and `S_AXIS_B_TREADY`) from the core stay asserted as the core is always ready for external inputs as $K_A = K_B = F$ in this case. Signals `S_AXIS_A_TLAST` and `S_AXIS_B_TLAST` are asserted by the external masters for the last cycle of input data, indicating the end of the current matrix. The output matrix exits from the core over the `M_AXIS_C_TDATA` bus after a delay of T_D cycles. The signal `M_AXIS_C_TVALID` is asserted by the core indicating valid data cycles over the data bus lasting for K_C cycles.

The external slave for outputs keeps the `M_AXIS_C_TREADY` signal asserted for receiving real-time streaming data from the core. The core asserts the `M_AXIS_C_TLAST` signal for the last cycle of K_C cycles indicating the last cycle of the current transfer of the output matrix.

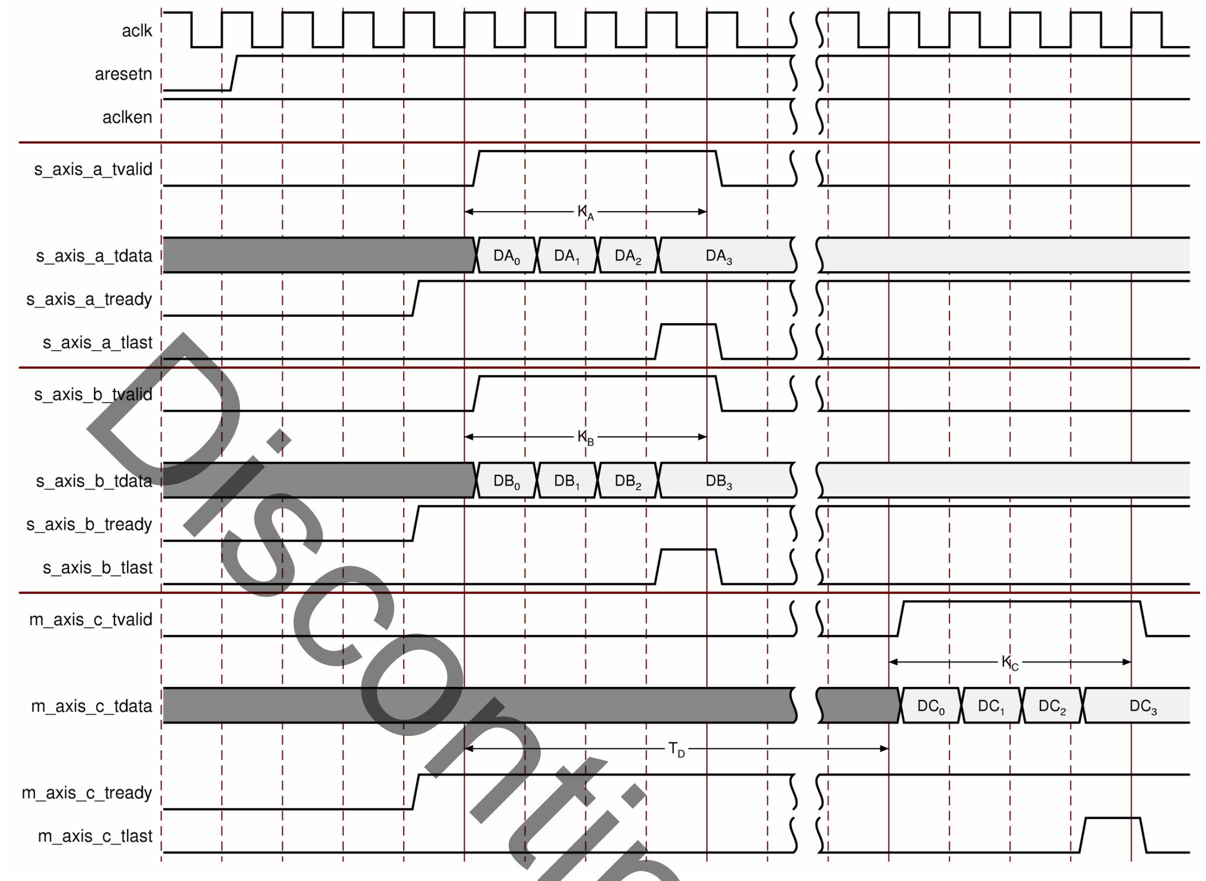


Figure 3-3: **Timing Diagram of Matrix Addition Mode**

Figure 3-4 shows continuous real-time streaming of two matrices in matrix-scalar multiplication mode. In this example, $K_A = K_C = 2$, $K_B = 1$ and F = 3. On port A, the signal `S_AXIS_A_TVALID` stays asserted for K_A and then goes down for F - K_A cycles before the next matrix is fed in. Similarly, for scalar b, `S_AXIS_A_TVALID` goes down for F - K_B cycles before the next scalar is made available. Corresponding ready signals `S_AXIS_A_TREADY` and `S_AXIS_B_TREADY` show similar behavior indicating the core is busy processing the current data. The signal `S_AXIS_B_TLAST` gets asserted for a single cycle like corresponding valid signal as K_B=1. On the output port C matrices come out one in every F cycles where `M_AXIS_C_TVALID` has a cadence of K_C cycles of high and F - K_C cycles of low.

