

---

# **Core429 v3.10**

## ***Handbook***



# Table of Contents

Introduction .....	5
General Description .....	5
ARINC 429 Overview .....	5
Core Versions .....	6
Core429 Device Requirements .....	7
Memory Requirements .....	8
<b>1 Functional Block Description .....</b>	<b>11</b>
Core429 Overview .....	11
Functional Description .....	11
Clock Requirements .....	12
Line Drivers .....	13
Line Receivers .....	13
Development System .....	14
<b>2 Tool Flows .....</b>	<b>15</b>
License Types .....	15
CoreConsole .....	15
Importing into Libero SoC .....	18
<b>3 Core Parameters .....</b>	<b>19</b>
<b>4 Core Interfaces .....</b>	<b>21</b>
I/O Signal Descriptions .....	21
<b>5 Timing Diagrams .....</b>	<b>23</b>
CPU Interface Timing for Default Mode .....	23
<b>6 Configuration Registers .....</b>	<b>25</b>
Default Mode Operation .....	25
Loopback Interface .....	28
<b>7 Testbench Operation .....</b>	<b>29</b>
Testbench .....	29
User Testbench .....	30
<b>8 List of Changes .....</b>	<b>31</b>
List of Changes .....	31

<b>A Testbench Support Routines</b>	<b>33</b>
VHDL Support	33
Verilog Support	33
<b>B Product Support</b>	<b>35</b>
Customer Service	35
Customer Technical Support Center	35
Technical Support	35
Website	35
Contacting the Customer Technical Support Center	35
ITAR Technical Support	36
<b>Index</b>	<b>37</b>

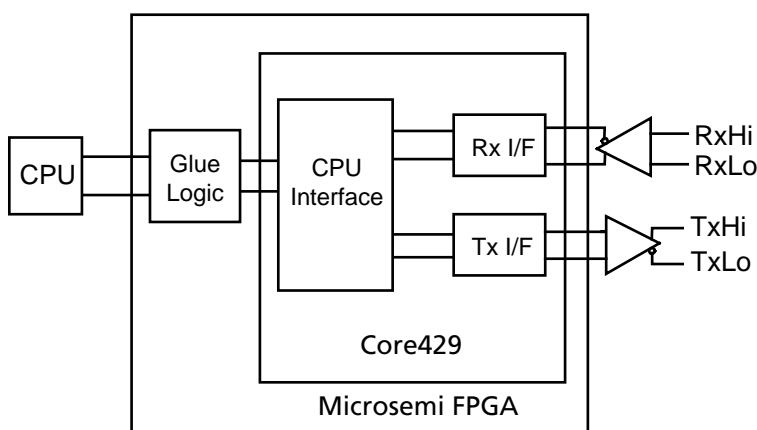


# Introduction

## 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:

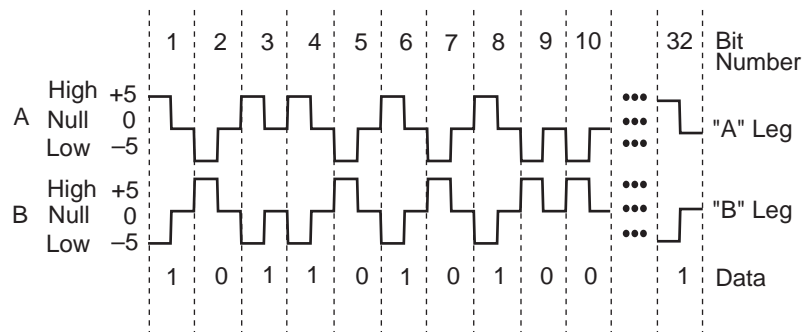
- Standard ARINC 429 line driver
- Standard ARINC 429 line receiver

## 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.

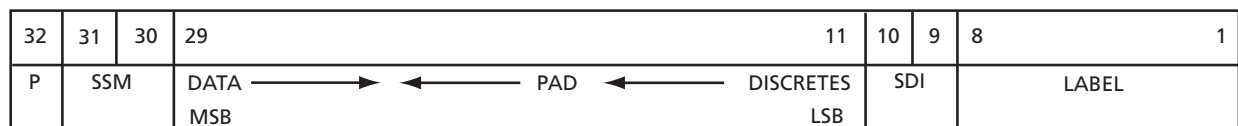
ARINC 429 uses a unidirectional data bus standard (Tx and Rx are on separate ports) known as the Mark 33 Digital Information Transfer System (DITS). Messages are transmitted at 12.5, 50 (optional), or 100 kbps to other system elements that are monitoring the bus messages. The transmitter is always transmitting 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 v3.10. 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 Microsemi 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			
SmartFusion®2	123	74	197	1	M2S050T	0.5%
SmartFusion	154	162	316	1	A2F500M3G	2%
Fusion	363	147	510	1	AFS600	4%
ProASIC®3/E	363	147	510	1	A3PE600	4%
ProASIC <sup>PLUS</sup> ®	441	146	587	1	APA075	19%
Axcelerator®	212	145	357	1	AX125	18%
RTAX™-S	258	171	429	1	RTAX250S	10%
IGLOO®2	115	77	192	1	M2GL150T	0.13%

