Core429 Handbook

v2.0

Actel®

Actel Corporation, Mountain View, CA 94043

Printed in the United States of America

Part Number: 50200088-0

Release: January 2007

Trademarks

Table of Contents

Introduction

General Description

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" on page 11.

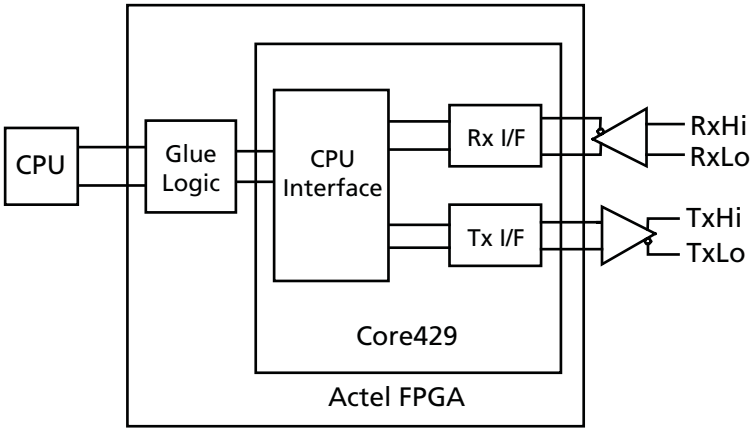

Figure 1. Typical Core429 System—One Tx and One Rx

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.

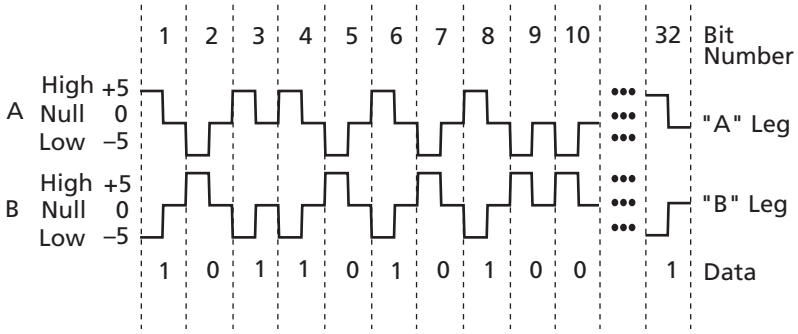

Figure 2. ARINC Standard

Figure 3 shows the bit positions of ARINC data.

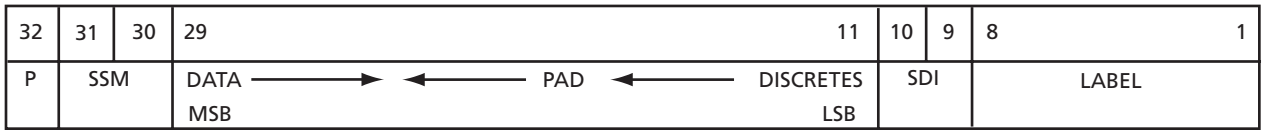

Figure 3. ARINC Data Bit Positions

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.

Core Versions

This handbook applies to Core429 v3.0. The release notes provided with the core list known discrepancies between this handbook and the core release associated with the release notes.

Core429 Device Requirements

Core429 can be implemented in several Actel FPGA devices. Table 1 through Table 5 on page 8 provide typical utilization figures 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)

Family	Cells or Tiles			Memory Blocks	Device	Utilization
	Combinatorial	Sequential	Total			
Fusion	363	147	510	1	AFS600	4%
ProASIC®3/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%

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

Family	Cells or Tiles			Memory Blocks	Devices	Utilization
	Combinatorial	Sequential	Total			
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%

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

Family	Cells or Tiles			Memory Blocks	Device	Utilization
	Combinatorial	Sequential	Total			
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%

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

Family	Cells or Tiles			Memory Blocks	Device	Utilization
	Combinatorial	Sequential	Total			
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%

Table 5. Device Utilization for Legacy Mode (two Rx and one Tx)

Family	Cells or Tiles			Memory Blocks	Device	Utilization
	Combinational	Sequential	Total			
Fusion	1,444	613	2,057	5	AFS600	15%
ProASIC3/E	1,444	613	2,057	5	A3PE600	15%
ProASIC^{PLUS}	1,840	674	2,514	5	APA150	41%
Axcelerator	955	653	1,608	5	RTAX250S	20%
RTAX-S	1,062	729	1,791	5	RTAX250S	42%

The Core429 clock rate can be programmed to be 1, 10, 16, or 20 MHz. All the Actel 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.

Number of memory blocks = NRx * (INT(LABEL_SIZE / X) + INT(RX_FIFO_DEPTH / Y) +
NTx * INT(FIFO_DEPTH / Y),

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 6 on page 9.

Each Channel Configured Differently

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

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

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 6.

Table 6. Memory Parameters

Device Family	X	Y
Fusion	512	128
ProASIC3/E	512	128
ProASIC^{PLUS}	256	64
Axcelerator/RTAX-S	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.

1 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 programmable 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 1-1 gives a functional description of the Rx block.