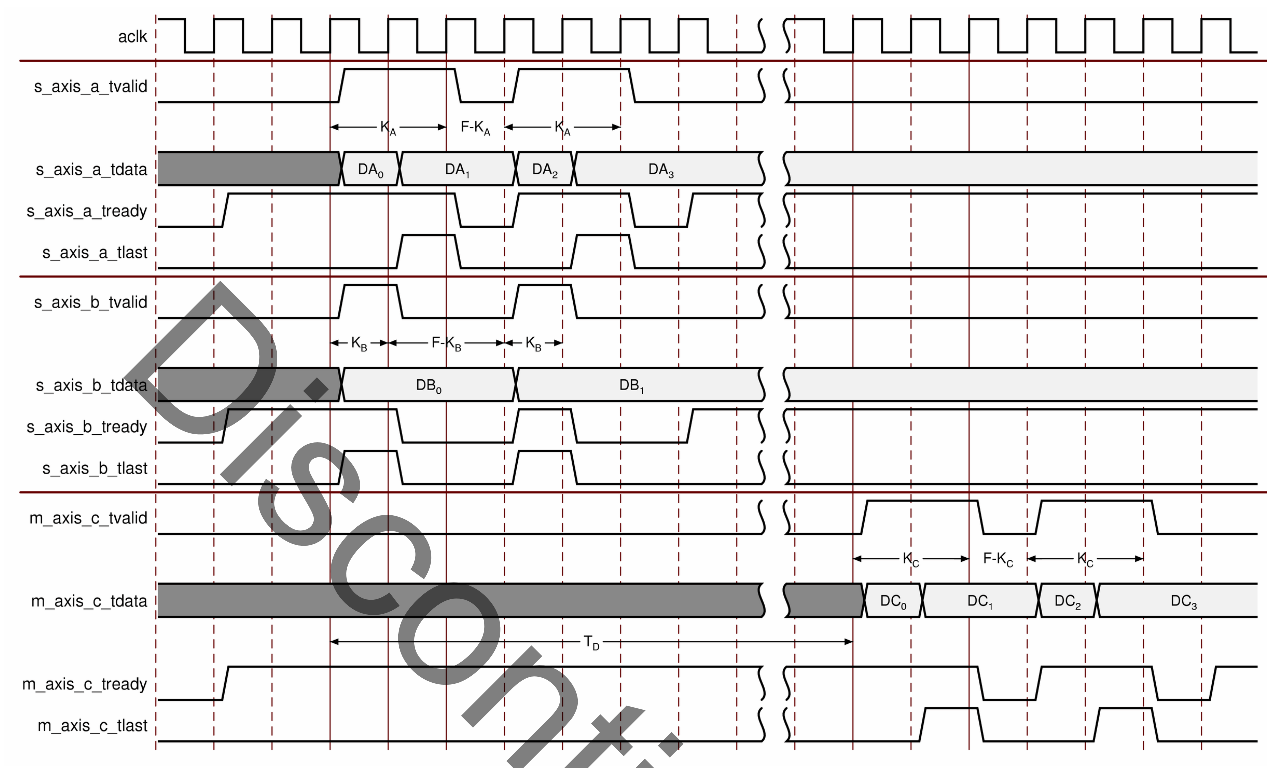


Figure 3-4: **Timing Diagram of Matrix-Scalar Multiplication Mode**

Continuous real-time streaming of two matrices is shown in Figure 3-5 for matrix-matrix multiplication mode.

$K_A = K_C = 3$, $K_B = 4$ and $F = 5$ in this case. Signals `S_AXIS_A_TVALID` and `S_AXIS_B_TVALID` stay asserted for K_A and K_B cycles respectively. On port C, output matrices come out after initial delay T_D maintaining the streaming rate of one matrix in every F cycles as in other functional modes.

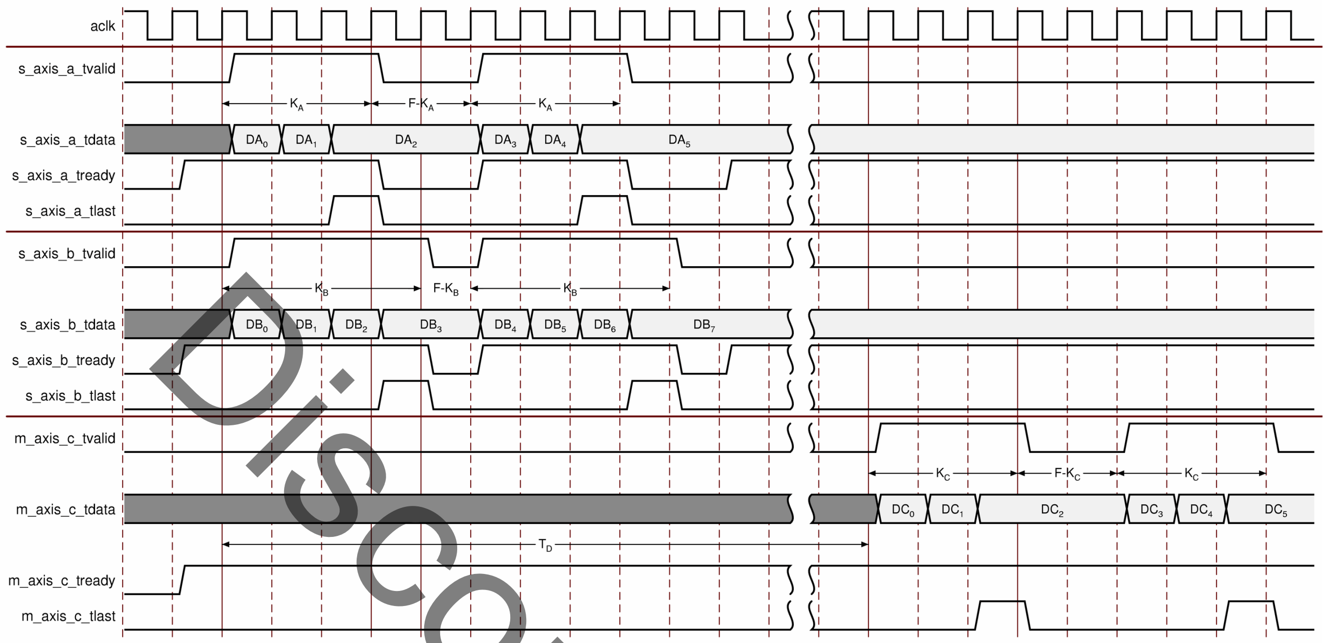


Figure 3-5: **Timing Diagram of Matrix-Matrix Multiplication Mode**

Figure 3-6 shows pause-and-resume mode of the core using clock enable signal. De-assertion of the signal `ACLKEN` freezes the core pausing both data and control paths. All the input signals (both control and data) are don't care when the core is paused. Output valid signal `M_AXIS_C_TVALID` is deasserted if in the middle of a transfer when the core is paused. As this is a fully synchronous design, output pauses after a delay of T_{CE} cycles from the de-assertion of clock enable signal. The transfer resumes (after the same delay of T_{CE}) when the clock enable signal is transitioned to the asserted state. T_{CE} is 1 clock cycles for addition/subtraction and scalar multiplication modes and for matrix-matrix multiplication this is 2 clock cycles. This feature can be used when data comes at a decimated rate.