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

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 <sup>PLUS</sup>	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%

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

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 <sup>PLUS</sup>	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%

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

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 <sup>PLUS</sup>	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%

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.

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

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.

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

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
ProASIC <sup>PLUS</sup>	256	64
Axcelerator/RTAX-S	512	128
IGLOO2	512	128
SmartFusion2	512	128
SmartFusion	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

$$\begin{aligned} \text{Number of memory blocks} &= 2 * (\text{INT}(64 / 512) + \text{INT}(32 / 128)) + 1 * \text{INT}(32 / 128) \\ &= 2 * (1 + 1) + 1 * (1) = 5. \end{aligned}$$

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

$$\begin{aligned} \text{Number of memory blocks} &= \text{INT}(64 / 128) + (\text{INT}(32 / 512) + \text{INT}(32 / 128)) + (\text{INT}(64 / 512) + \\ &\text{INT}(64 / 128)) = 1 + (1 + 1) + (1 + 1) = 5. \end{aligned}$$



# 1 – Functional Block Description

## Core429 Overview

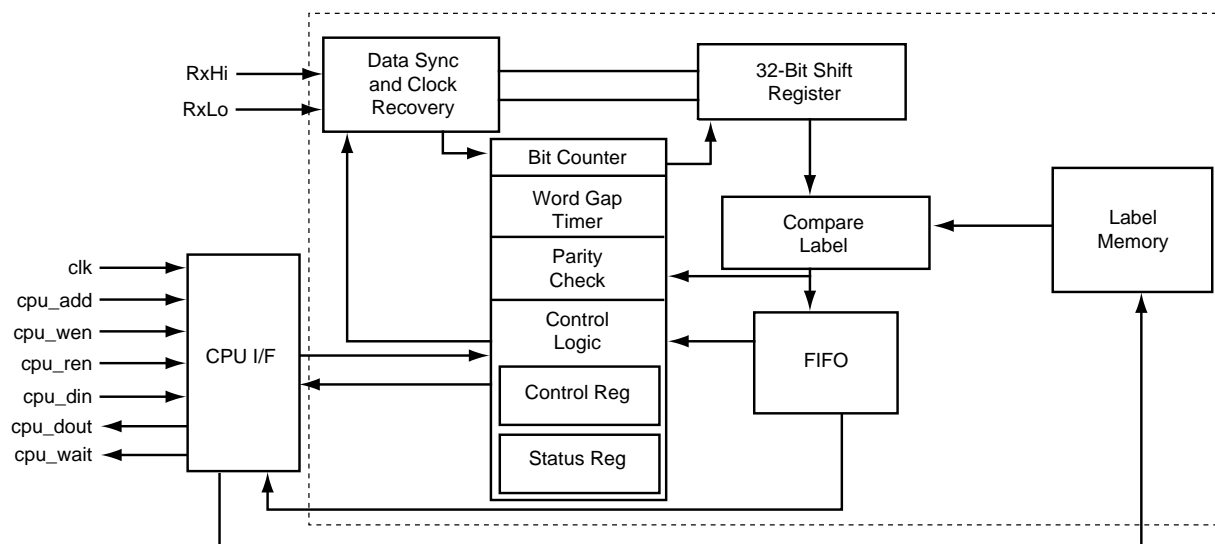
Core429 provides a complete and flexible interface to a microprocessor and an ARINC 429 data bus. Connection to an ARINC 429 data bus requires additional line drivers and line receivers.

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 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 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 26](#) and [Table 6-4 on page 26](#) 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.

