Core429 v3.12

Handbook

Microsemi

Revision History

Date	Revision	Change
April 2016	6	Added PolarFire support.
August 2015	5	Updated the version v3.11 and applied new template.

Confidentiality Status

This is a non-confidential document.

Table of Contents

Preface 5
About this Document 5
Intended Audience 5

Introduction 6
Overview 6
Key Features 8
Core Version 8
Supported Families 8
Utilization and Performance 9
Memory Requirements 11

Functional Block Description 12
Core429 Overview 12
Functional Description 12
Clock Requirements 14
Line Drivers 14
Line Receivers 14
Development System 14

Operation 16
Default Mode Operation 16
Loopback Interface 19

Interface Description 20
Configuration Parameters 20
I/O Signal Descriptions 21

Timing Diagrams 23
CPU Interface Timing for Default Mode 23

Tool Flows 25
License Types 25
SmartDesign 25

Testbench Operation and Modification 29
Testbench 29
User Test-bench 30

Ordering Information 31
Ordering Codes 31

Appendix A: Testbench Support Routines 32
VHDL Support 32
Verilog Support 32

List of Changes 33

Product Support 34
Customer Service 34
Customer Technical Support Center 34
Technical Support 34
Website 34
Contacting the Customer Technical Support Center 34
ITAR Technical Support 35

Preface

About this Document

This handbook provides details about the Core429 DirectCore module, and how to use it.

Intended Audience

FPGA designers using Libero® System-on-Chip (SoC).

Introduction

Overview

Core429 provides a complete Transmitter (Tx) and Receiver (Rx). A typical system implementation using Core429 is shown in Figure 1.

The core consists of three main blocks: Transmit, Receive, and CPU Interface (Figure 1). Core429 requires connection to an external CPU. The CPU interface configures the transmit and receive control registers and initializes the label memory. The core interfaces to the ARINC 429 bus through an external ARINC 429 line driver and line receiver. A detailed description of the Rx and Tx interfaces is provided in Functional Description.

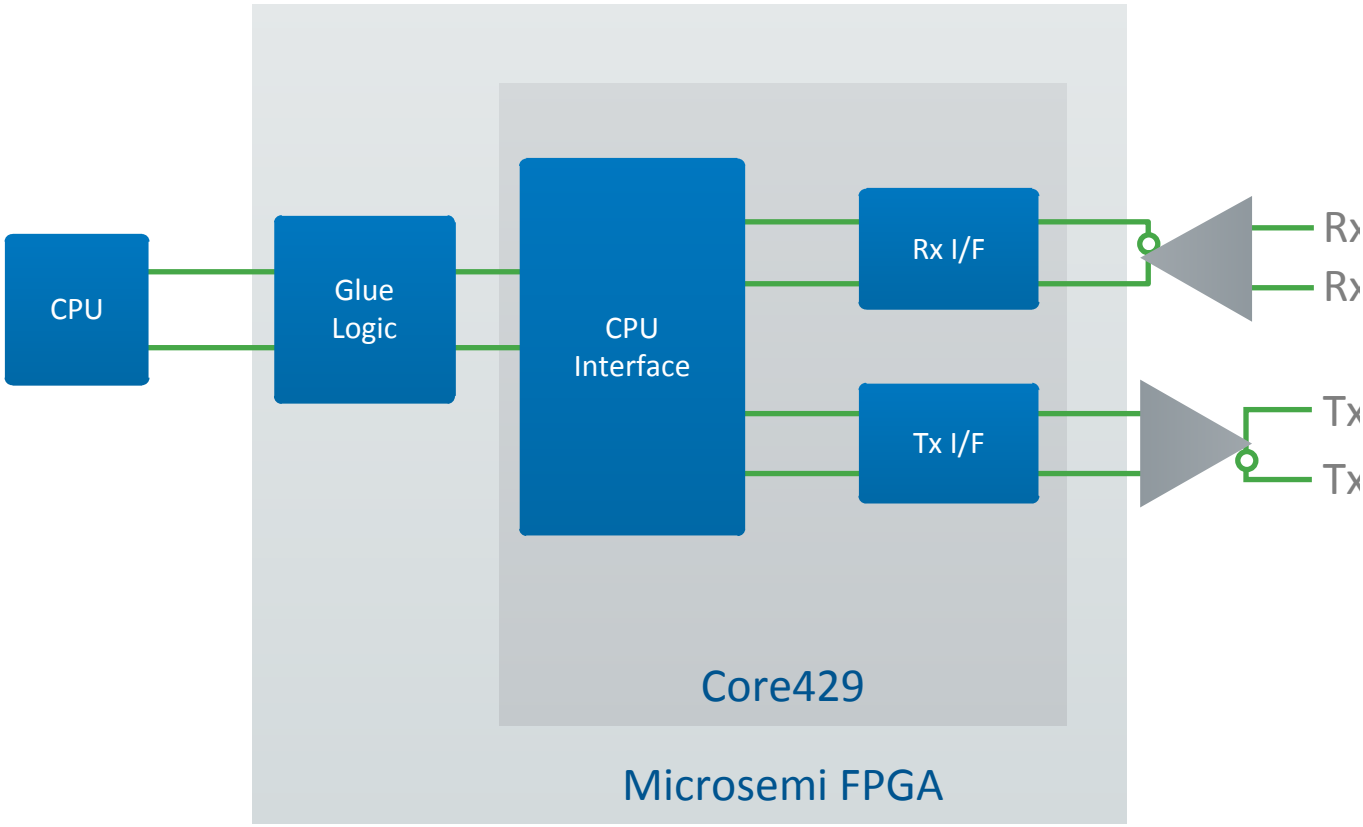

Figure 1 Typical Core429 System—One Tx and One Rx

External Components

There are two external components required for proper operation of Core429:

- Standard ARINC 429 line driver
- Standard ARINC 429 line receiver

ARINC 429 Overview

ARINC 429 is a two-wire, point-to-point data bus that is application-specific for commercial and transport aircraft. The connection wires are twisted pairs. Words are 32 bits in length and most messages consist of a single data word. The specification defines the electrical standard and data characteristics and protocols.

ARINC 429 uses a unidirectional data bus standard (Tx and Rx are on separate ports) known as the Mark 33 Digital Information Transfer System (DITS). Messages are transmitted at 12.5, 50 (optional), or 100 kbps to other system elements that are monitoring the bus messages. The transmitter is always transmitting either 32-bit data words or the Null state.

The ARINC standard supports High, Low, and Null states (Figure 2). A minimum of four Null bits should be transmitted between ARINC words. No more than 20 receivers and no less than one receiver can be connected to a single bus (wire pair), though there will normally be more.

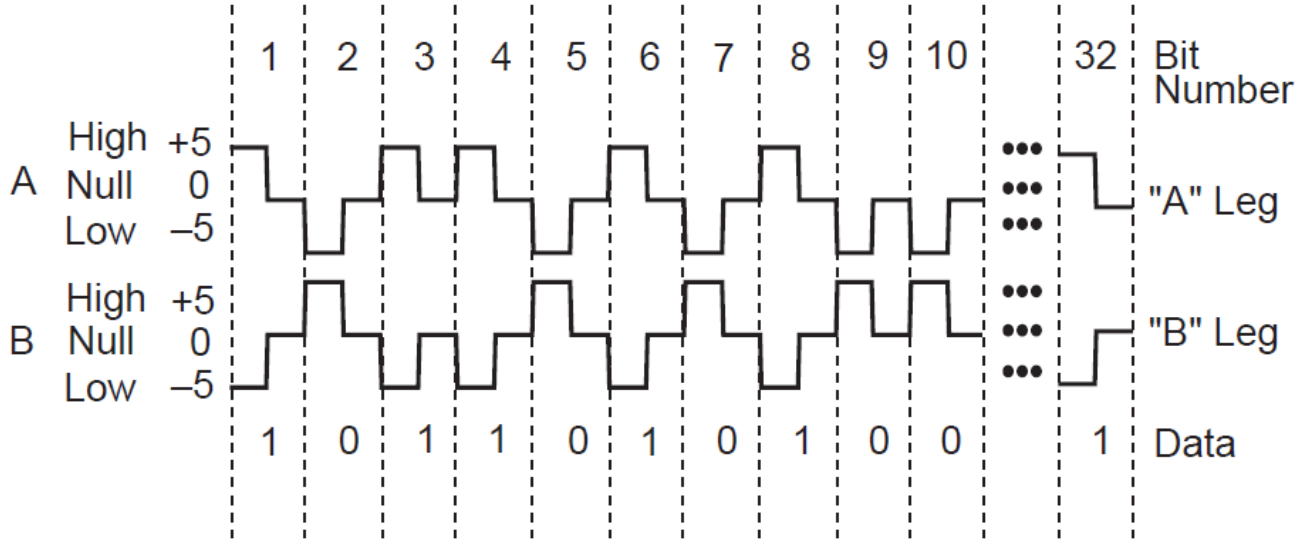

Figure 2 ARINC Standard

Figure 3 shows the bit positions of ARINC data.

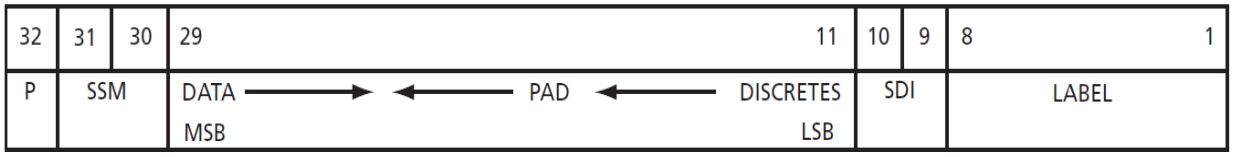

Figure 3 ARINC Data Bit Positions

Each ARINC word contains five fields:

- Parity
- Sign/Status Matrix
- Data
- Source/Destination Identifiers
- Label

The parity bit is bit 32 (the MSB). SSM is the Sign/Status Matrix and is included as bits 30 and 31. Bits 11 to 29 contain the data. Binary-coded decimal (BCD) and binary encoding (BNR) are common ARINC data formats. Data formats can also be mixed. Bits 9 and 10 are Source/Destination Identifiers (SDI) and indicate for which receiver the data is intended. Bits 1 to 8 contain a label (label words) identifying the data type.