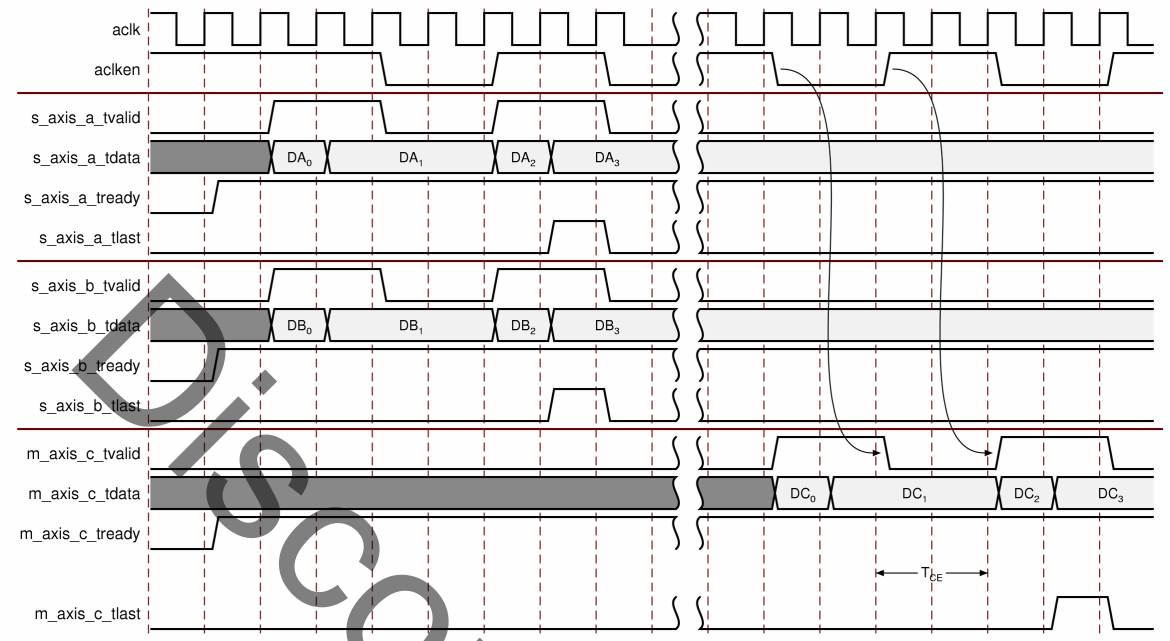


Figure 3-6: **Interface Timing Diagram With Pause-and-Resume**

Clocking

The core is fully synchronous with the AXI4-Stream clock `aclk`. The whole design operates at the positive edge of this clock.

Resets

The core has a single reset signal `aresetn`. This is an AXI4-Stream interface compliant "active-Low" reset synchronous to the core clock `aclk`. The reset needs to be held asserted at least for four clock cycles for the core to get reset fully.

C Model Reference

Introduction

The Xilinx LogiCORE™ IP Linear Algebra Toolkit v2.0 core bit accurate C model is a self-contained, linkable, shared library that models the functionality of this core with finite precision arithmetic. This model provides a bit accurate representation of the various modes of the Linear Algebra Toolkit core, and it is suitable for inclusion in a larger framework for system-level simulation or core-specific verification.

LAT v2.0 C model will be delivered through the web lounge of the product.

Features

- Bit accurate to Linear Algebra Toolkit v2.0 core
- Available for 32-bit and 64-bit Linux platforms
- Available for 32-bit and 64-bit Windows platforms
- Supports all features of the core
- Designed for integration into a larger system model
- Example C++ code provided showing how to use the C model functions

Overview

This chapter provides information about the Xilinx LogiCORE IP Linear Algebra Toolkit v2.0 bit accurate C model for 32-bit and 64-bit Linux, and 32-bit and 64-bit Windows platforms.

The model consists of a set of C functions that reside in a shared library. Example C code is provided to demonstrate how these functions form the interface to the C model. Full details of this interface are given later in this document.

The model is bit accurate but not cycle-accurate; it performs exactly the same matrix operations as the modes in the core. However, it does not model the core latency or its interface signals.

Unpacking and Model Contents

There are separate ZIP files containing all the files necessary for use with a specific computing platform. Each ZIP file contains

- The C model shared library
- Standard Template Library (STL) portability shared libraries
- The C model header file
- The example code showing customers how to call the C model
- Documentation

Table 4-1: **Example C Model ZIP File Contents - Linux**

File	Description
linear_algebra_toolkit_v2_0_bitacc_cmodel.h	Header file which defines the model API
libIp_linear_algebra_toolkit_v2_0_bitacc_cmodel.so	Model shared object library
libstlport.so.5.1	Portability library, used by the model
run_bitacc_cmodel.c	Example program for calling the C-model
README.txt	Release notes

Table 4-2: **Example C Model ZIP File Contents - Windows**

File	Description
linear_algebra_toolkit_v2_0_bitacc_cmodel.h	Header file which defines the model API
libIp_linear_algebra_toolkit_v2_0_bitacc_cmodel.dll	Model dynamically linked library
libIp_linear_algebra_toolkit_v2_0_bitacc_cmodel.lib	Model .lib file for compiling
stlport.5.1.dll	Portability library, used by the model
run_bitacc_cmodel.c	Example program for calling the C-model
README.txt	Release notes

Installation

Linux

1. Unpack the contents of the ZIP file.
2. Ensure that the directory where the `libIp_linear_algebra_toolkit_v2_0_bitacc_cmodel.so` and `libstlport.so.5.1` files reside is
   - Included in the path of the environment variable LD_LIBRARY_PATH or
   - The directory in which the executable that calls the C model is run

Windows

1. Unpack the contents of the ZIP file.
2. Ensure that the directory where the `libIp_linear_algebra_toolkit_v2_0_bitacc_cmodel.dll` and `stlport.5.1.dll` files reside is
   - Included in the path of the environment variable PATH or
   - The directory in which the executable that calls the C model is run.

Linear Algebra Toolkit v2.0 Bit Accurate C Model

C Model Interface

The Application Programming Interface (API) of the C model is defined in the header file `linear_algebra_toolkit_v2_0_bitacc_cmodel.h`. The interface consists of data structures and functions as described in the following sections.

IMPORTANT: *Included is an example C file, `run_bitacc_cmodel.c`, that demonstrates how to call the C model. See that file for examples of using the interface described in the following sections.*

Structures

There are different data structures to modularly define generics, state, inputs, and outputs of the C model to accurately reflect the behavior of the corresponding core implementation.