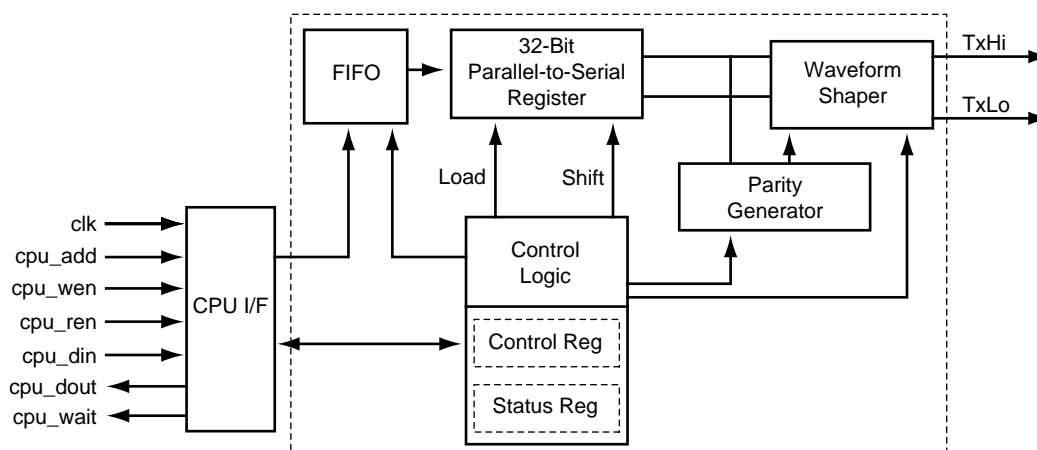
The core supports reloading label memory using bit 7 of the Rx control register. Note that when you set bit 7 to initialize the label memory, the old label content still exists, but the core keeps track only of the new label and does not use the old label during label compare.

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 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 27](#) and [Table 6-8 on page 27](#) 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.

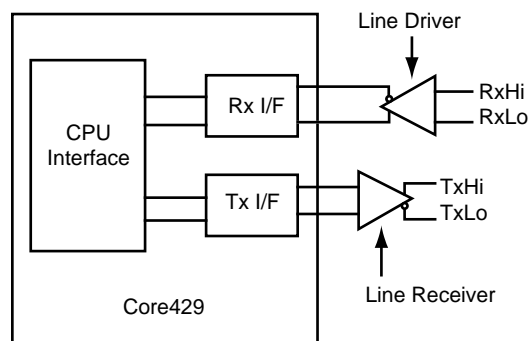
## Clock Requirements

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  $\pm 0.01\%$ .

## 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.



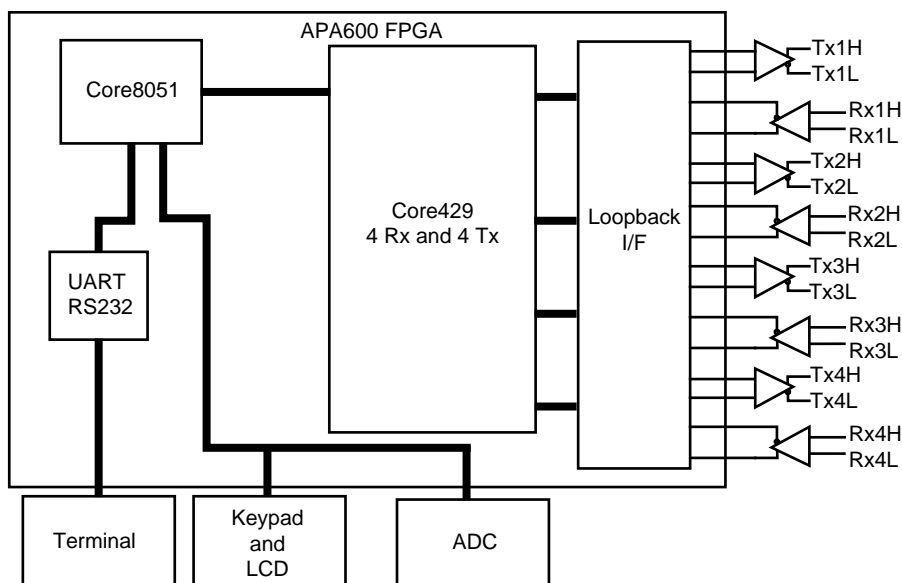
**Figure 1-3 • Core429 Line Driver and Line Receiver Interface**

## 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<sup>PLUS</sup>, 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<sup>PLUS</sup> 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 two ways. Tool flow functionality may be limited.

### License Types

#### 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<sup>®</sup> System-on-Chip (SoC) 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.