Label words are quite specific in ARINC 429. Each aircraft may be equipped with different electronic equipment and systems needing interconnection. A large amount of equipment may be involved, depending on the aircraft. The ARINC specification identifies the equipment ID, a series of digital identification numbers. Examples of equipment are flight management computers, inertial reference systems, fuel tanks, tire pressure monitoring systems, and GPS sensors.

Transmission Order

The least significant bit of each byte, except the label, is transmitted first, and the label is transmitted ahead of the data in each case. The order of the bits transmitted on the ARINC bus is as follows:

8, 7, 6, 5, 4, 3, 2, 1, 9, 10, 11, 12, 13 … 32.

Key Features

Core429 supports the following features:

- Supports ARINC specification 429-16
- Configurable up to 16 Rx and 16 Tx channels
- Programmable FIFO depth
- Programmable interrupt generation
- Configurable label memory size
- Selectable data rate on each channel

Core Version

This handbook is for Core429 version 3.12.

Supported Families

- PolarFire
- SmartFusion®2
- IGLOO®2
- IGLOO
- IGLOOe
- IGLOO PLUS
- ProASIC®3/E
- Fusion®
- SmartFusion
- Axcelerator®
- RTAX™-S
- ProASIC PLUS®
- ProASIC®3
- ProASIC3L

Utilization and Performance

Core429 can be implemented in several Microsemi FPGA devices. Table 1 through Table 5 provide typical utilization data using standard synthesis tools for different Core429 configurations. Table 1 assumes that the label size is set to 64 and FIFO depth is set to 64.

Table 1 Device Utilization for One Tx Module (default mode)

FPGA Family	Cells or Tiles			Memory Blocks	Device	Utilization
	Combinatorial	Sequential	Total			
SmartFusion	123	74	197	1	M2S050T	0.5%
SmartFusion	154	162	316	1	A2F500M3G	2%
Fusion	363	147	510	1	AFS600	4%
ProASIC3/E	363	147	510	1	A3PE600	4%
ProASIC[PLUS]	441	146	587	1	APA075	19%
Axcelerator	212	145	357	1	AX125	18%
RTAX-S	258	171	429	1	RTAX250S	10%
IGLOO2	115	77	192	1	M2GL150T	0.13%
PolarFire	112	77	189	1	PA5M500	0.02%

Table 2 Device Utilization for One Rx Module (default mode)

FPGA Family	Cells or Tiles			Memory Blocks	Devices	Utilization
	Combinatorial	Sequential	Total			
SmartFusion2	320	247	567	2	M2S050T	1%
SmartFusion	415	229	644	2	A2F500M3G	6%
Fusion	431	233	664	2	AFS600	5%
ProASIC3/E	431	233	664	2	A3PE600	5%
ProASIC[PLUS]	588	236	824	2	APA075	27%
Axcelerator	307	234	541	2	AX125	27%
RTAX-S	350	259	609	2	RTAX250S	14%
IGLOO2	400	337	737	2	M2GL150T	0.5%
PolarFire	432	344	776	2	PA5M500	0.07%

Table 3 Device Utilization for One Rx and One Tx Module (default mode)

FPGA Family	Cells or Tiles			Memory Blocks	Device	Utilization
	Combinatorial	Sequential	Total			
SmartFusion2	619	430	1,049	3	M2S050T	2%
SmartFusion	802	384	1,186	3	A2F500M3G	10%
Fusion	848	380	1.228	3	AFS600	10%
ProASIC3/E	848	380	1,228	3	A3PE600	10%
ProASIC^{PLUS}	1,084	382	1,466	3	APA075	48%
Axcelerator	518	378	896	3	AX125	44%
RTAX-S	604	429	1,033	3	RTAX250S	24%
IGLOO2	696	556	1,252	3	M2GL150T	0.86%
PolarFire	754	562	1,316	3	PA5M500	0.12%

Table 4 Device Utilization for 16 Rx and 16 Tx Modules (default mode)

FPGA Family	Cells or Tiles			Memory Blocks	Device	Utilization
	Combinatorial	Sequential	Total			
SmartFusion2	9,439	6,695	16,134	48	M2S050T	29%
SmartFusion	12,308	6,039	18,347	48	A2F500M3G	159%
Fusion	13,435	6,080	19,515	48	AFS1500	51%
ProASIC3/E	13,435	6,080	19,515	48	A3PE1500	51%
ProASIC^{PLUS}	16,835	6,112	22,947	48	APA750	69%
Axcelerator	8,044	5,944	13,988	48	AX2000	43%
RTAX-S	9,594	6,745	16,339	48	RTAX2000S	51%
IGLOO2	10,901	8,831	19,732	48	M2GL150T	13.5%
PolarFire	11,573	8,902	20,475	48	PA5M500	1.8%

The Core429 clock rate can be configured to be 1, 10, 16, 20, or 66 MHz. All the Microsemi families listed above easily meet the required performance.

Core429 I/O requirements depend on the system requirements and external interfaces. If the core and memory blocks are implemented within the FPGA and the CPU interface has a bidirectional data bus, approximately 74 I/O pins are required to implement four Rx and four Tx modules. The core will require 62 pins to implement one Rx and one Tx module.

The core has various FIFO flags available for debugging purposes. These flags may not be needed in the final design, and this will reduce the I/O count.

Memory Requirements

The number of memory blocks required differs depending on whether each channel is configured identically or differently.

Each Channel Configured Identically

Use EQ 1 to calculate the number of memory blocks required if each channel is configured identically.

$$\text{Number of memory blocks} = NRx * (INT(LABEL\_SIZE / X) + INT(RX\_FIFO\_DEPTH / Y) + NTx * INT(FIFO\_DEPTH / Y), \quad \text{EQ 1}$$

where NRx is the number of receive channels, NTx is the number of transmit channels, INT is the function to round up to the next integer, and X and Y are defined in Table 5.

Each Channel Configured Differently

Use EQ 2 to calculate the number of memory blocks required if each channel is configured differently.

$$\text{Number of memory blocks} = \sum_{I=0}^{NTx-1} INT(FIFO\_DEPTH[I] / Y + \sum_{I=0}^{NRx-1} (INT(LABEL\_SIZE[I] / X) + INT(FIFO\_DEPTH[I] / Y)), \quad \text{EQ 2}$$

where NRx is the number of receive channels, NTx is the number of transmit channels, INT is the function to round up to the next integer, and X and Y are defined in Table 5.

Table 5 Memory Parameters

Device Family	X	Y
Fusion	512	128
ProASIC3/E	512	128
ProASICPLUS	256	64
Axcelerator/RTAX-S	512	128
IGLOO2	512	128
SmartFusion2	512	128
SmartFusion	512	128
PolarFire	512	128

Examples for the ProASIC3/E Device Family

If the design has 2 receivers, 1 transmitter, 64 labels for each receiver, and a 32-word-deep FIFO for each receiver and transmitter, then

Number of memory blocks = 2 * (INT(64 / 512) + INT(32 / 128)) + 1 * INT(32 / 128)

= 2 * (1 + 1) + 1 * (1) = 5.

If the design has 2 receivers, 1 transmitter, 32 labels for receiver #1, 64 labels for receiver #2, a 32-word-deep FIFO for receiver #1, a 64-word-deep FIFO for receiver #2, and a 64-word-deep FIFO for the transmitter, then

Number of memory blocks = INT(64 / 128) + (INT(32 / 512) + INT(32 / 128)) + (INT(64 / 512) + INT(64 / 128)) = 1 + (1 + 1) + (1 + 1) = 5.

Functional Block Description

Core429 Overview

Core429 provides a complete and flexible interface to a microprocessor and an ARINC 429 data bus. Connection to an ARINC 429 data bus requires additional line drivers and line receivers.

Core429 interfaces to a processor through the internal memory of the receiver. Core429 can be easily interfaced to an 8-, 16-, or 32-bit data bus. Lookup tables loaded into memory enable the Core429 receive circuitry to filter and sort incoming data by label and destination bits. Core429 supports multiple (configurable) ARINC 429 receiver channels, and each receives data independently. The receiver data rates (high- or low-speed) can be programmed independently. Core429 can decode and sort data based on the ARINC 429 Label and SDI bits and stores it in a FIFO. Each receiver uses a configurable FIFO to buffer received data. Core429 supports multiple (configurable) ARINC 429 transmit channels, and each channel can transmit data independently.

Functional Description

The core has three main blocks: Transmit, Receive, and CPU Interface. The core can be configured to provide up to 16 transmit and receive channels.

Figure 4 gives a functional description of the Rx block.

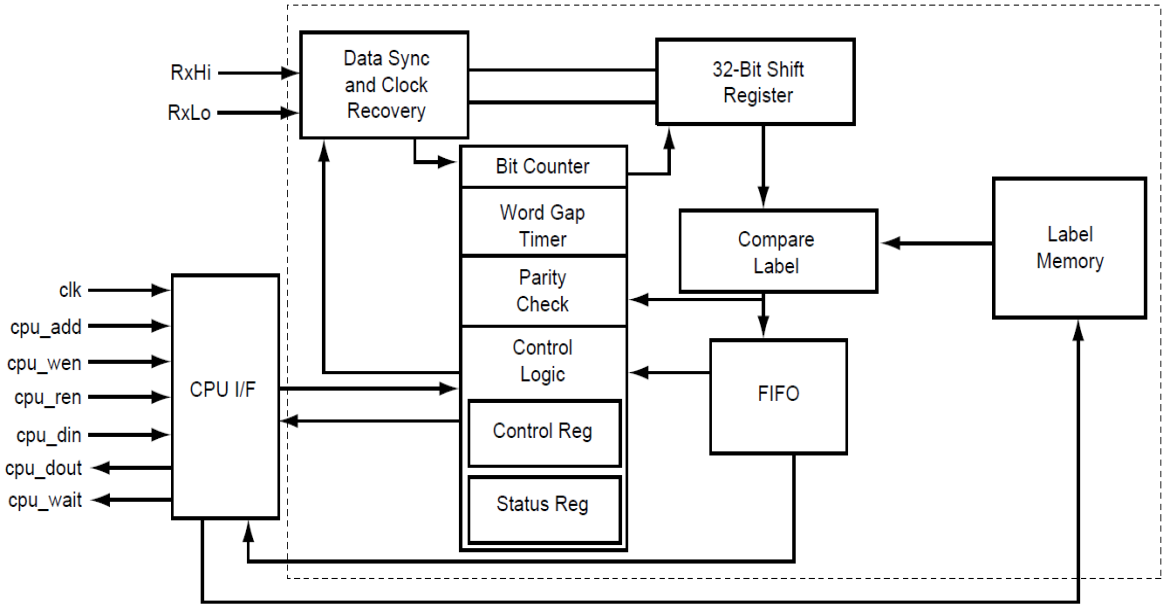

Figure 4 Core429 Rx Block Diagram

The Rx block is responsible for recovering the clock from the input serial data and performs serial-to- parallel conversion and gap/parity checking on the incoming data. It also interfaces with the CPU.