Generics

The `xip_linear_algebra_toolkit_v2_0_config` data structure defines parameters that are generally associated with hardware-design language (HDL) parameters. These parameters affect core implementation at the time of customization of the core and generation of its netlist, but they do not change dynamically during core operation. These parameters have a "C_" prefixed to their names and have a one-to-one mapping with the Linear Algebra Toolkit Vivado™ Integrated Design Environment (IDE) interface.

Table 4-3: **Generics structure *xilinx_ip_ linear_algebra_toolkit_v2_0_generics***

Member	Type	Description
C_M	Unsigned int	Number or rows of Matrix A Range: (2-32)
C_N	Unsigned int	Number of columns of Matrix B Range: (2-32)
C_L	Unsigned int	Applicable only to Matrix Multiplication. Number or columns of Matrix A or rows of Matrix B Range: (2-32)
C_OP_MODE	Enum variable that takes values from {MMULT,MADD,MSUB,SMULT}	Defines the Core operation to be performed. All operations involve two's complement arithmetic.
C_CMPLX	Enum variable that takes values from {REAL_TYPE, CMPLX_TYPE}	This is same as "Data Type" field in the Linear Algebra Toolkit Vivado IDE interface. All Inputs and outputs will belong to the same Data type.
C_A_WIDTH	Unsigned int	Width of the A matrix Range: (2-24 bits)(1)
C_B_WIDTH	Unsigned int	Width of the B matrix Range: (2-24 bits)(1)
C_C_WIDTH	Unsigned int	Width of the output C matrix Range:(2-48 bits)(1)
C_LSB	Unsigned int	Start location for the LSB of the result compared to the internal full precision output. 0 implies that the LSB is same as that of the internal full precision output. Range: (0-47)(1)
C_RND_MODE	Enum variable that takes values from {SYMINF,SYMZERO,ASYMINF,ASYMZERO, TRUNC,CONVEVEN,CONVODD}	Rounding Options
C_IO_MODE	Unsigned int	Does not impact C implementation and is informative in nature. 0-Serial I/O 1-Parallel I/O
C_FOLD_FAC	Unsigned int	Does not impact C implementation and is informative in nature.

1. Actual range can be further constrained by other generic parameters. See Chapter 5, Customizing and Generating the Core.

State

The `xip_linear_algebra_toolkit_v2_0` data structure defines the internal state of the C model. This structure is solely for internal use by the model, and therefore, its layout is not visible to the user of the model. User modification of this structure can lead to undefined model behavior.

Inputs

The `xip_linear_algebra_toolkit_v2_0_data_req` data structure defines the run time input parameters to the C model. This essentially contains pointers to the input matrices and their dimensions. Separate pointers are used for the real and imaginary part of the matrices. The input data is of type `xip_linear_algebra_toolkit_v2_0_data` which is obtained from sign extending the actual input to 32 bits.

Outputs

The `xip_linear_algebra_toolkit_v2_0_data_resp` data structure defines the output data of the C model. This contains pointers to the output matrix and its dimensions. The output data is of type `xip_linear_algebra_toolkit_v2_0_l_data` which is obtained by sign extending the actual output to 64 bits.

Functions

There are several functions to define the behavior of the C model, as described in the following sections.

Default Generics

The `xip_linear_algebra_toolkit_v2_0_default_config` function populates the `xip_linear_algebra_toolkit_v2_0_config` structure with default values. The members of the structure can be further modified to reflect the customization of the core in the C model behavior. The signature of this function is as follows

```
xip_linear_algebra_toolkit_v2_0_status
xip_linear_algebra_toolkit_v2_0_default_config(xip_linear_algebra_toolkit_v2_0_
config *config);
```

Create State

The `xip_linear_algebra_toolkit_v2_0_create` function creates the internal state data structure of the model and returns a pointer to this structure. If for some reason the creation of the structure fails, the function returns a null pointer. The signature of this function is as follows.

```
xip_linear_algebra_toolkit_v2_0 *xip_linear_algebra_toolkit_v2_0_create(
  const xip_linear_algebra_toolkit_v2_0_config   *config,
  msg_handler                                    handler,
  void                                           *handle
);
```

Allocate Memory for Input

The `xip_linear_algebra_toolkit_v2_0_alloc_data_req` function dynamically generates the memory required for the input structure based on the generics provided. The signature of this function is as follows.

```
xip_linear_algebra_toolkit_v2_0_statusxip_linear_algebra_toolkit_v2_0_alloc_data_req
        ( xip_linear_algebra_toolkit_v2_0          *s,
          xip_linear_algebra_toolkit_v2_0_data_req *r
        );
```

Allocate memory for output

The `xip_linear_algebra_toolkit_v2_0_alloc_data_resp` function dynamically generates the memory required for the output structure based on the generics provided. The signature of this function is as follows.

```
xip_linear_algebra_toolkit_v2_0_status xip_linear_algebra_toolkit_v2_0_alloc_data_resp
  ( xip_linear_algebra_toolkit_v2_0           *s,
    xip_linear_algebra_toolkit_v2_0_data_resp *r
  );
```

Deallocate input Memory for input

The `xip_linear_algebra_toolkit_v2_0_free_data_req` function frees the dynamically generated memory for the input structure. The signature of this function is as follows.

```
xip_linear_algebra_toolkit_v2_0_status xip_linear_algebra_toolkit_v2_0_free_data_req
  ( xip_linear_algebra_toolkit_v2_0          *s,
    xip_linear_algebra_toolkit_v2_0_data_req *r
  );
```

Deallocate memory for output

The `xip_linear_algebra_toolkit_v2_0_free_data_resp` function frees the dynamically generated memory for the output structure. The signature of this function is as follows.

```
xip_linear_algebra_toolkit_v2_0_status xip_linear_algebra_toolkit_v2_0_free_data_resp
  ( xip_linear_algebra_toolkit_v2_0           *s,
    xip_linear_algebra_toolkit_v2_0_data_resp *r
    );
```

Simulate

The `xip_linear_algebra_toolkit_v2_0_data_do` function performs bit accurate simulation. The function takes data in the input and performs the appropriate core operation. The output of the encoding is placed in output data structure. The function returns an integer value of 0 if no error occurred during the simulation process; other values indicate a failure during simulation. The signature of this function is as follows.

```
xip_linear_algebra_toolkit_v2_0_status xip_linear_algebra_toolkit_v2_0_data_do
  ( xip_linear_algebra_toolkit_v2_0           *s,
    xip_linear_algebra_toolkit_v2_0_data_req  *req,
    xip_linear_algebra_toolkit_v2_0_data_resp *resp
  );
```

Destroy State

The `xip_linear_algebra_toolkit_v2_0_destroy` function releases all memory allocated to store the state of the C model. The signature of this function is as follows.

```
xip_linear_algebra_toolkit_v2_0_status
xip_linear_algebra_toolkit_v2_0_destroy(xip_linear_algebra_toolkit_v2_0 *s);
```

Compiling

Compilation of user code requires access to the `linear_algebra_toolkit_v2_0_bitacc_cmodel.h` header file. The header file should be copied to a location where it is available to the compiler. Depending on the location chosen, the include search path of the compiler might need to be modified.