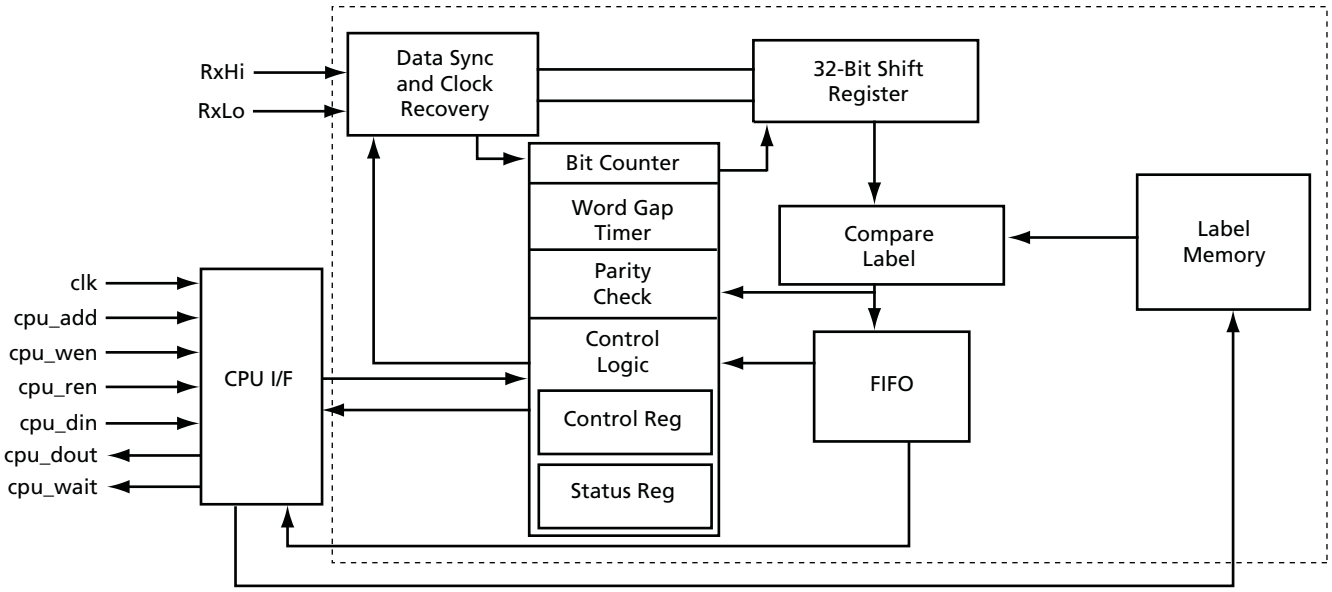

Figure 1-1. Core429 Rx Block Diagram

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 6-3 on page 28 and Table 6-4 on page 28 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 programmed 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 1-2 gives a functional description of the Tx block.

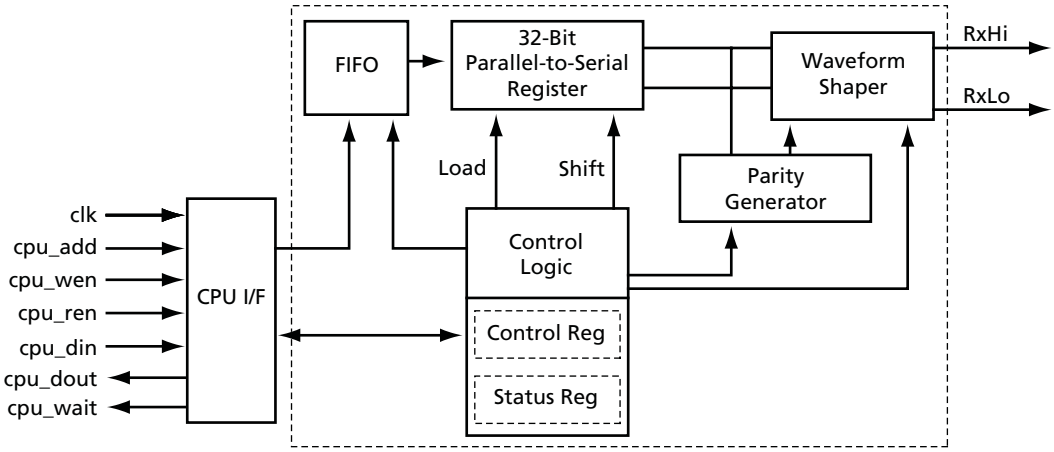

Figure 1-2. Core429 Tx Block Diagram

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 6-7 on page 29 and Table 6-8 on page 29 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 programmed TX_FIFO_LEVEL
- tx_fifo_full: Tx 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_tx is generated when one of the FIFO status signals (tx_fifo_empty, tx_fifo_half_full, or tx_fifo_full) is HIGH.

Legacy Mode

In this mode, there is a legacy interface block that communicates with the CPU interface. When legacy mode is enabled, the core supports two receive (Rx) channels and one transmit (Tx) channel only. This is not configurable.

Clock Requirements

To meet the ARINC 429 transmission bit rate requirements, the Core429 clock input must be 1, 10, 16, or 20 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 1-3 shows the connections required from Core429 to the line drivers.

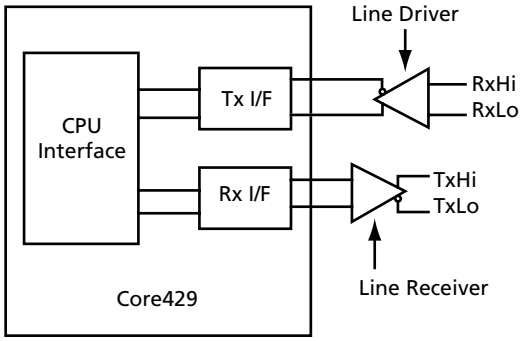

Figure 1-3. Core429 Line Driver and Line Receiver Interface

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, Actel 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 1-4 shows the development system.