The Rx module contains two 8-bit registers. One is used for control and the other is used for status. Refer to Table 12 and Table 13 for detailed descriptions of the control and status register bits. The CPU interface configures the internal RAM with the labels, which are used to compare against incoming labels from received ARINC data.

If the label-compare bit in the receive control register is enabled, data whose labels match the stored labels will be stored in the FIFO. If the label-compare bit in the receive control register is disabled, the incoming data will be stored in the FIFO without comparing against the labels in RAM.

The core supports reloading label memory using bit 7 of the Rx control register. Note that when you set bit 7 to initialize the label memory, the old label content still exists, but the core keeps track only of the new label and does not use the old label during label compare.

The FIFO asserts three status signals:

- rx_fifo_empty: FIFO is empty
- rx_fifo_half_full: FIFO is filled up to the configured RX_FIFO_LEVEL
- rx_fifo_full: FIFO is full

Depending on the FIFO status signals, the CPU will either read the FIFO before it overflows or will not attempt to read the FIFO if it is empty. The interrupt signal int_out_rx is generated when one of the FIFO status signals (rx_fifo_empty, rx_fifo_half_full, or rx_fifo_full) is High.

Figure 5 gives a functional description of the Tx block.

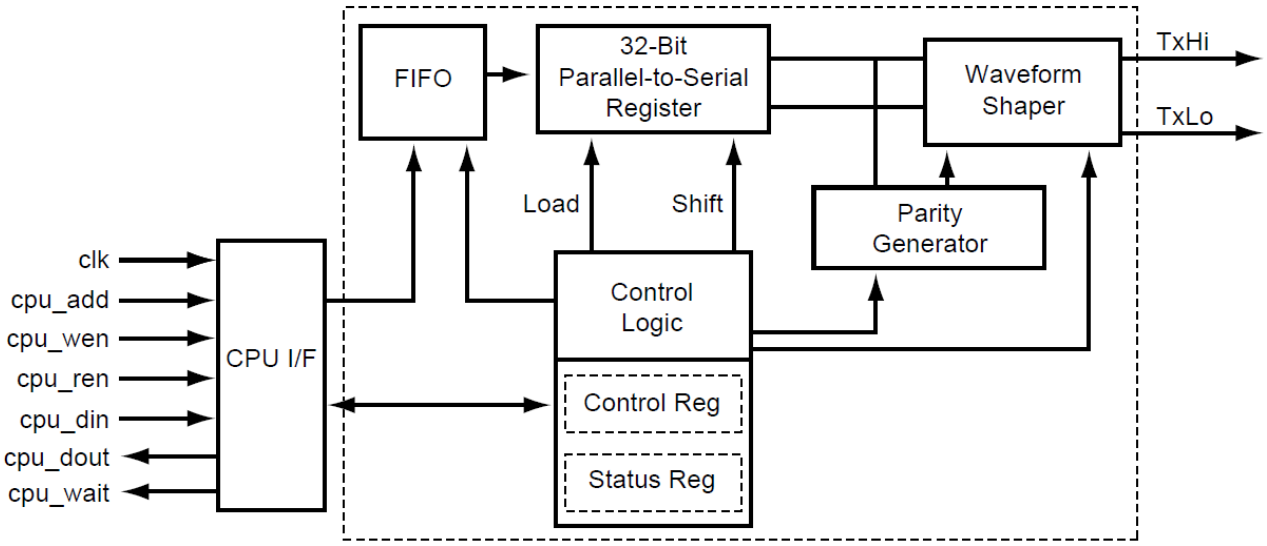

Figure 5 Core429 Tx Block Diagram

The Tx module converts the 32-bit parallel data from the Tx FIFO to serial data. It also inserts the parity bit into the ARINC data when parity is enabled. The CPU interface is used to fill the FIFO with ARINC data. The Tx FIFO can hold up to 512 ARINC words of data. The transmission starts as soon as one complete ARINC word has been stored in the transmit FIFO.

The Tx module contains two 8-bit registers. One is used for a control function and the other is used for status. Refer to Table 16 and Table 17 for detailed descriptions. The CPU interface allows the system CPU to access the control and status registers within the core.

The Tx FIFO asserts three status signals:

- tx_fifo_empty: Tx FIFO is empty
- tx_fifo_half_full: Tx FIFO is filled up to the configured TX_FIFO_LEVEL
- tx_fifo_full: Tx FIFO is full

Depending on the FIFO status signals, the CPU will either write to the FIFO as long as it is not full or will not attempt to write to the FIFO if it is full. The interrupt signal int_out_tx is generated when one of the FIFO status signals (tx_fifo_empty, tx_fifo_half_full, or tx_fifo_full) is HIGH.

Clock Requirements

To meet the ARINC 429 transmission bit rate requirements, the Core429 clock input must be 1, 10, 16, 20, or 66 MHz with a tolerance of ±0.01%.

Line Drivers

Core429 needs ARINC 429 line drivers to drive the ARINC 429 data bus. Core429 is designed to interface directly to common ARINC 429 line drivers, such as HOLT HI-8382/HI-8383, DDC DD-03182, or Device Engineering DEI1070.

Figure 6 shows the connections required from Core429 to the line drivers.

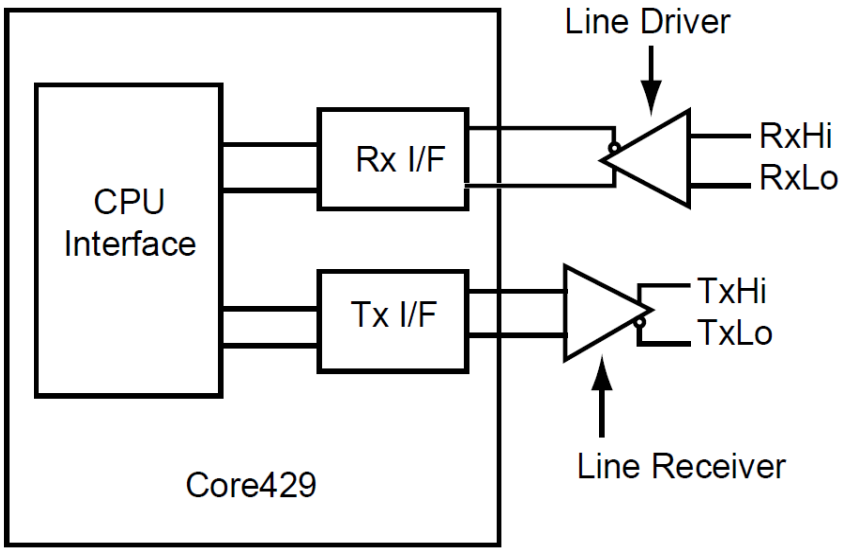

Figure 6 Core429 Line Driver and Line Receiver Interface

Line Receivers

Core429 needs ARINC 429 line receivers to receive the ARINC 429 data bus. Core429 is designed to interface directly to common ARINC 429 line receivers, such as HOLT HI-8588 or Device Engineering DEI3283. When using an FPGA from the ProASIC^{PLUS}, RTAX-S, or Axcelerator families, level translators are required to connect the 5 V output levels of the Core429 line receivers to the 3.3 V input levels of the FPGA.

Development System

A complete ARINC 429 development system is also available, part number Core429-DEV-KIT. The development system uses an external terminal (PC) with a serial UART link to control Core429 with four Rx and four Tx channels implemented in a single ProASIC^{PLUS} APA600 FPGA.

Figure 7 shows the development system.

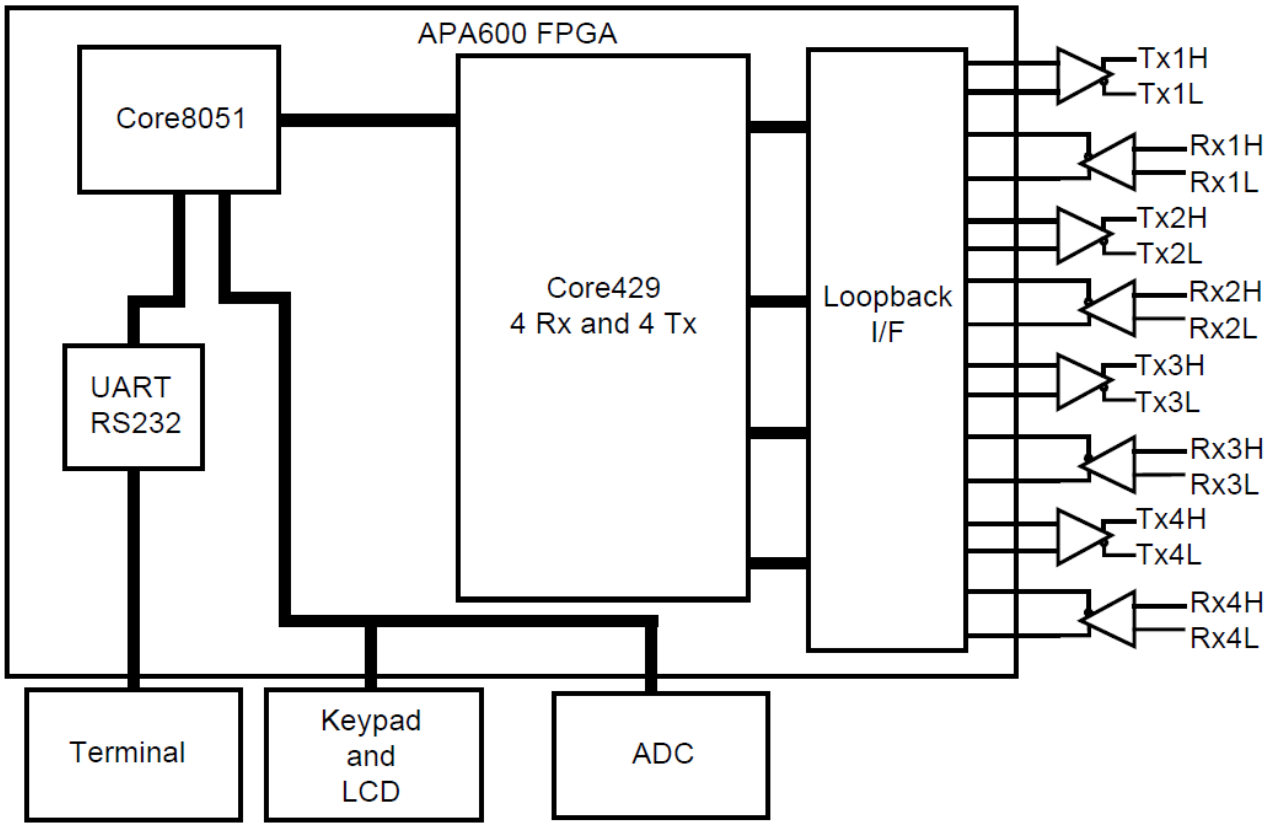

Figure 7 Core429 Development System

The loopback interface logic allows the ARINC core to operate in loopback mode. The development kit includes ARINC line drivers and line receivers. On power-up, Core8051 reads the message from the ADC, which could be the aircraft fuel level or flap position, for example, and transmits it over the transmit channel. The message will be transmitted to the receiver through the loopback interface. Then the message will be retrieved by Core8051 from the receiver and displayed on the LCD.