Linking

To use the C model the user executable must be linked against the correct libraries for the target platform.

Linux

The executable must be linked against the following shared object libraries.

- `libIp_linear_algebra_toolkit_v2_0_bitacc_cmodel.so`
- `libstlport.so.5.1`

Using GCC, linking is typically achieved by adding the following command line options:

```
-L. -lIp_linear_algebra_toolkit_v2_0_bitacc_cmodel -lSTL
```

This assumes the shared object libraries are in the current directory. If this is not the case, the -L. option should be changed to specify the library search path to use.

Windows

The executable must be linked against the following dynamic link libraries:

- `libIp_linear_algebra_toolkit_v2_0_bitacc_cmodel.dll`
- `stlport.5.1.dll`

Depending on the compiler, the import library might also be required:

- `libIp_linear_algebra_toolkit_v2_0_bitacc_cmodel.lib`

Using Microsoft Visual Studio.NET, linking is typically achieved by adding the import libraries to the Additional Dependencies edit box under the Linker tab of Project Properties.

Example

The `run_bitacc_cmodel.c` file contains example code to show basic operation of the C model.

The main function follows. The notes in red gives some indications on where and how to set the input data.

```
int main()
{
  unsigned  int row_index,col_index;

  cout << "C model version = " << xip_linear_algebra_toolkit_v2_0_get_version() << endl;

  // Create a configuration structure
  xip_linear_algebra_toolkit_v2_0_config config;
  xip_linear_algebra_toolkit_v2_0_status status =
xip_linear_algebra_toolkit_v2_0_default_config(&config);
  /*Configuration can be customized here. An example is shown below*/
  config.C_OP_MODE = MADD;
  config.C_CMPLX    = CMPLX_TYPE;


  if (status != XIP_LINEAR_ALGEBRA_TOOLKIT_V2_0_STATUS_OK) {
    cerr << "ERROR: Could not get C model default configuration" << endl;
    return 1;
  }

  // Create model object
  xip_linear_algebra_toolkit_v2_0* pstate;
  pstate = xip_linear_algebra_toolkit_v2_0_create(&config, &msg_print, 0);
```

```
    if (pstate == NULL) {
      cerr << "ERROR: Could not create C model state object" << endl;
      return 1;
    }

    // Create request and response structures
    xip_linear_algebra_toolkit_v2_0_data_req  request;
    xip_linear_algebra_toolkit_v2_0_data_resp response;

    //get the sizes for input data and the response
    xip_linear_algebra_toolkit_v2_0_data_calc_size(pstate, &request, &response);

    //request memory for input data
    xip_linear_algebra_toolkit_v2_0_alloc_data_req(pstate, &request);

    // Set up request
    for (row_index = 0; row_index < request.mat_a_nrows ; row_index++)
    {
      for (col_index = 0; col_index < request.mat_a_ncols ; col_index++)
      {
        //construct a simple input for Matrix A
        request.mat_a_re[row_index][col_index]=row_index+col_index;
        request.mat_a_im[row_index][col_index]=row_index+col_index;
      }
    }

    for (row_index = 0; row_index < request.mat_b_nrows ; row_index++)
    {
      for (col_index = 0; col_index < request.mat_b_ncols ; col_index++)
      {
        //construct a simple input for Matrix B
        request.mat_b_re[row_index][col_index]=row_index-col_index;
        request.mat_b_im[row_index][col_index]=row_index-col_index;
      }
    }


    // Request memory for output data
    xip_linear_algebra_toolkit_v2_0_alloc_data_resp(pstate, &response);

    //Simulate the core
    cout << "Running the C model..." << endl;

    if (xip_linear_algebra_toolkit_v2_0_data_do(pstate, &request, &response) !=
  XIP_LINEAR_ALGEBRA_TOOLKIT_V2_0_STATUS_OK)
    {
      cerr << "ERROR: C model did not complete successfully" << endl;
      xip_linear_algebra_toolkit_v2_0_free_data_req(pstate,&request);
      xip_linear_algebra_toolkit_v2_0_free_data_resp(pstate,&response);
      xip_linear_algebra_toolkit_v2_0_destroy(pstate);
      return 1;
    }
    else
    {
      cout << "C model completed successfully" << endl;
    }
    // Check response are correct

    for (row_index = 0; row_index < response.mat_out_nrows; row_index++)
    {
      for (col_index = 0; col_index < response.mat_out_ncols; col_index++)
      {
        if (response.mat_out_re[row_index][col_index]!=(2*row_index) ||
            response.mat_out_im[row_index][col_index]!=(2*row_index))
        {
          cerr << "ERROR: C model data output is incorrect" << endl;
          xip_linear_algebra_toolkit_v2_0_free_data_req(pstate,&request);
          xip_linear_algebra_toolkit_v2_0_free_data_resp(pstate,&response);
          xip_linear_algebra_toolkit_v2_0_destroy(pstate);
          return 1;
        }
      }//for row_index
    }
    cout << "C model data output is correct" << endl;

    // Clean up
    xip_linear_algebra_toolkit_v2_0_free_data_req(pstate,&request);
    xip_linear_algebra_toolkit_v2_0_free_data_resp(pstate,&response);
    cout << "C model input and output data freed" << endl;

    xip_linear_algebra_toolkit_v2_0_destroy(pstate);
    cout << "C model destroyed" << endl;

    // We will already have returned if there was an error
    return 0;
  }
```

Customizing and Generating the Core

This chapter includes information about using Xilinx tools to customize and generate the core in the Vivado™ Design Suite.