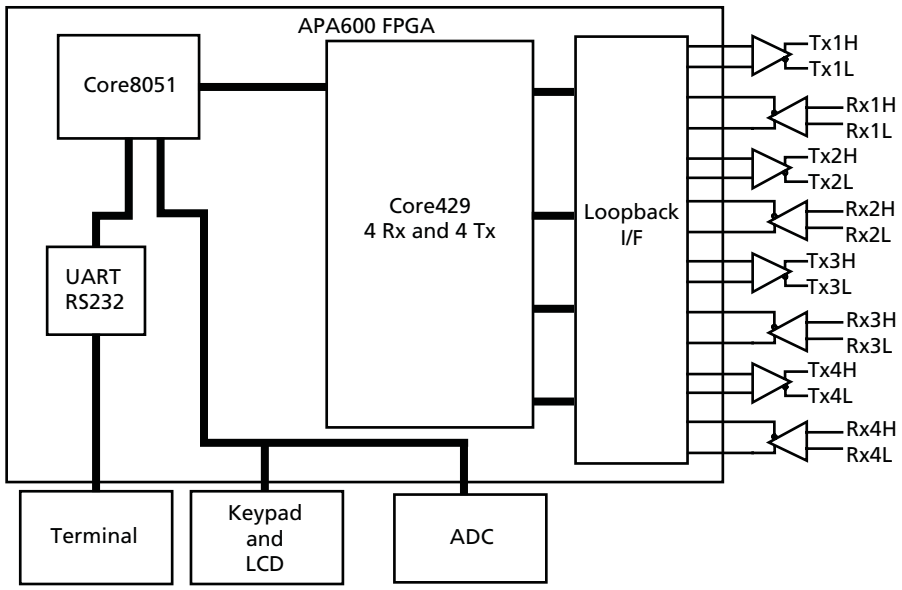

Figure 1-4. Core429 Development System

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.

2 Tool Flows

Core429 is licensed in three ways. Tool flow functionality may be limited.

Evaluation

Precompiled simulation libraries are provided, allowing the core to be instantiated in CoreConsole and simulated within Actel Libero® Integrated Design Environment (IDE), as described in the "CoreConsole" section. Using the Evaluation version of the core, it is possible to create and simulate the complete design in which the core is being included. The design may not be synthesized, as source code is not provided.

Obfuscated

Complete RTL code is provided for the core, enabling the core to be instantiated with CoreConsole. Simulation, Synthesis, and Layout can be performed with Libero IDE. 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.

CoreConsole

Core429 is preinstalled in the CoreConsole IP Deployment Design Environment. To use the core, click and drag it from the IP core list into the main window. The CoreConsole project may be exported to Libero IDE at this point, providing access to the core only. Alternately, IP blocks can be interconnected, allowing the complete system to be exported from CoreConsole to Libero IDE.

The core can be configured using the configuration GUI within CoreConsole, as shown in Figure 2-1 on page 16 through Figure 2-2 on page 17.