Another method is to transmit the ADC message over the transmit channel through the line drivers to another system similar to the one described above. The message will go through the receive channel of the second system and can be displayed on the LCD.

Operation

Default Mode Operation

In the default mode, the core operates with the following register map:

CPU Address Map

Address bits 0 and 1 are used to create byte indexes.

- For an 8-bit CPU data bus:
  - 00 – Byte 0
  - 01 – Byte 1
  - 10 – Byte 2
  - 11 – Byte 3
- For a 16-bit CPU data bus:
  - 00 – Lower half word
  - 10 – Upper half word
- For a 32-bit CPU data bus:
  - 00 – Word

Address bits 2 and 3 select the registers within each Rx or Tx block (see Register Definitions).

Address bit 4 is used to determine Rx/Tx as follows: 0: Rx

1: Tx

Address bits 5, 6, 7, and 8 are used for decoding the 16 channels as follows:

0000: Channel0

0001: Channel1

. .

. .

1110: Channel14

1111: Channel15

Table 6 shows the CPU address bit information.

Table 6 CPU Address Bit Positions

Channel Number				Tx/Rx	Register Index		Byte Index	
8	7	6	5	4	3	2	1	0
MSB	9-Bit CPU Address							LSB

Register Definitions

Rx Registers

Following is the detailed definition of cpu_add[3:2] decoding and the explanation of the Data Register, Control Register, Status Register, and Label Memory Register (Table 7 through Table 10).

Address map:

00 – Data Register

01 – Control Register

10 – Status Register

11 – Label Memory

Table 7 Rx Data Register

Bit	Function	Reset State	Type	Description
31:0	Data	0	R	Read data

Table 8 Rx Control Register

Bit	Function	Reset State	Type	Description
0	Data rate	0	R/W	Data rate: 0 = 100 kbps; 1 = 12.5 or 50 kbps
1	Label recognition	0	R/W	Label compare: 0 = disable; 1 = enable
2	Enable 32nd bit as parity	0	R/W	0 = 32nd bit is data; 1 = 32nd bit is parity
3	Parity	0	R/W	Parity: 0 = odd; 1 = even
4	Decoder	0	R/W	0: SDI bit comparison disabled 1: SDI bit comparison enabled; ARINC bits 9 and 10 must match bits 5 and 6, respectively.
5	Match header bit 9	0	R/W	If bit 4 is set, this bit should match ARINC header bit 9 (SDI bit).
6	Match header bit 10	0	R/W	If bit 4 is set, this bit should match ARINC header bit 10 (SDI bit).
7	Reload label memory	0	R/W	When bit 7 is set to '1', label memory address pointers are initialized to '000'. Set this bit to change the contents of the label memory.

Table 9 Rx Status Register

Bit	Function	Reset State	Type	Description
0	FIFO empty	1	R	0 = not empty; 1 = empty
1	FIFO half full or configured level	0	R	0 = less than configured level; 1 = FIFO is filled at least up to configured level
2	FIFO full	0	R	0 = not full; 1 = full

Table 10 Rx Label Memory Register

Bit	Function	Reset State	Type	Description
7:0	Label	0	R/W	Read/write labels

Tx Registers

Following is a detailed definition of cpu_add[3:2] decoding and an explanation of the Data Register, Pattern RAM, Control Register, and Status Register.

Address map:

00 – Data Register

01 – Control Register

10 – Status Register

11 – Unused

Table 11 Tx Data Register

Bit	Function	Reset State	Type	Description
31:0	Data	0	W	Write Data

Table 12 Tx Control Register

Bit	Function	Reset State	Type	Description
0	Data rate	0	R/W	Data rate: 0 = 100 kbps; 1 = 12.5 or 50 kbps
1	Loopback	0	R/W	0 = disable loopback; 1 = enable loopback
2	Enable 32nd bit as parity	0	R/W	0 = 32nd bit is data; 1 = 32nd bit is parity
3	Parity	0	R/W	Parity: 0 = odd; 1 = even

Table 13 Tx Status Register

Bit	Function	Reset State	Type	Description
0	FIFO empty	1	R	0 = not empty; 1 = empty
1	FIFO half full or configured level	0	R	0 = less than half full or configured level; 1 = half full or configured level
2	FIFO full	0	R	0 = not full; 1 = full

Label Memory Operation

The label memory is implemented using an internal memory block. The read and write addresses are generated by internal counters, which can be reset by setting bit 7 of the receive (Rx) control register to '1'. The write counter increments each time the label memory register is written. The read counter increments every time the label memory register is read.

The label memory operation is shown in Figure 8.

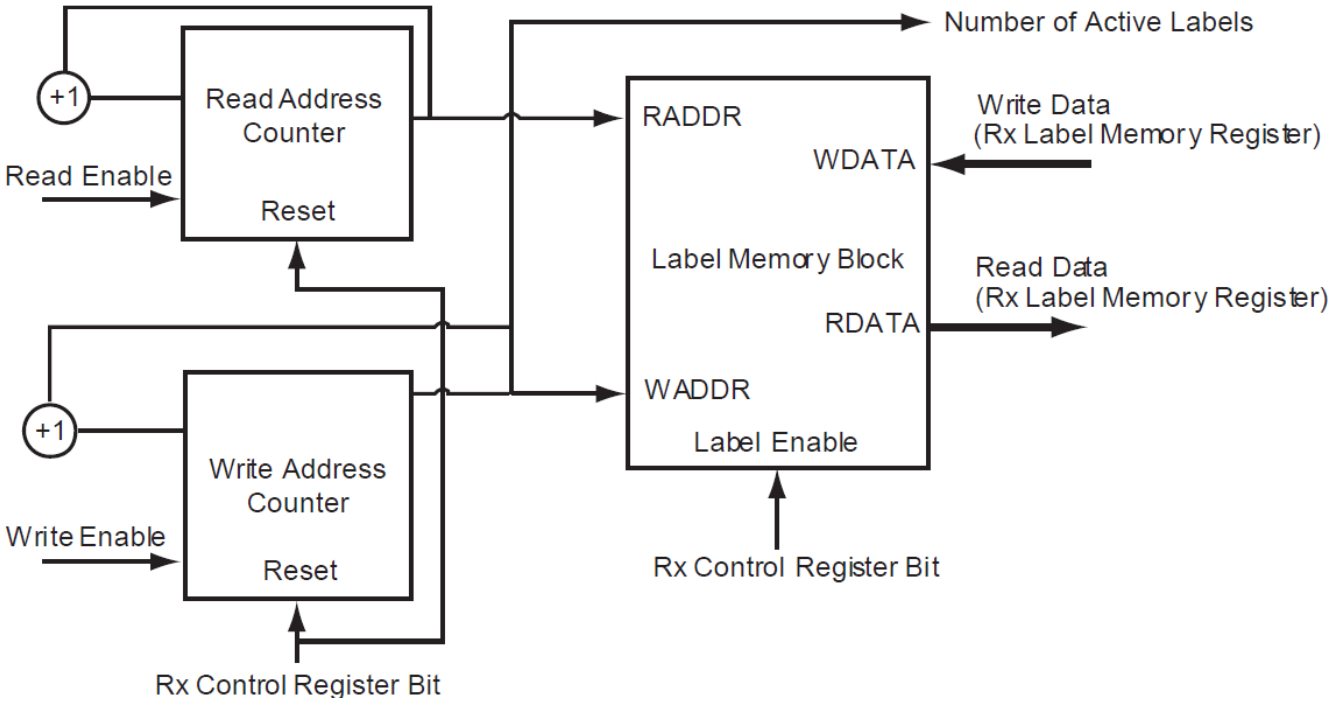

Figure 8 Label Memory Diagram

To program labels, the CPU first resets the read and write counters by setting bit 7 of the receive (Rx) control register to '1'. Then the labels are written to the label memory. The core will compare the incoming ARINC word label (bits 1–8 of ARINC word) against the labels contained in the label memory. The contents of the label memory can be read by reading the label memory register. While writing to or reading from label memory, bit 1 of the receive (Rx) control register should be set to '0'.

To reload the label memory, set bit 7 of the receive (Rx) control register to '1'. The core will then ignore all previous labels, and new labels can be written to the label memory.

Loopback Interface

If the loopback bit in the transmit control register is enabled, the transmit outputs will be connected to the receive inputs. If there are equal numbers of transmit and receive channels, each transmit channel output is connected to the corresponding receive channel input. As an example, the transmit channel 0 output is connected to the receive channel 0 input.

If there are more receive channels than transmit channels, the extra receive channels are connected to transmit channel 0. As an example, if there are two transmit channels (0 and 1) and four receive channels (0, 1, 2, and 3), the connections are made as follows:

- Connect transmit channel 0 output to receive channel 0 input
- Connect transmit channel 1 output to receive channel 1 input
- Connect transmit channel 0 output to receive channel 2 input
- Connect transmit channel 0 output to receive channel 3 input

Interface Description

Configuration Parameters

Core429 has several top-level Verilog parameters (VHDL generics) that are used to select the number of channels and FIFO sizes of the implemented core. Using these parameters allows the size of the core to be reduced when all the channels are not required.