Vivado IDE Parameters

There are many parameters available for customizing an implementation of the LAT v2.0. These parameters can be configured using the two pages of the Vivado IDE. In addition, the Vivado IDE also contains three informational tabs. Tool tips appear when hovering the mouse over parameters that require additional explanation.

Tab 1: IP symbol

IP symbol tab illustrates the core pinout.

Tab 2: Implementation

The implementation tab displays the following details:

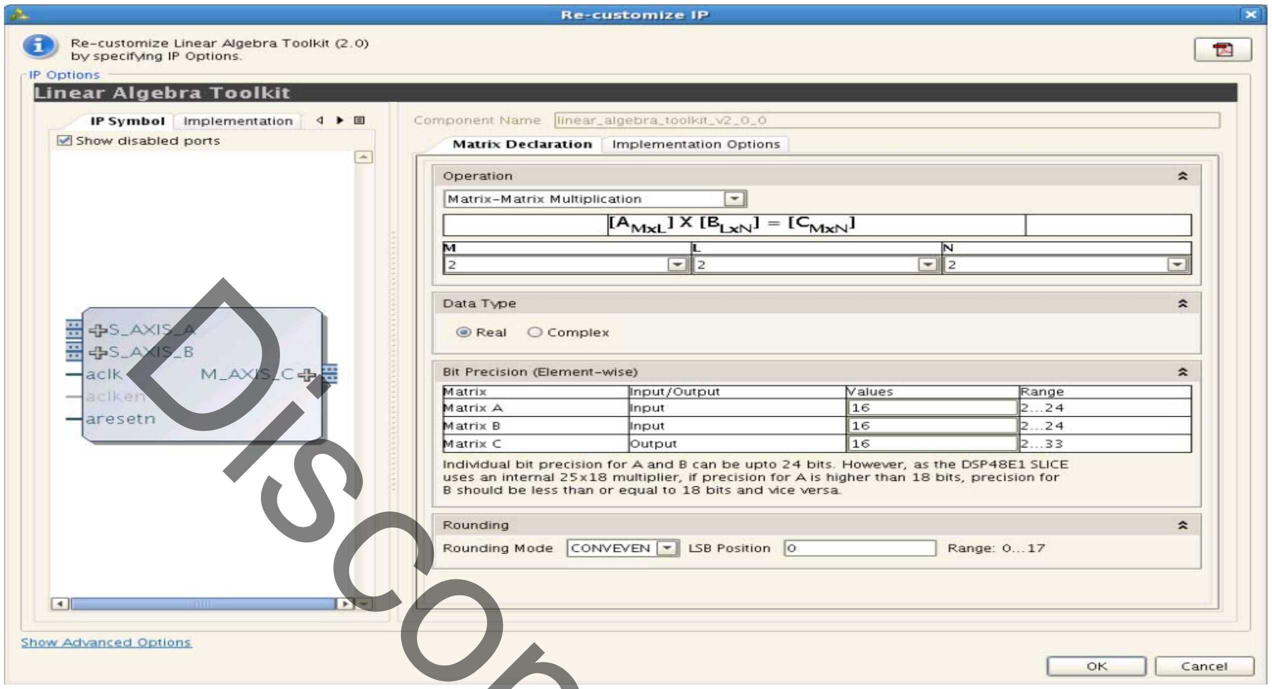
- Number of DSP48E1 slices and Block RAM elements required to implement the design
- Latency Information for the core

Tab 3: C Model

This tab points to information regarding the Bit Accurate C Model for the LAT.

Page 1

The first page of the Vivado IDE is used to define the functional mode of the LAT core as shown in Figure 5-1. Broadly, this page covers the matrix operation to be performed, the dimensions, bit precision of the matrix elements and the rounding options. The parameters are explained in greater detail in Table 5-1. All together, this page defines the complete functionality of the core.



Component Name: linear_algebra_toolkit_v2_0_0

Operation: Matrix-Matrix Multiplication

$[A_{MxL}]$ X $[B_{LxN}]$ = $[C_{MxN}]$

M	L	N
2	2	2

Data Type: [✓ Real] Complex

Bit Precision (Element-wise)

Matrix	Input/Output	Values	Range
Matrix A	Input	16	2...24
Matrix B	Input	16	2...24
Matrix C	Output	16	2...33

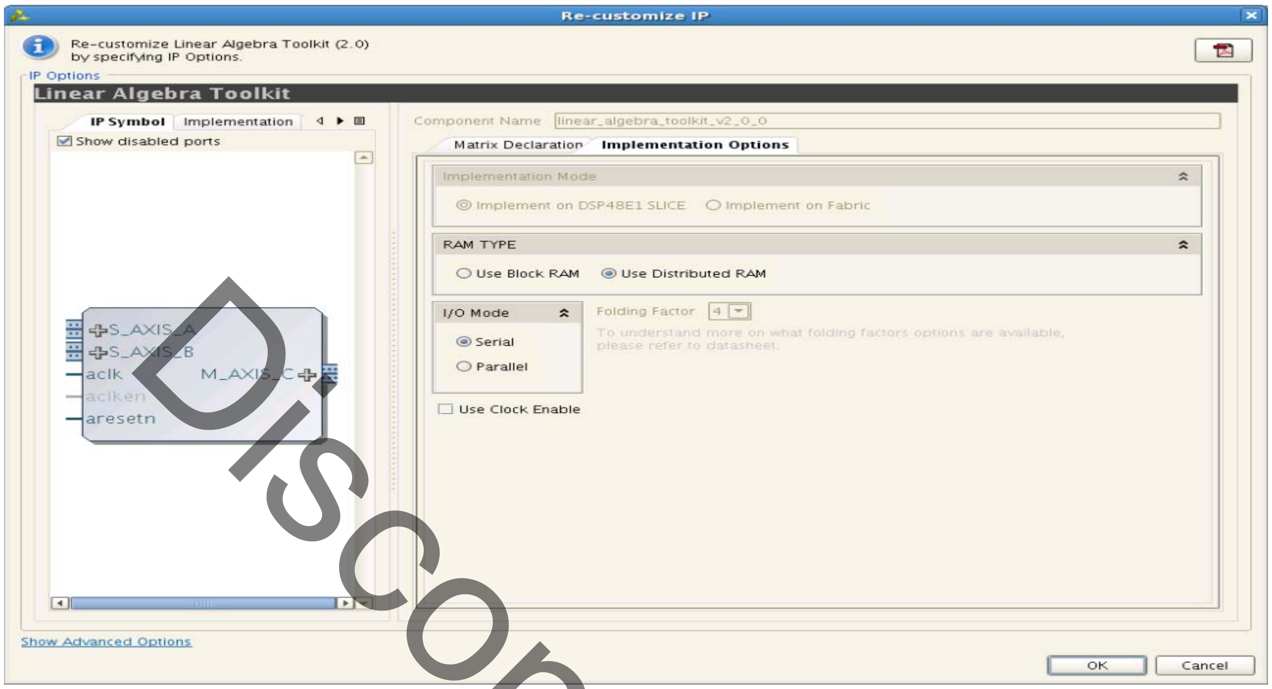
Individual bit precision for A and B can be upto 24 bits. However, as the DSP48E1 SLICE uses an internal 25x18 multiplier, if precision for A is higher than 18 bits, precision for B should be less than or equal to 18 bits and vice versa.