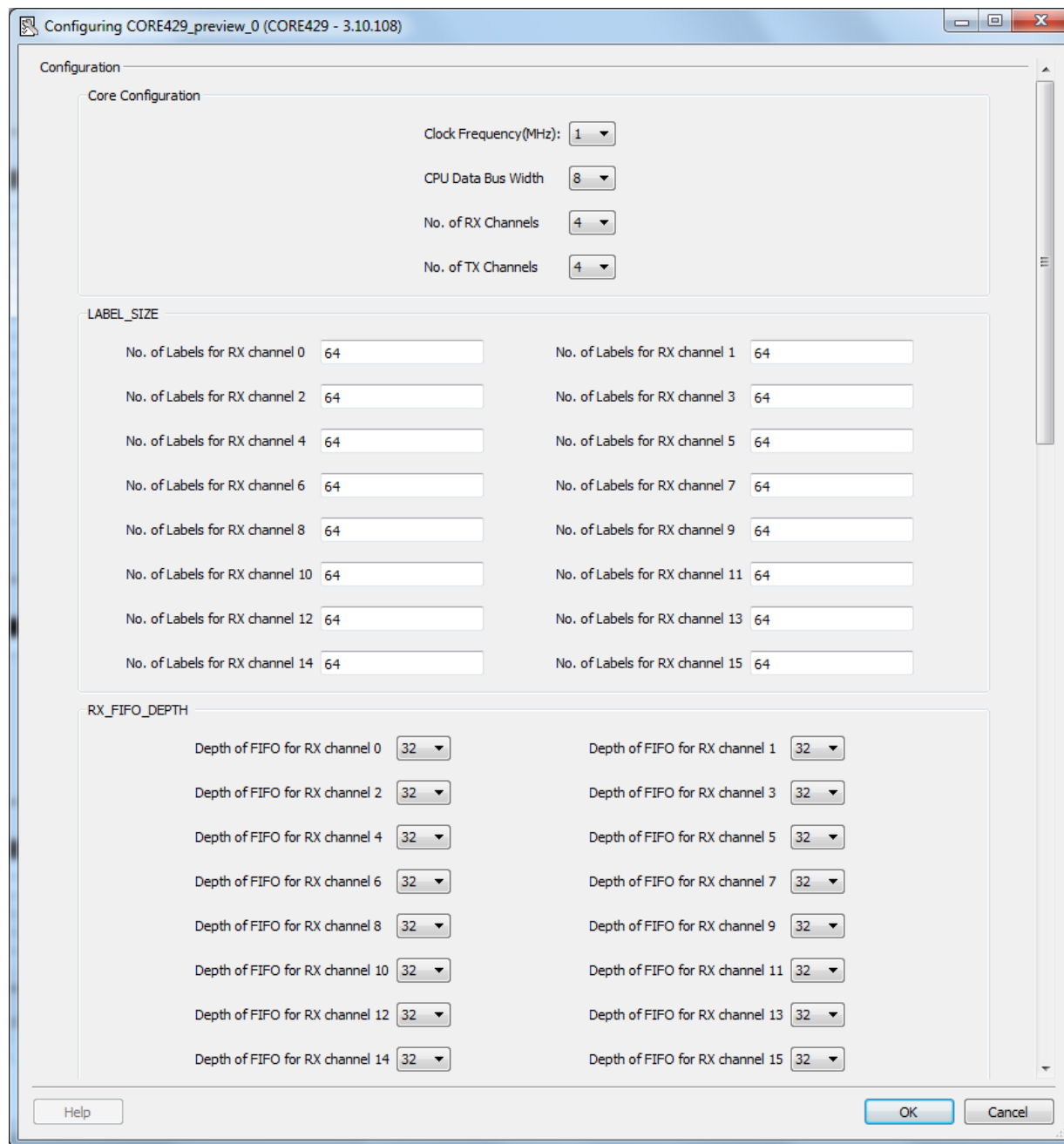
#### RTL

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

### 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 SoC 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 SoC.

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



Configuring CORE429\_preview\_0 (CORE429 - 3.10.108)