For RTL versions, the parameters in Table 6 can be set directly. For netlist versions of the core, a netlist implementing four Tx and four Rx channels is provided as per the defaults above. Microsemi will supply netlists with alternative parameter settings on request.

Table 14 FIFO and Label Parameters

Parameter Name	Description	Allowed Values	Default	Note
FAMILY	Must be set to the required FPGA family.	11 to 26	15	
CLK_FREQ	Clock frequency	0: 1 MHz 1: 10 MHz 2: 16 MHz 3: 20 MHz 4: 66 MHz	0	(2)
CPU_DATA_WIDTH	CPU data bus width	8, 16, 32 bits	8	
RXN	Number of Rx channels	1 to 16	4	
TXN	Number of Tx channels	1 to 16	4	
LABEL_SIZE_i	Number of labels for Rx channel i	1 to 256	64	(1) & (2)
RX_FIFO_DEPTH_j	Depth of FIFO for Rx channel j ARINC word	32, 64, 128, 256, 512	32	(1)
RX_FIFO_LEVEL_k	FIFO level for Rx channel k	1 to 64	16	(1)
TX_FIFO_DEPTH_l	Depth of FIFO for Tx channel l ARINC word	32, 64, 128, 256, 512	32	(1)
TX_FIFO_LEVEL_m	FIFO level for Tx channel m	1 to 64	16	(1)
TXRXSPEED_n	When this parameter is set to '1', a bit rate of 100/50 kbps is selected. Otherwise, a bit rate of 100/12.5 kbps is selected. The bit rate can be changed for the Rx/Tx channel pair. Refer to the Tx and Rx control register bit descriptions in Table 6-3 on page 26 and Table 6-7 on page 27.	0, 1	0	(1) & (2)
TXRXRESET	When this parameter is set to '1', soft reset of individual Tx/Rx channels is possible by setting bit 7 of a desired channel's control register to '1' temporarily. When this parameter is set to '0', only the input RESET_N can reset the core.	0, 1	0	

Notes:

1. *The parameters i, j, k, l, m, and n can have values from 0 to 15.*
2. *The parameter/generic TXRXRESET can only be set from the core429_default file.*

I/O Signal Descriptions

ARINC Interface

Table 15 Clock and Reset

Name	Type	Description
clk	In	Master clock input (1, 10, 16, or 20 MHz)
reset_n	In	Active low asynchronous reset
txa[TXN–1:0]	Out	ARINC transmit output A
txb[TXN–1:0]	Out	ARINC transmit output B
rxa[RXN–1:0]	In	ARINC receiver input A
rxb[RXN–1:0]	In	ARINC receiver input B

Default Mode Signals

Table 16 Core Interface Signals

Name	Type	Description
int_out_rx[RXN–1:0]	Out	Interrupt from each receive channel. This interrupt is generated when one of the following conditions occurs: • FIFO empty • FIFO full • FIFO is full up to the configured RX_FIFO_LEVEL This is an active high signal.
int_out_tx[TXN–1:0]	Out	Interrupt from each transmit channel. This interrupt is generated when one of the following conditions occurs: • FIFO empty • FIFO full • FIFO is full up to the configured TX_FIFO_LEVEL This is an active high signal.
rx_fifo_full[RXN–1:0]	Out	Rx FIFO full flag for each receive channel. This is an active high signal.
rx_fifo_half_full[RXN–1:0]	Out	Rx FIFO configured level flag for each receive channel. By default it is programmed to half full. This is an active high signal.
rx_fifo_empty[RXN–1:0]	Out	Rx FIFO empty flag for each receive channel. This is an active high signal.
tx_fifo_full[TXN–1:0]	Out	Tx FIFO full flag for each transmit channel. This is an active high signal.
tx_fifo_half_full[TXN–1:0]	Out	Tx FIFO configured level flag for each transmit channel. By default it is programmed to half full. This is an active high signal.
tx_fifo_empty[TXN–1:0]	Out	Tx FIFO empty flag for each transmit channel. This is an active high signal.

CPU Interface

The CPU interface allows access to the Core429 internal registers, FIFO, and internal memory. This interface is synchronous to the clock.

Table 17 CPU Interface Signals

Name	Type	Description
cpu_ren	In	CPU read enable, active low
cpu_wen	In	CPU write enable, active low
cpu_add[8:0]	In	CPU address
cpu_din[CPU_DATA_WIDTH–1:0]	In	CPU data input
cpu_dout[CPU_DATA_WIDTH–1:0]	Out	CPU data output
int_out	Out	Interrupt to CPU, active high. int_out is the OR function of int_out_rx and int_out_tx.
cpu_wait	Out	Indicates that the CPU should hold cpu_ren or cpu_wen active while the core completes the read or write operation.

Timing Diagrams

CPU Interface Timing for Default Mode

The CPU interface signals are synchronized to the Core429 master clock. Figure 9 through Figure 16 show the waveforms for the CPU interface.

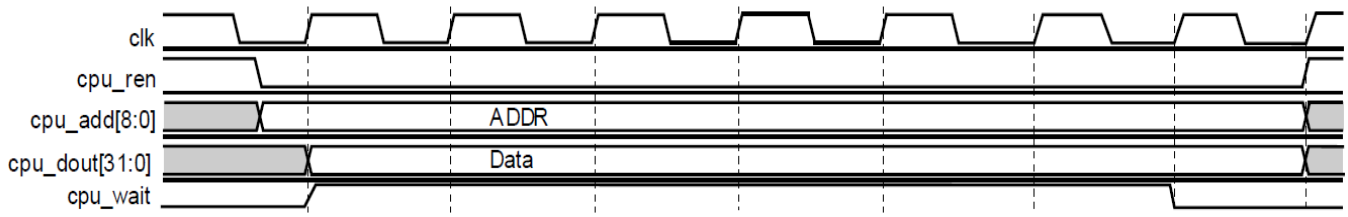

Figure 9 CPU Interface Control/Status Register Read Cycle

Notes:

1. cpu_ren should be deasserted on the next clock cycle after cpu_wait is deasserted. Read data is available one cycle after cpu_ren is sampled.
2. The cpu_wait signal will assert for a minimum of six clock cycles during a read cycle.

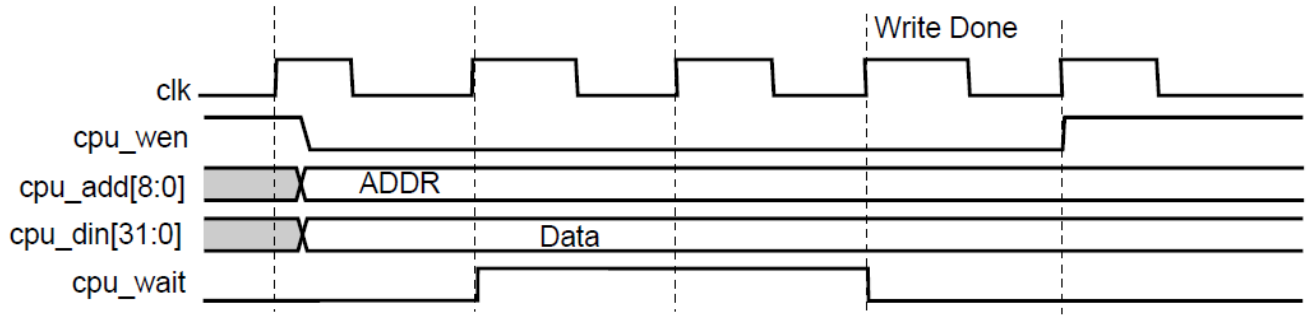

Figure 10 CPU Interface Control Register Write Cycle

Note: cpu_wen should be deasserted on the next clock cycle after cpu_wait is deasserted. The write is done two cycles after cpu_wen is sampled.

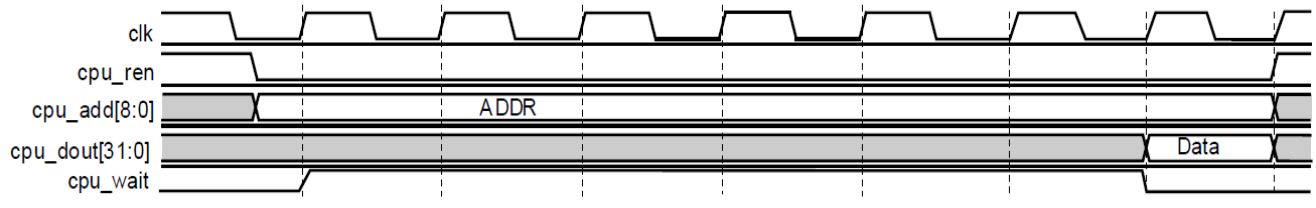

Figure 11 CPU Interface Data Register Read Cycle

Note: cpu_ren should be deasserted on the next clock cycle after cpu_wait is deasserted. Read data is available six cycles after cpu_ren is sampled.

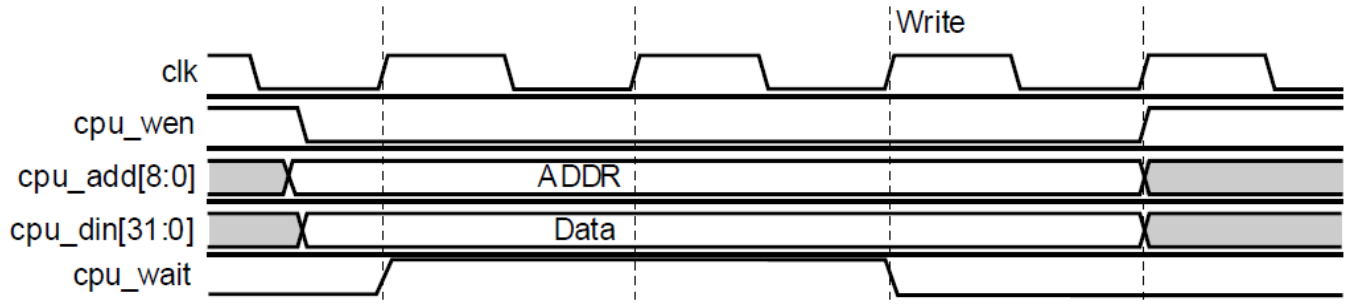

Figure 12 CPU Interface Data Register Write Cycle

Note: cpu_wen should be deasserted on the next clock cycle after cpu_wait is deasserted. The write is done two cycles after cpu_wen is sampled.