Rounding

Rounding Mode: CONVEVEN LSB Position: 0 Range: 0...17

Figure 5-1: **Linear IP Symbol - Matrix Declaration Tab**

The second page of the Vivado IDE (shown in Figure 5-2) is used to define the implementation details for the core operation that was defined in Page 1. For the Matrix-Matrix Addition/Subtraction, it is possible to choose between a LUT (fabric) based implementation and a DSP slice based implementation. For the Matrix-Matrix Multiplication operation, it is possible to choose between a Block RAM based implementation or a Distributed RAM based implementation. In addition, you have the flexibility to choose between a parallel and serial I/O mode. When in parallel mode, you can also choose a folding factor as a trade-off between throughput and resource utilization. The parameters with ranges and default settings are explained in detail in Table 5-1.


Re-customize IP

Re-customize Linear Algebra Toolkit (2.0) by specifying IP Options.

IP Options

Linear Algebra Toolkit

IP Symbol | Implementation

[✓] Show disabled ports

Component Name: linear_algebra_toolkit_v2_0_0

Matrix Declaration | **Implementation Options**

Implementation Mode

[● Implement on DSP48E1 SLICE] ○ Implement on Fabric

RAM TYPE

○ Use Block RAM [● Use Distributed RAM]

I/O Mode

[● Serial] ○ Parallel

Folding Factor: 4

To understand more on what folding factors options are available, please refer to datasheet.

☐ Use Clock Enable

Show Advanced Options

OK | Cancel

Figure 5-2: **IP Symbol - Implementation Options Screen**

Table 5-1: **LAT Vivado IDE Parameters**

Parameter	Range of Values	Description
	General Parameters	
Component Name	ASCII text using characters: a..z, 0..9 and "_" starting with a letter	linear_algebra_toolkit_v2_0_0
OP_MODE	Matrix-Matrix Addition, Matrix-Matrix Subtraction, Matrix-Scalar Multiplication, Matrix-Matrix Multiplication	This parameter decides which core operation to be performed.
M	1-32	Number of rows of input Matrix A, which is same as number of rows of output Matrix C
L	1-32	Applicable only to Matrix-Matrix Multiplication where it denotes the columns of A or rows of B (or the inner dimension of matrix multiplication). This value is ignored in other modes.
N	1-32	Columns of input Matrix B, which is same as number of columns of output Matrix C.
Data Type	Real, Complex	This denotes the data type of elements of Matrix A, Matrix B and Matrix C.

Table 5-1: **LAT Vivado IDE Parameters**

Parameter	Range of Values	Description
Input-Output Precision		
A_WIDTH	2-24	Input bit-width precision for elements of Matrix A. For complex data type it indicates real (or imaginary) field precision as both real and imaginary fields are assumed to be of same precision.
B_WIDTH	2-18 for Matrix Scalar multiplication, 2-24 otherwise	Input bit-width precision for elements of Matrix B. For complex data type it indicates real (or imaginary) field precision as both real and imaginary fields are assumed to be of same precision.
C_WIDTH	2-MAX_PRECISION	Output bit-width precision for elements of Matrix C. For complex data type it indicates real (or imaginary) field precision as both real and imaginary fields are assumed to be of same precision. MAX_PRECISION is 25 bits for addition/subtraction modes and 48 for scalar and matrix multiplication modes.
LSB	0-MAX_PRECISION-C_WIDTH	Determines the LSB location of the output relative to the internal full precision output value.
Rounding Mode	SYMZERO SYMINF ASYMZERO ASYMINF CONVEVEN CONVODD TRUNC	SYMZERO: Symmetric Rounding, Round down in absolute value SYMINF: Symmetric Rounding, Round up in absolute value ASYMZERO: Asymmetric Rounding, rounds towards negative infinity ASYMINF: Asymmetric Rounding, rounds towards positive infinity CONVEVEN: Convergent Rounding towards even CONVODD: Convergent Rounding towards odd TRUNC: Remove bits from LSB-1 down to zero
Implementation Specific Parameters		
Implement on Fabric/DS48E1 Slice	Radio Button	For Matrix-Matrix Addition/Subtraction the implementation can be on fabric or DSP slice. Only one of these two options can be selected, default is implementation on fabric. Note, matrix-scalar multiplication and matrix-matrix multiplication are implemented on DSP48E1 and hence this choice is not available for these modes.
I/O Mode	Serial, Parallel	Selects whether data will be transferred serially or in parallel. Default I/O mode is serial.

Table 5-1: **LAT Vivado IDE Parameters**

Parameter	Range of Values	Description
Folding Factor	1-M, N (addition/subtraction and scalar multiplication modes) 1-M (matrix multiplication mode)	Implies the amount of reuse of the resources. Roughly, F implies (1/F)*100% use of resources compared to Folding Factor 1. This is applicable only when I/O Mode is parallel, as the Folding Factor gets decided automatically in serial I/O mode.
Use Clock Enable	Enable/Disable	Adds or Removes Clock Enable port from the core. Default is Disable (no Clock Enable port).
Use Block RAM/Use Distributed RAM	Radio Button	Available only in matrix-matrix multiplication mode, where user has a choice between Block RAM and Distributed RAM for internal data storage. Default option is use Distributed RAM.

Output Generation

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 User Guide: Designing with IP* (UG896).

- **<project_directory>**
  Top-level project directory; name is user-defined
  - <project_directory>/<component name>
    Contains files for configuration parameters and encrypted VHDL source

<project_directory>

The `project` directory contains all the Vivado IP catalog project files.