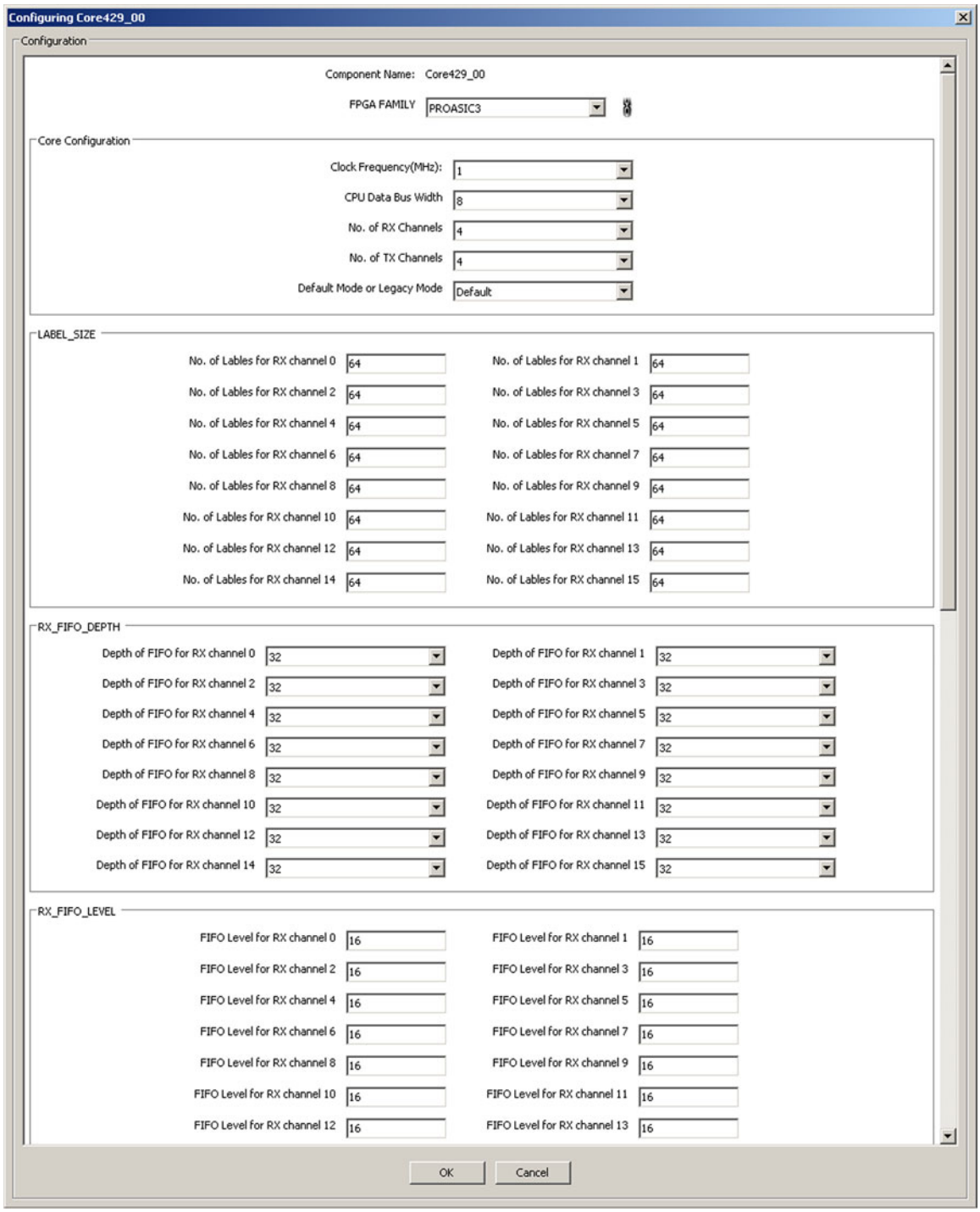

Figure 2-1. Core429 Configuration within CoreConsole

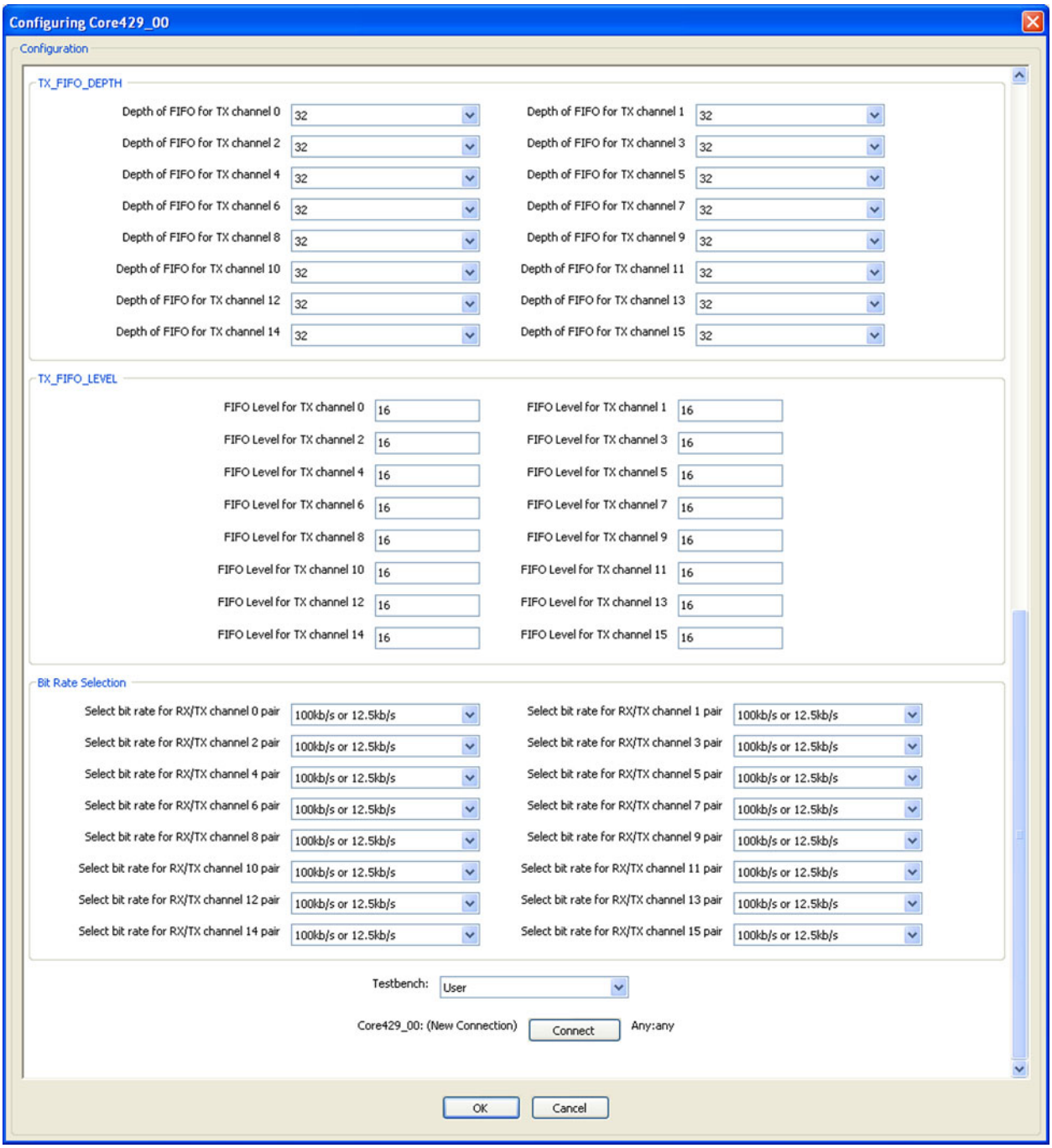

Figure 2-2. Core429 Configuration within CoreConsole (continued)

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

Once the core is configured, invoke the Generate function in CoreConsole. This will export all the required files to the project directory in the *LiberoExport* directory. This is in the CoreConsole installation directory by default.

Importing into Libero IDE

After generating and exporting the core from CoreConsole, the core can be imported into Libero IDE. Create a new project in Libero IDE and import the CoreConsole project from the *LiberoExport* directory. Libero IDE will then install the core and the selected testbenches, along with constraints and documentation, into its project.

Note: If two or more DirectCores are required, they can both be included in the same CoreConsole project and imported into Libero IDE at the same time.

Simulation Flows

To run simulations, the required testbench flow must be selected within CoreConsole, and Save & Generate must be run from the Generate pane. Select the required testbench through the core configuration GUI in CoreConsole. The following simulation environments are supported:

- Full Core429 verification environment (Verilog only)
- Simple testbench (VHDL and Verilog)

When CoreConsole generates the Libero IDE project, it will install the appropriate testbench files. To run the testbenches, simply set the design root to *Core429 instantiation* in the Libero IDE file manager and click the **Simulation** icon in Libero IDE. This will invoke Model*Sim* and automatically run the simulation.

Synthesis in Libero IDE

To run Synthesis on the core with parameters set in CoreConsole, set the design root to the top of the project imported from CoreConsole. This is a wrapper around the core that sets all the generics appropriately. Make sure the required timing constraints files are associated with the synthesis tool.