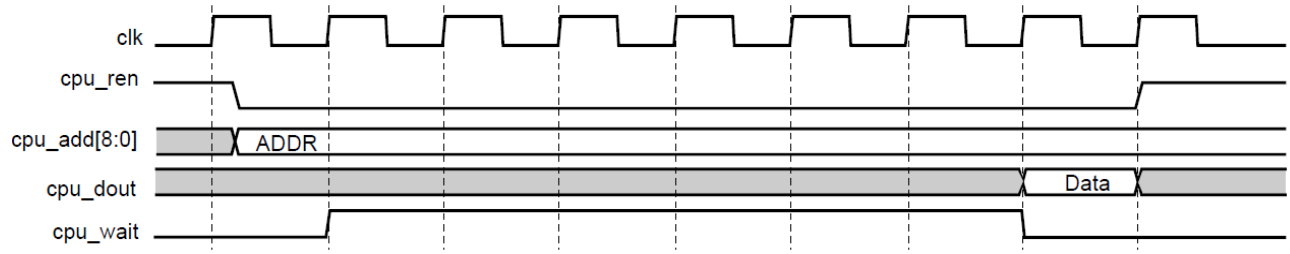

Figure 13 CPU Interface Label Memory Read Cycle

Note: cpu_ren should be deasserted on the next clock cycle after cpu_wait is deasserted. Read data is available six cycles after cpu_ren is sampled.

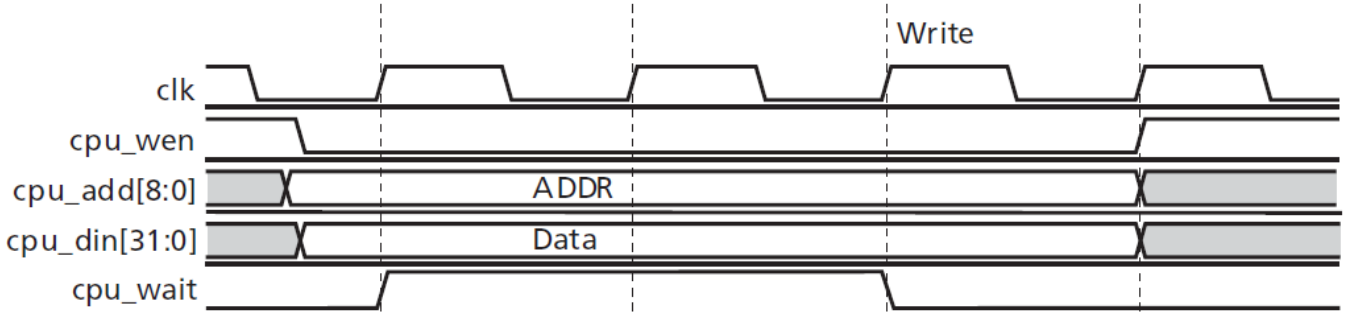

Figure 14 CPU Interface Label Memory Write Cycle

Note: cpu_wen should be deasserted on the next clock cycle after cpu_wait is deasserted. The write is done two cycles after cpu_wen is sampled.

Tool Flows

Core429 is licensed in two ways.

License Types

Obfuscated

Complete RTL code is provided for the core, enabling the core to be instantiated with SmartDesign. Simulation, Synthesis, and Layout can be performed with Libero software. The RTL code for the core is obfuscated, and some of the testbench source files are not provided; they are precompiled into the compiled simulation library instead.

RTL

Complete RTL source code is provided for the core and testbenches.

SmartDesign

Core429 is available through the Libero SoC IP Catalog. Download it from a remote web-based repository and install into your local vault to make it ready to use. Once installed in the Libero software, the core can be instantiated, configured, connected, and generated using the SmartDesign tool.

For more information on using SmartDesign to instantiate and generate cores, refer to the *Using DirectCore in Libero® System-on-Chip (SoC) User Guide* or consult the Libero SoC online help.

The core can be configured using the configuration GUI within SmartDesign, as shown in Figure 16 through Figure 18.