<project_directory>/<component name>

The `component name` directory contains files for configuration parameters and encrypted VHDL source.

Table 5-2: **Component Name Directory**

Name	Description
`<project_directory>/<component_name>`	
`<component_name>.xci`	This file contains all configuration parameters needed to generate the core.
`<file_name>.vhd`	Encrypted VHDL source files

Back to Top

Constraining the Core

This chapter contains information about constraining the core in the Vivado™ Design Suite.

Required Constraints

There are no required constraints for this core.

Device, Package, and Speed Grade Selections

There are no device, package, or speed grade selections for this core.

Clock Frequencies

There are no specific clock frequency requirements for this core.

Clock Management

There is no specific clock management for this core.

Clock Placement

There are no specific clock placement requirements for this core.

Banking

There are no specific banking rules for this core.

Transceiver Placement

There are no transceiver placement requirements for this core.

I/O Standard and Placement

There are no specific I/O standards and placement requirements for this core.

Detailed Example Design

This chapter contains information about the test bench in the Vivado™ Design Suite.

When the core is generated using the Vivado IDE, a demonstration test bench (and example test vectors file) can be created by following these steps:

1. Right-click on the core name in **Project Manager, Sources**.
2. Select G**enerate Output Products...**
3. In the **Manage Output Products** dialog box, select **Generate** in the Test Bench drop-down menu.

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 design tools output directory. The source code is comprehensively commented.

Using the Demonstration Test Bench

The demonstration test bench instantiates the generated Linear Algebra Toolkit core. If the Vivado IDE project options are set to generate a structural model, a VHDL or Verilog netlist named `<component_name>.vhd` or `<component_name>.v` is generated. This file can be generated after synthesizing the core with the following tcl command:

write_vhdl -mode funcsim <component_name>**.vhd**

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 simulators waveform viewer to see the operations of the test bench.

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
    - Reset behavior
    - Clock Enable support
    - AXI4-Stream support
- Checks the core output data against a reference output for errors

More information on the demonstration test bench is available in the `tb_readme.txt` file that is generated as part of the Vivado Design Suite output.

Migrating

This appendix describes migrating from older versions of the IP to the current IP release.

For information on migrating to the Vivado™ Design Suite, see the *Vivado Design Suite Migration Methodology Guide* (UG911).

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 Linear Algebra Toolkit core.

The following topics are included in this appendix

- Finding Help on Xilinx.com
- Debug Tools
- Hardware Debug
- Debug for AXI4-Stream Interface

Finding Help on Xilinx.com

To help in the design and debug process when using the Linear Algebra Toolkit core, the Xilinx Support web page (www.xilinx.com/support) contains key resources such as product documentation, release notes, answer records, information about known issues, and links for opening a Technical Support WebCase.

Documentation

This product guide is the main document associated with the Linear Algebra Toolkit core. This guide, along with documentation related to all products that aid in the design process, can be found on the Xilinx Support web page (www.xilinx.com/support) or by using the Xilinx Documentation Navigator.

Download the Xilinx Documentation Navigator from the Design Tools tab on the Downloads page (www.xilinx.com/download). For more information about this tool and the features available, open the online help after installation.

Known Issues

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 can also 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 Linear Algebra Toolkit core

AR 54505

Contacting Technical Support

Xilinx provides technical support at www.xilinx.com/support for this LogiCORE™ IP product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support of product if implemented in devices that are not defined in the documentation, if customized beyond that allowed in the product documentation, or if changes are made to any section of the design labeled DO NOT MODIFY.

To contact Xilinx Technical Support:

1. Navigate to www.xilinx.com/support.
2. Open a WebCase by selecting the WebCase link located under Support Quick Links.

When opening a WebCase, include:

- Target FPGA including package and speed grade.
- All applicable Xilinx Design Tools and simulator software versions.
- Additional files based on the specific issue might also be required. See the relevant sections in this debug guide for guidelines about which file(s) to include with the WebCase.

Debug Tools

There are many tools available to address Linear Algebra Toolkit core design issues. It is important to know which tools are useful for debugging various situations.

Example Design

The Linear Algebra Toolkit core is delivered a with test bench that can be simulated. Information about the example design can be found in *Chapter 7, Detailed Example Design*.

Vivado Lab Tools

Vivado™ lab tools insert logic analyzer and virtual I/O cores directly into your design. Vivado lab tools 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 FPGA devices 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)

Reference Boards

Various Xilinx development boards support Linear Algebra Toolkit core. These boards can be used to prototype designs and establish that the core can communicate with the system.

- 7 series FPGA evaluation boards
    - KC705
    - KC724

C-Model Reference

See *Chapter 4, C Model Reference* in this guide for tips and instructions for using the provided C-Model files to debug your design.

Hardware Debug

Hardware issues can range from link bring-up to problems seen after hours of testing. This section provides debug steps for common issues. The Vivado lab tools are a valuable resource to use in hardware debug. The signal names mentioned in the following individual sections can be probed using the Vivado Lab Tools for debugging the specific problems.

Ensure that all the timing constraints for the core were properly incorporated from the example design and that all constraints were met during implementation. Following is a list of general checks.

- Does it work in post-place and route timing simulation? If problems are seen in hardware but not in timing simulation, this could indicate a PCB issue. Ensure that all clock sources are active and clean.
- If using MMCMs in the design, ensure that all MMCMs have obtained lock by monitoring the `LOCKED` port.
- If your outputs go to 0, check your licensing.

Debug for AXI4-Stream Interface

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 3-3, Figure 3-4, Figure 3-5, and Figure 3-6.
- Check core configuration.
- Add appropriate core specific checks.

Additional Resources

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see the Xilinx Support website at:

www.xilinx.com/support.

For a glossary of technical terms used in Xilinx documentation, see:

www.xilinx.com/company/terms.htm.

Reference Documents

1. *AXI Design Reference Guide* (UG761)
2. AMBA® 4 AXI4-Stream Protocol Version: 1.0 Specification
3. Vivado™ design tools user documentation
4. Synthesis and Simulation Design Guide
5. *Vivado Design Suite User Guide: Designing with IP* (UG896)
6. *Vivado Design Suite Migration Methodology Guide* (UG911)

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
03/20/2013	1.0	Initial Xilinx release of product guide. This product guide replaces ds829.

Notice of Disclaimer