Click the **Synthesis** icon in Libero IDE. The synthesis window appears, displaying the Synplicity® project. To run Synthesis, click the **Run** icon.

Place-and-Route in Libero IDE

Having set the design route appropriately and run Synthesis, click the **Layout** icon in Libero IDE to invoke Designer. Core429 requires no special place-and-route settings.

3 Core 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 3-1 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. Actel will supply netlists with alternative parameter settings on request.

Table 3-1. FIFO and Label Parameters

Parameter Name	Description	Allowed Values	Default
FAMILY	Must be set to the required FPGA family.	11 to 21	15
CLK_FREQ_	Clock frequency	1, 10, 16, 20 MHz	1 MHz
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
LEGACY_MODE	0 = Default mode; 1 = Legacy mode	0,1	0
LABEL_SIZE_i	Number of labels for Rx channel i	1 to 256	64
RX_FIFO_DEPTH_j	Depth of FIFO for Rx channel j ARINC word	32, 64, 128, 256, 512	32
RX_FIFO_LEVEL_k	FIFO level for Rx channel k	1 to 64	16
TX_FIFO_DEPTH_l	Depth of FIFO for Tx channel l ARINC word	32, 64, 128, 256, 512	32
TX_FIFO_LEVEL_m	FIFO level for Tx channel m	1 to 64	16
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 28 and Table 6-7 on page 29.	0, 1	0

Note: The parameters i, j, k, l, m, and n can have values from 0 to 15.

4

Core Interfaces

I/O Signal Descriptions

ARINC Interface

Table 4-1. 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 4-2. 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 programmed 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 programmed 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 programmed 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.

Table 4-2. Core Interface Signals (Continued)

Name	Type	Description
tx_fifo_half_full[TXN–1:0]	Out	Tx FIFO programmed 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 4-3. 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.

Legacy Interface

The legacy interface allows access to the Core429 internal registers, FIFO, and internal memory. This interface is synchronous to the clock. The Tx module contains two 8-bit registers. One is used for control function and the other is used for status.

Table 4-4. Legacy Interface Signals

Name	Type	Description
data_ready1	Out	Receiver 1 data ready (FIFO not empty) flag
fifo_full1	Out	Receiver 1 FIFO full
half_full1	Out	Receiver 1 FIFO half full
data_ready2	Out	Receiver 2 data ready (FIFO not empty) flag
fifo_full2	Out	Receiver 2 FIFO full
half_full2	Out	Receiver 2 FIFO half full
transmit_fifo_full	Out	Transmit FIFO full
transmit_half_full	Out	Transmit FIFO half full

Table 4-4. Legacy Interface Signals (Continued)

Name	Type	Description
rsel	In	Receiver data half word selection
ctrl_n	In	Clock for control word register
str_n	In	Read status register if rsel = 0, read control register if rsel = 1
entx	In	Enable transmission
txr	Out	Transmitter ready flag. Goes LOW when ARINC word loaded into FIFO. Goes HIGH after transmission and FIFO empty.
pl1_n	In	Latch enable for word 1 entered from data bus to transmitter FIFO
pl2_n	In	Latch enable for word 2 entered from data bus to transmitter FIFO. Must follow pl1_n.
en1_n	In	Data bus control. Enables receiver 1 data to outputs.
en2_n	In	Data bus control. Enables receiver 2 data to outputs if en1_n is HIGH.
test	In	Disables transmitter output if HIGH.
dout	In/Out	Bidirectional data bus
data_valid	Out	Data is valid when data_valid = 1.

5

Timing Diagrams

CPU Interface Timing for Default Mode

The CPU interface signals are synchronized to the Core429 master clock. Figure 5-1 through Figure 5-6 on page 26 show the waveforms for the CPU interface.

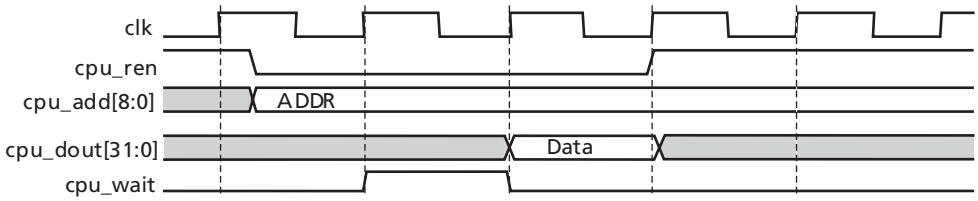

Figure 5-1. CPU Interface Control/Status Register Read Cycle

Note: 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.

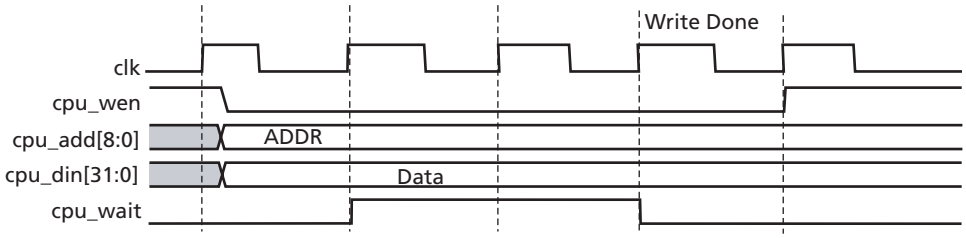

Figure 5-2. CPU Interface Control Register Write Cycle

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.