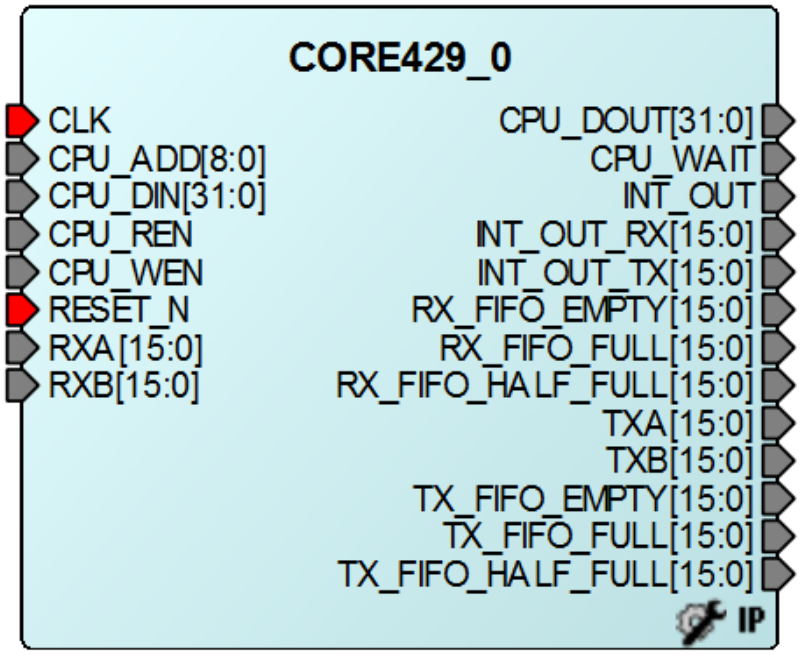

Figure 15 Core429 Full I/O View

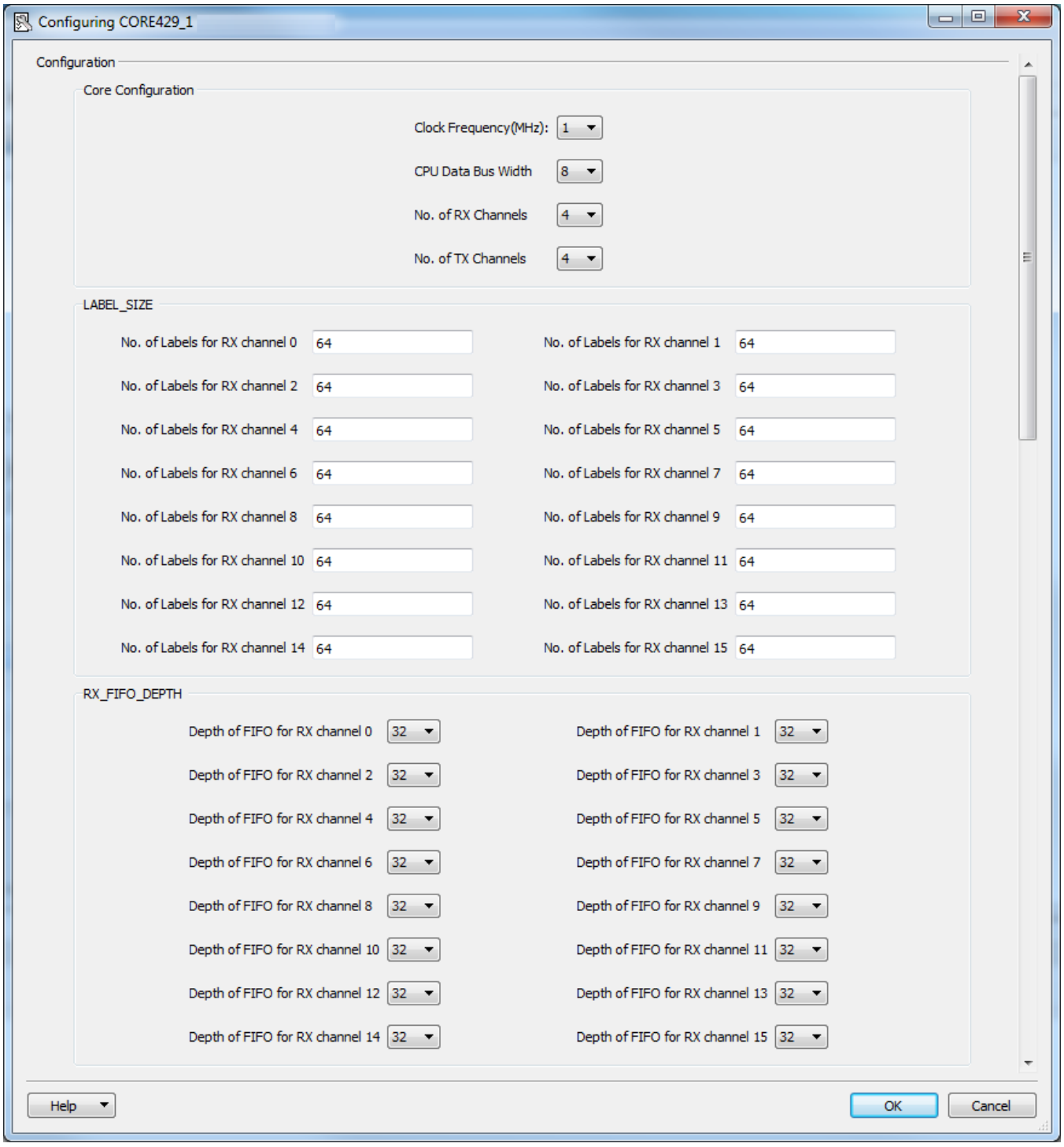

Figure 16 Core429 SmartDesign Configuration

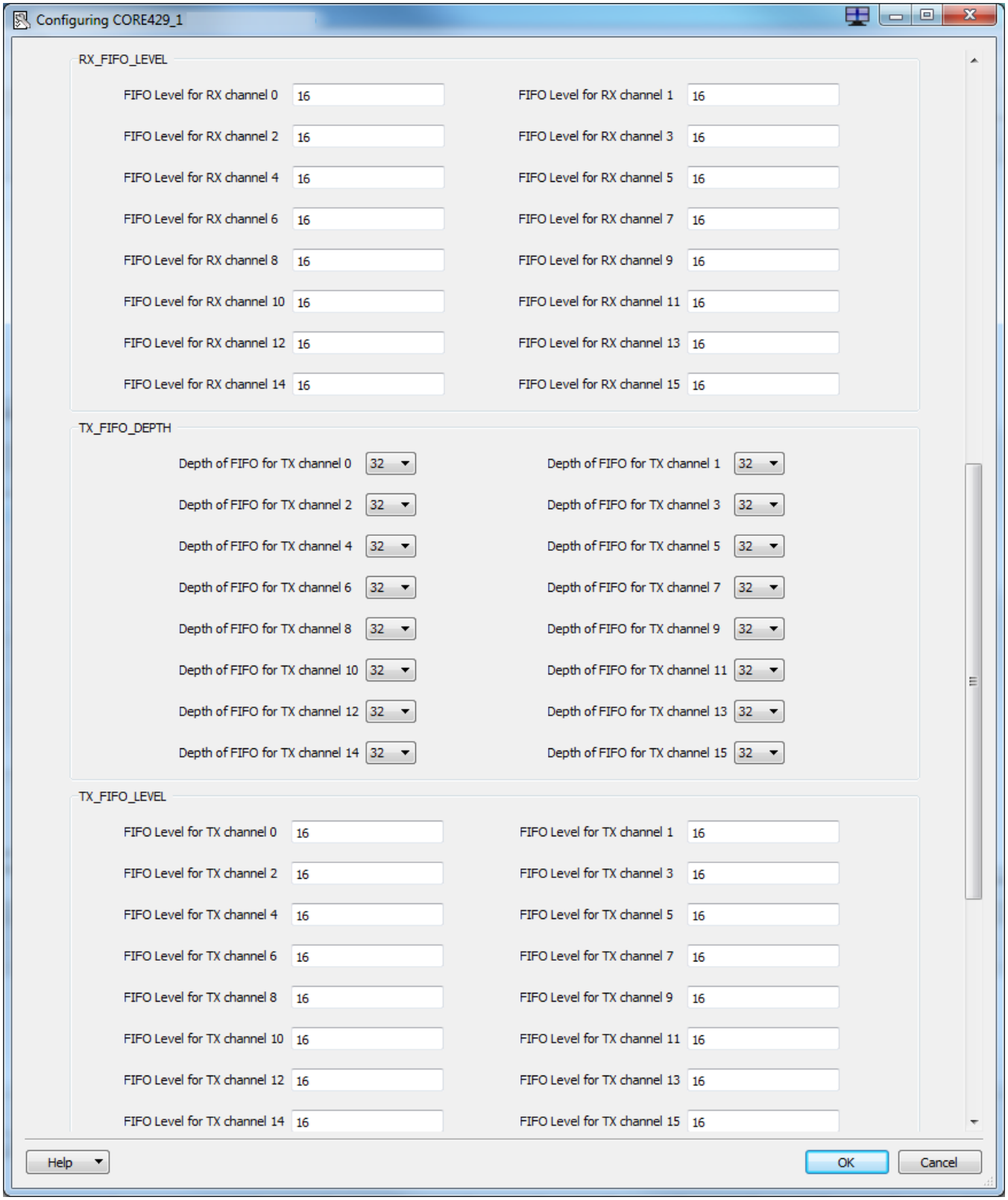

Figure 17 Core429 SmartDesign Configuration (continued)

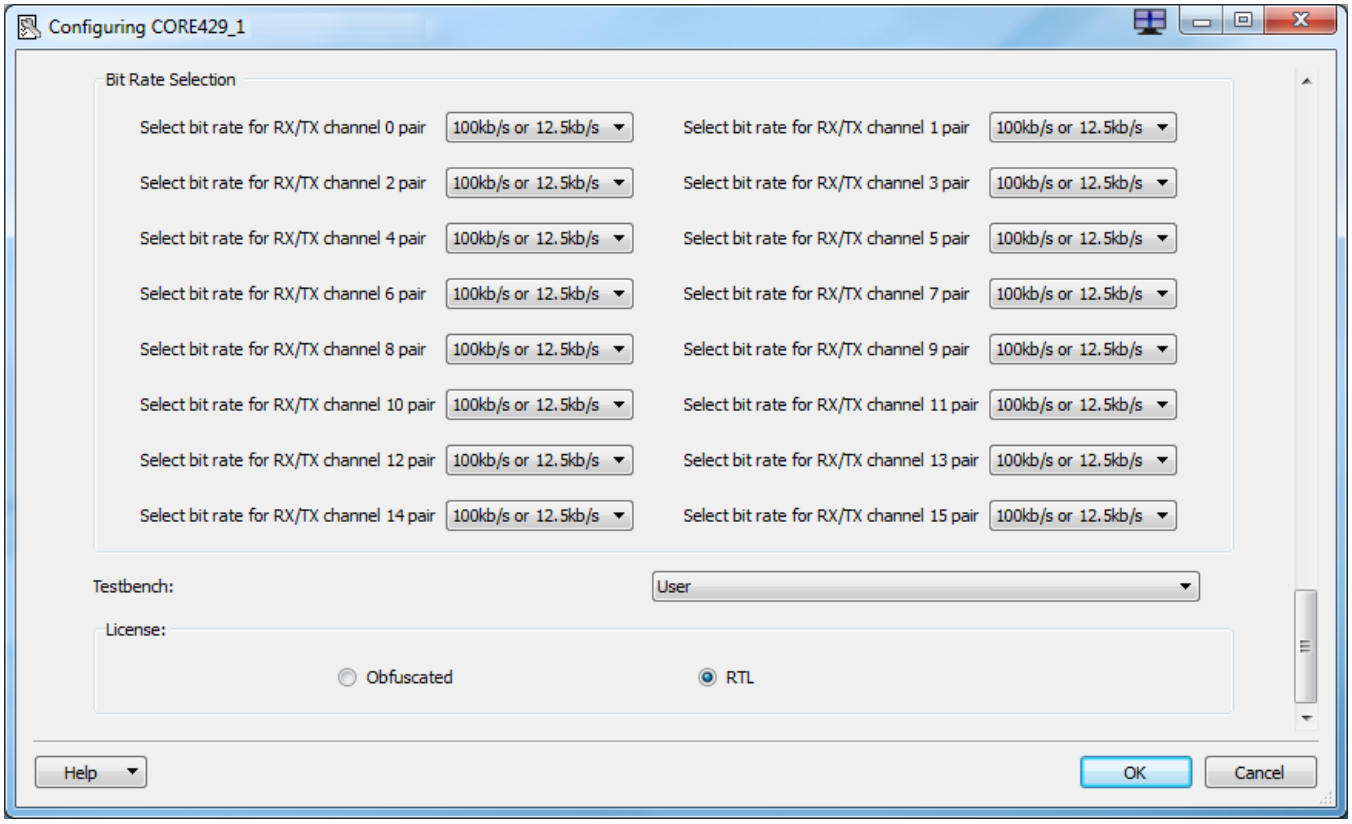

Figure 18 Core429 SmartDesign Configuration (continued)

After configuring the core, Microsemi recommends you use the top-level Auto Stitch function to connect all the core interface signals to the top level of the SmartDesign project.

Simulation Flows

To run simulations, select the User Testbench flow within SmartDesign and click **Save & Generate** on the Generate window. The User Testbench is selected through the Core Testbench Configuration GUI.

When SmartDesign generates the Libero IDE project, it will install the user testbench files.

To run the user testbench, set the design root to the **Core429 instantiation** in the Libero design hierarchy window and click the **Simulation** icon in the Libero Design Flow window. This will invoke ModelSim® and automatically run the simulation.

Synthesis in Libero SoC

To run synthesis on Core429, set the design root to the IP component instance and run the synthesis tool from the Libero design flow window.

Place-and-Route in Libero SoC

After the design is synthesized, run the compilation and then place-and-route the tools. The Core429 does not require any additional place-and-route settings.

Testbench Operation and Modification

Two testbenches are provided with Core429:

- **VHDL user testbench:** Simple-to-use testbench written in VHDL. This testbench is intended for customer modification.
- **Verilog user testbench:** Simple-to-use testbench written in Verilog. This testbench is intended for customer modification.

Testbench

The CPU model sets up Core429 via the CPU interface and loads the transmit data. The transmit data will be sent to the receiver. The CPU model can retrieve the receive data through the CPU interface and compare it against the transmitted data.

The user testbenches are intended to simplify core integration into the target system (Figure 19). This consists of the core connections to a CPU model and loopback logic that connects the Tx output to the Rx input.

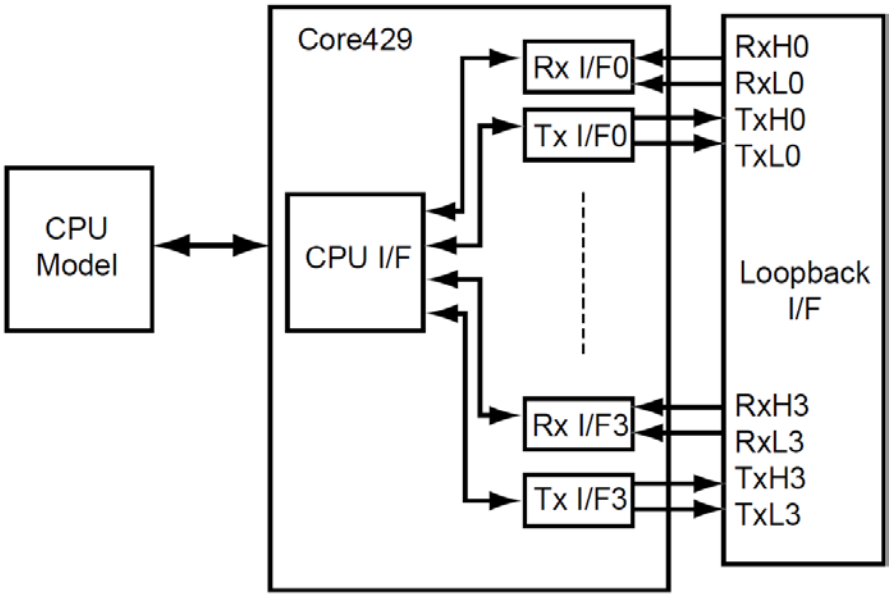

Figure 19 Test-bench Diagram

User Test-bench

An example user testbench is included with the Evaluation, Obfuscated, and RTL releases of Core429. The user testbench is provided in precompiled Model*Sim* format and in RTL source code for all releases (Obfuscated and RTL) for you to examine and modify to suit your needs. The source code for the user testbench is provided to ease the process of integrating the Core429 macro into your design and verifying according to your own custom needs. A block diagram of the user testbench is shown in Figure 20.