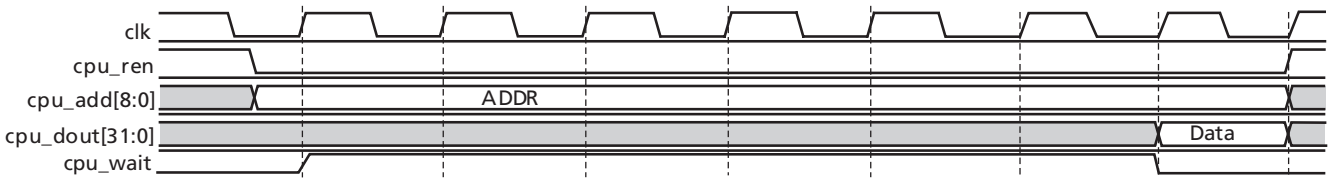

Figure 5-3. CPU Interface Data Register Read Cycle

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.

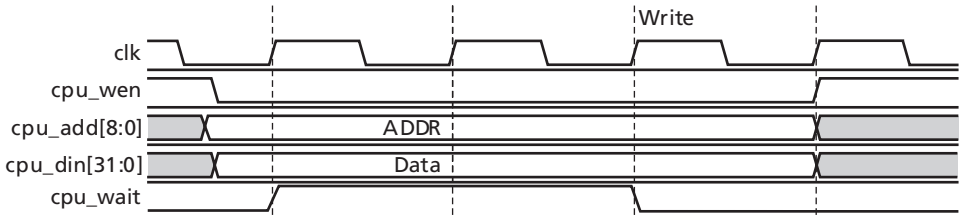

Figure 5-4. CPU Interface Data Register Write Cycle

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.

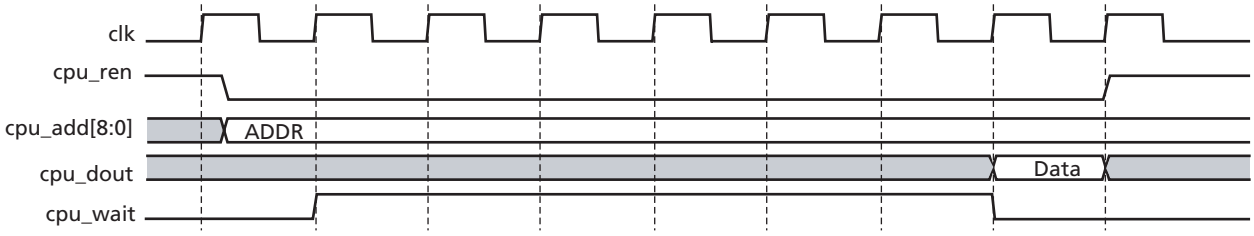

Figure 5-5. CPU Interface Label Memory Read Cycle

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.

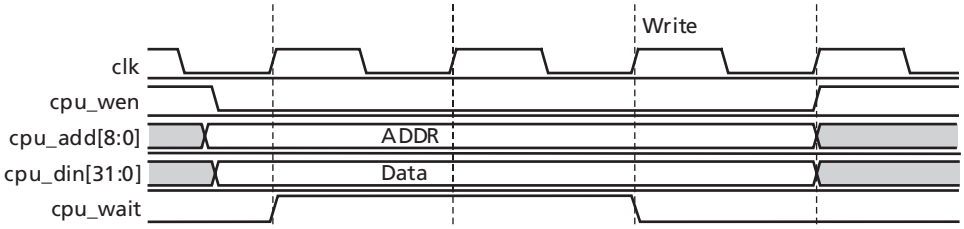

Figure 5-6. CPU Interface Label Memory Write Cycle

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.

6

Configuration Registers

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" on page 28).

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 6-1 shows the CPU address bit information.

Table 6-1. 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 6-2 through Table 6-5 on page 29).

Address map:

00 – Data Register

01 – Control Register

10 – Status Register

11 – Label Memory

Table 6-2. Rx Data Register

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

Table 6-3. 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 6-4. Rx Status Register

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

Table 6-5. 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 6-6. Tx Data Register

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

Table 6-7. 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 6-8. Tx Status Register

Bit	Function	Reset State	Type	Description
0	FIFO empty	0	R	0 = not empty; 1 = empty
1	FIFO half full or programmed level	0	R	0 = less than half full or programmed level; 1 = half full or programmed 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 6-1.

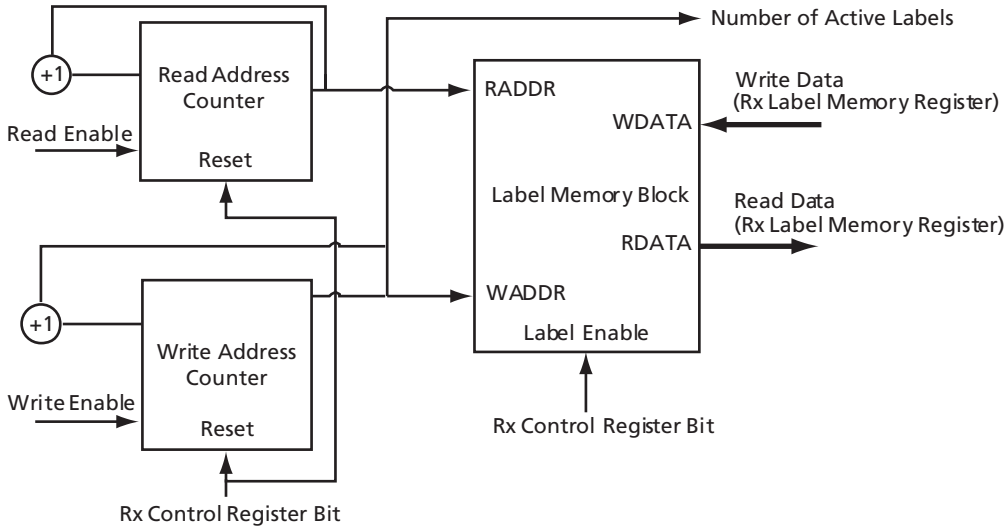

Figure 6-1. Label Memory Diagram

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

Legacy Operation

In this mode, there is a legacy interface block that communicates with the CPU interface. When legacy mode is enabled, the core supports two receive (Rx) channels and one transmit (Tx) channel only. Legacy mode is not configurable to support multiple transmit and receive channels. The purpose of the legacy mode interface is to replace existing standard products.