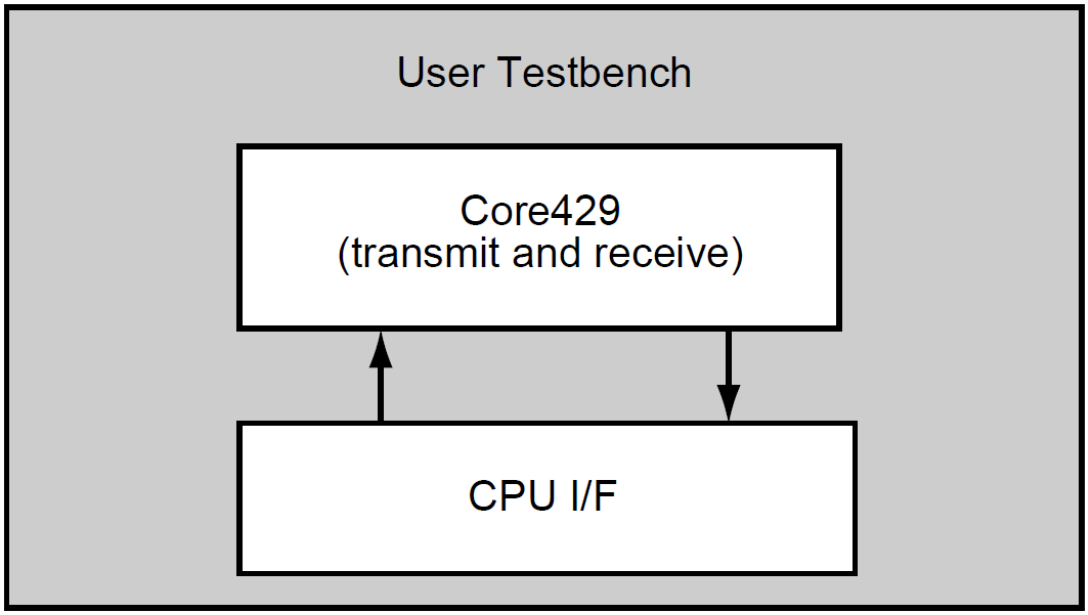

Figure 20 Core429 User Test-bench

The user testbench includes a simple example design that serves as a reference for users who want to implement their own designs. RTL source code for the user testbench shown in Figure 20 is included in the source directory for the Obfuscated and RTL releases of Core429. The example user testbench files are *$CORE429/source/user_tbench.vhd and $CORE429/source/user_tbench.v.*

The testbench for the example user design implements a subset of the functionality tested in the verification testbench, described in Configuration Parameters. As shown in Figure 20, Core429 is instantiated in the user testbench, and the CPU interface drives the data into the transmitter. The user testbench causes exchanges of data to take place between the transmitter and receiver. The user testbench verifies the sent data by receiving it from the receiver.

Ordering Information

Ordering Codes

Core429 can be ordered through your local Sales Representatives. It must be ordered using the following number scheme: Core429-XX, where XX is listed in Table 18.

Table 18 ·Ordering Codes

XX	Description
OM	RTL for Obfuscated—RTL Multiple use license
RM	RTL for RTL source—Multiple-use license

Appendix A: Testbench Support Routines

The verification and user testbenches for the Core429 macro make use of support routines, both in VHDL and Verilog. The support routines are described in this appendix for the VHDL and Verilog testbenches.

VHDL Support

The VHDL versions of the testbenches make use of the following procedures, which are included within the top-level module of the verification and user testbenches.

The following procedure does a simple read from the Core429 registers or memory:

```
PROCEDURE cpu_read (
    address      :IN std_logic_vector(8 DOWNTO 0);
    SIGNAL data  :OUT std_logic_vector(CPU_DATA_WIDTH - 1 DOWNTO 0))
```

The following procedure does a simple write to the Core429 registers or memory:

```
PROCEDURE cpu_write (
    address      :IN std_logic_vector(8 DOWNTO 0);
    data         :IN std_logic_vector(CPU_DATA_WIDTH - 1 DOWNTO 0))
```

Verilog Support

The Verilog versions of the testbenches make use of the following tasks, which are included within the top-level module of the verification and user testbenches.

The following procedure does a simple read from the Core429 registers or memory:

```
task cpu_read;
input [8:0] address;
output [CPU_DATA_WIDTH-1:0] data; reg
[CPU_DATA_WIDTH-1:0] data;
```

The following procedure does a simple write to the Core429 registers or memory:

```
task cpu_write; input
[8:0] address;
input [CPU_DATA_WIDTH-1:0] data;
```

List of Changes

The following table shows important changes made in this document for each revision.

Date and Revision	Change	Page
Revision 6 (April 2016)	Updated the version number v3.12.	N/A
Revision 5 (August 2015)	Updated the version number v3.11 and applied new template	N/A
Revision 4 (April 2014)	Updated Table 3-1.	19
	Updated Table 6-3.	26
	Updated "Label Memory Operation" section.	28
Revision 3 (February 2014)	Added IGLOO2 to Table 1, Table 2, Table 3, Table 4, and Table 5.	7, 8, 9
	Added IGLOO2, SmartFusion and SmartFusion2 to Table 6.	9
	Updated Table 3.1 - FIFO and Label Parameters	19
	Updated Table 6.4 - Rx Status Register	26
	Updated Table 6.8 - Tx Status Register	27
	Removed all Verification Testbench information because it is no longer included with the core.	31, 32
	Removed all Legacy Mode information, this mode is now obsolete so it is no longer included with the core.	Multi
Revision 2 (February 2013)	Table 1 • Device Utilization for One Tx Module (default mode) through Table 5 • Device Utilization for Legacy Mode (two Rx and one Tx) were revised to include values for SmartFusion and SmartFusion2.	7, 9
	The words "programmable" or "programmed" were changed to "configurable" or "configured" in several places throughout the document, regarding Core429 clock rate, receiver FIFO, RX_FIFO_LEVEL, TX_FIFO_LEVEL, Rx FIFO level flag, TxFIFO level flag, and FIFO half full.	Multi
	RxLo and RxHi were corrected to TxLo and TxHi in Figure 1-2 • Core429 Tx Block Diagram.	12
	The following note was added with regard to Legacy mode: Legacy is now obsolete; however, the RT is still available with this core.	12
	Rx I/F and Tx I/F had been mislabeled in Figure 1-3 • Core429 Line Driver and Line Receiver Interface. This was corrected.	13
	The Evaluation license option was removed from the "Tool Flows" section.	15
	The TXRXRESET parameter was added to Table 3-1 • FIFO and Label Parameters. Notes were added to the table regarding the limit of the label recognition feature and setting the TXRXRESET parameter.	19
	Figure 5-1 • CPU Interface Control/Status Register Read Cycle was replaced.	23
Revision 1 (April 2007)	Revised the allowed values and default columns for CLK_FREQ in Table 3-1 • FIFO and Label Parameters.	19

Product Support

Microsemi SoC Products Group backs its products with various support services, including Customer Service, Customer Technical Support Center, a website, electronic mail, and worldwide sales offices. This appendix contains information about contacting Microsemi SoC Products Group and using these support services.

Customer Service

Contact Customer Service for non-technical product support, such as product pricing, product upgrades, update information, order status, and authorization.

From North America, call **800.262.1060**
From the rest of the world, call **650.318.4460**
Fax, from anywhere in the world **650. 318.8044**

Customer Technical Support Center

Microsemi SoC Products Group staffs its Customer Technical Support Center with highly skilled engineers who can help answer your hardware, software, and design questions about Microsemi SoC Products. The Customer Technical Support Center spends a great deal of time creating application notes, answers to common design cycle questions, documentation of known issues and various FAQs. So, before you contact us, please visit our online resources. It is very likely we have already answered your questions.

Technical Support

For Microsemi SoC Products Support, visit http://www.microsemi.com/products/fpga-soc/design-support/fpga-soc-support.

Website

You can browse a variety of technical and non-technical information on the Microsemi SoC Products Group home page, at http://www.microsemi.com/soc/.

Contacting the Customer Technical Support Center

Highly skilled engineers staff the Technical Support Center. The Technical Support Center can be contacted by email or through the Microsemi SoC Products Group website.

Email

You can communicate your technical questions to our email address and receive answers back by email, fax, or phone. Also, if you have design problems, you can email your design files to receive assistance. We constantly monitor the email account throughout the day. When sending your request to us, please be sure to include your full name, company name, and your contact information for efficient processing of your request.

The technical support email address is soc_tech@microsemi.com.

My Cases

Microsemi SoC Products Group customers may submit and track technical cases online by going to My Cases.

Outside the U.S.

Customers needing assistance outside the US time zones can either contact technical support via email (soc_tech@microsemi.com) or contact a local sales office. Sales office listings can be found at www.microsemi.com/soc/company/contact/default.aspx.

ITAR Technical Support

For technical support on RH and RT FPGAs that are regulated by International Traffic in Arms Regulations (ITAR), contact us via soc_tech_itar@microsemi.com. Alternatively, within My Cases, select **Yes** in the ITAR drop-down list. For a complete list of ITAR-regulated Microsemi FPGAs, visit the ITAR web page.

Microsemi Corporate Headquarters
One Enterprise, Aliso Viejo,
CA 92656 USA

Within the USA: +1 (800) 713-4113
Outside the USA: +1 (949) 380-6100
Sales: +1 (949) 380-6136
Fax: +1 (949) 215-4996

E-mail: sales.support@microsemi.com

Microsemi Corporation (Nasdaq: MSCC) offers a comprehensive portfolio of semiconductor and system solutions for communications, defense and security, aerospace, and industrial markets. Products include high-performance and radiation-hardened analog mixed-signal integrated circuits, FPGAs, SoCs, and ASICs; power management products; timing and synchronization devices and precise time solutions; voice processing devices; RF solutions; discrete components; enterprise storage and communications solutions, security technologies, and scalable anti-tamper products; Ethernet solutions; Power-over-Ethernet ICs and midspans; custom design capabilities and services. Microsemi is headquartered in Aliso Viejo, California, and has approximately 4,800 employees worldwide. Learn more at **www.microsemi.com**.

50200088-6/04.16