Control Register

Core429 contains a 16-bit control register, which is used to configure the Rx and Tx channels. Control register bits 0–15 are loaded from the data bus when CTRL_n is LOW. The control register contents are put out on the data bus when RSEL is HIGH and STR_n is LOW. Each bit of the control register is explained in Table 6-9.

Table 6-9. Legacy Control Register

Bit	Function	Reset State	Type	Description
0	Receiver 1 data rate	0	R/W	Data rate: 0 = 100 kbps; 1 = 12.5 kbps. Note: Does not support 50 kbps.
1	Label compare	0	R/W	0 = disable; 1 = enable Load 16 labels using pl1_n/pl2_n. Read 16 labels using en1_n/en2_n.
2	Enable label recognition (Receiver 1)	0	R/W	0: Disable label recognition 1: Enable label recognition
3	Enable label recognition (Receiver 2)	0	R/W	0: Disable label recognition 1: Enable label recognition
4	Enable bit 32 as parity	0	R/W	0 = bit 32 is data; 1 = bit 32 is parity
5	Self test	1	R/W	0: The transmitter's digital outputs are internally connected to the receiver's logic inputs. 1: Default operation
6	Receiver 1 decoder	0	R/W	0: Receiver 1 decoder disabled 1: ARINC bits 9 and 10 must match bits 7 and 8 of the control register.
7	Match ARINC bit 9 (Receiver 1)	0	R/W	If Receiver 1 decoder is enabled, ARINC bit 9 should match this bit.
8	Match ARINC bit 10 (Receiver 1)	0	R/W	If Receiver 1 decoder is enabled, ARINC bit 10 should match this bit.
9	Receiver 2 decoder	0	R/W	0: Receiver 2 decoder disabled 1: ARINC bits 9 and 10 must match bits 10 and 11 of the control register.
10	Match ARINC bit 9 (Receiver 2)	0	R/W	If Receiver 2 decoder is enabled, ARINC bit 9 should match this bit.
11	Match ARINC bit 10 (Receiver 2)	0	R/W	If Receiver 2 decoder is enabled, ARINC bit 10 should match this bit.
12	Transmitter parity	0	R/W	Parity: 0 = odd; 1 = even
13	Transmitter data rate	0	R/W	Data rate: 0 = 100 kbps; 1 = 12.5 kbps. Note: Does not support 50 kbps.
14	Receiver 2 data rate	0	R/W	Data rate: 0 = 100 kbps; 1 = 12.5 kbps. Note: Does not support 50 kbps.

Status Register

Core429 contains a 16-bit status register which can be read to determine the status of the ARINC receivers, data FIFOs, and transmitter. The contents of the status register are put out on the data bus when RSEL is LOW and STR is LOW. Each bit of the status register is explained in Table 6-10.

Table 6-10. Legacy Status Register

Bit	Function	Reset State	Type	Description
0	Receiver 1 FIFO empty	0	R	0 = Receiver 1 FIFO not empty 1 = Receiver 1 FIFO empty
1	Receiver 1 FIFO half full	0	R	0 = Receiver 1 FIFO not half full 1 = Receiver 1 FIFO half full
2	Receiver 1 FIFO full	0	R	0 = Receiver 1 FIFO not full 1 = Receiver 1 FIFO full
3	Receiver 2 FIFO empty	0	R	0 = Receiver 2 FIFO not empty 1 = Receiver 2 FIFO empty
4	Receiver 2 FIFO half full	0	R	0 = Receiver 2 FIFO not half full 1 = Receiver 2 FIFO half full
5	Receiver 2 FIFO full	0	R	0 = Receiver 2 FIFO not full 1 = Receiver 2 FIFO full
6	Transmitter FIFO empty	0	R	0 = Transmitter FIFO not empty 1 = Transmitter FIFO empty
7	Transmitter FIFO full	0	R	0 = Transmitter FIFO not full 1 = Transmitter FIFO full
8	Transmitter FIFO half full	0	R	0 = Transmitter FIFO not half full 1 = Transmitter FIFO half full

7 Testbench Operation

Three testbenches are provided with Core429:

- Verilog verification testbench: Complex testbench that verifies core operation. This testbench exercises all the features of the core. Actel recommends not modifying this testbench. This Verilog testbench may be used to simulate the VHDL version of the core if a mixed-mode simulator is available.
- 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.

Core429 Verification

The comprehensive verification simulation testbench (included with the Obfuscated and RTL versions of the core) verifies correct operation of the Core429 macro. The verification testbench applies several tests to the Core429 macro, including the following:

- Receive interface tests
- Transmit interface tests
- CPU interface tests
- Legacy interface tests
- Loopback tests

Using the supplied user testbench as a guide, the user can easily customize the verification of the core by adding or removing tests.

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 7-1). This consists of the core connections to a CPU model and loopback logic that connects the Tx output to the Rx input.

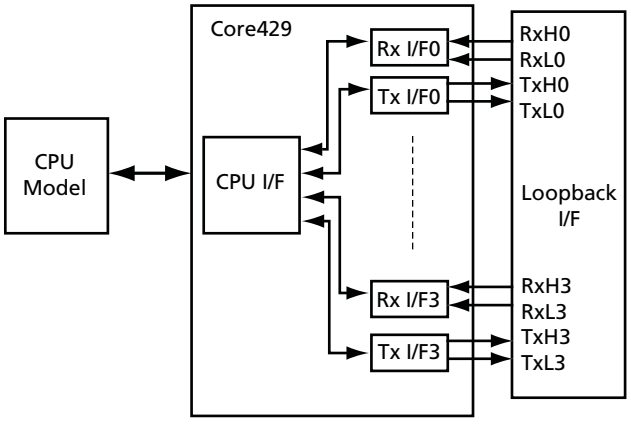

Figure 7-1. Testbench Diagram

Verification Testbench

Included with the Obfuscated and RTL releases of Core429 is a verification testbench that verifies operation of the Core429 macro. A simplified block diagram of the verification testbench is shown in Figure 7-2.

The verification testbench instantiates the Design Under Test (DUT), which is the Core429 macro, as well as the test vector modules that provide stimuli source for the DUT and perform comparisons for expected values throughout the simulation process. A procedural testbench controls each module and applies the sequential stimuli to the DUT.

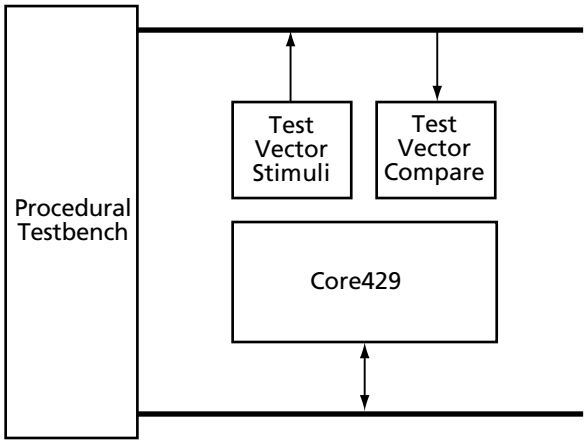

Figure 7-2. Core429 Verification Testbench

The source code for the verification testbench is only available with the Core429 RTL release. A compiled Model*Sim* simulation is available with the Obfuscated and RTL releases.

Verification Tests

Core429 is verified through various tests, listed below:

- Basic receive and transmit data words
- Parity errors
- Incorrectly formatted data words
- Label match
- FIFO flags, including FIFO level
- Control register settings
- Status register settings
- CPU interface data width tests
- Tx FIFO overflow and underrun
- Rx FIFO overflow and underrun
- Interrupts
- Loopback

User Testbench

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 (Evaluation, 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 7-3.

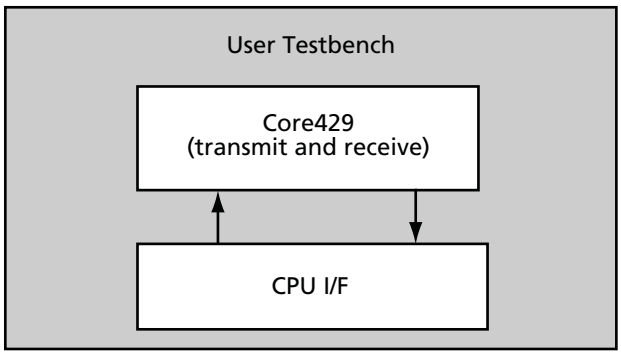

Figure 7-3. User Testbench

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 7-3 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 Registers" on page 27. As shown in Figure 7-3, 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.

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;
```

B

Product Support

Actel backs its products with various support services including Customer Service, a Customer Technical Support Center, a web site, an FTP site, electronic mail, and worldwide sales offices. This appendix contains information about contacting Actel 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 Northeast and North Central U.S.A., call **650.318.4480**
From Southeast and Southwest U.S.A., call **650. 318.4480**
From South Central U.S.A., call **650.318.4434**
From Northwest U.S.A., call **650.318.4434**
From Canada, call **650.318.4480**
From Europe, call **650.318.4252** or **+44 (0) 1276 401 500**
From Japan, call **650.318.4743**
From the rest of the world, call **650.318.4743**
Fax, from anywhere in the world **650.318.8044**

Actel Customer Technical Support Center

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

Actel Technical Support

Visit the Actel Customer Support website (www.actel.com/custsup/search.html) for more information and support. Many answers available on the searchable web resource include diagrams, illustrations, and links to other resources on the Actel web site.

Website

You can browse a variety of technical and non-technical information on Actel's home page, at www.actel.com.

Contacting the Customer Technical Support Center

Highly skilled engineers staff the Technical Support Center from 7:00 A.M. to 6:00 P.M., Pacific Time, Monday through Friday. Several ways of contacting the Center follow:

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 tech@actel.com.

Phone

Our Technical Support Center answers all calls. The center retrieves information, such as your name, company name, phone number and your question, and then issues a case number. The Center then forwards the information to a queue where the first available application engineer receives the data and returns your call. The phone hours are from 7:00 A.M. to 6:00 P.M., Pacific Time, Monday through Friday. The Technical Support numbers are:

650.318.4460
800.262.1060

Customers needing assistance outside the US time zones can either contact technical support via email (tech@actel.com) or contact a local sales office. Sales office listings can be found at www.actel.com/contact/offices/index.html.

Index

For more information about Actel's products, visit our website at http://www.actel.com

Actel Corporation *• 2061 Stierlin Court • Mountain View, CA 94043 USA*
Customer Service: 650.318.1010 • Customer Applications Center: 800.262.1060

Actel Europe Ltd*. • River Court, Meadows Business Park • Station Approach, Blackwater • Camberley Surrey GU17 9AB • United Kingdom*
Phone +44 (0) 1276 609 300 • Fax +44 (0) 1276 607 540

Actel Japan *• EXOS Ebisu Bldg. 4F • 1-24-14 Ebisu Shibuya-ku • Tokyo 150 • Japan*
Phone +81.03.3445.7671 • Fax +81.03.3445.7668 • www.jp.actel.com

Actel Hong Kong *• Suite 2114, Two Pacific Place • 88 Queensway, Admiralty Hong Kong*
Phone +852 2185 6460 • Fax +852 2185 6488 • www.actel.com.cn

50200088-0 /1.07