Configuration

Core Configuration

Clock Frequency(MHz): 1

CPU Data Bus Width 8

No. of RX Channels 4

No. of TX Channels 4

LABEL\_SIZE

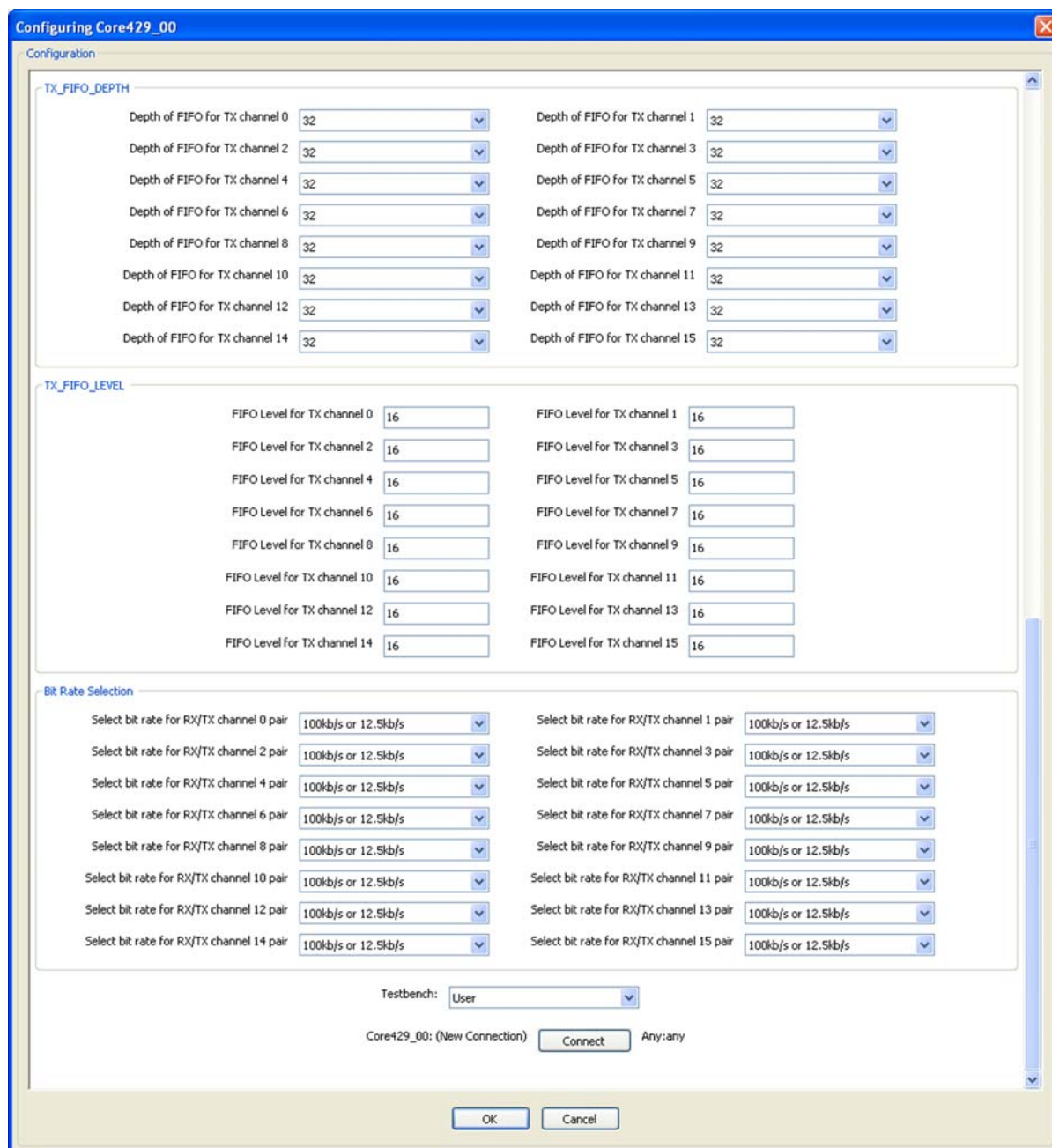
No. of Labels for RX channel 0	64	No. of Labels for RX channel 1	64
No. of Labels for RX channel 2	64	No. of Labels for RX channel 3	64
No. of Labels for RX channel 4	64	No. of Labels for RX channel 5	64
No. of Labels for RX channel 6	64	No. of Labels for RX channel 7	64
No. of Labels for RX channel 8	64	No. of Labels for RX channel 9	64
No. of Labels for RX channel 10	64	No. of Labels for RX channel 11	64
No. of Labels for RX channel 12	64	No. of Labels for RX channel 13	64
No. of Labels for RX channel 14	64	No. of Labels for RX channel 15	64

RX\_FIFO\_DEPTH

Depth of FIFO for RX channel 0	32	Depth of FIFO for RX channel 1	32
Depth of FIFO for RX channel 2	32	Depth of FIFO for RX channel 3	32
Depth of FIFO for RX channel 4	32	Depth of FIFO for RX channel 5	32
Depth of FIFO for RX channel 6	32	Depth of FIFO for RX channel 7	32
Depth of FIFO for RX channel 8	32	Depth of FIFO for RX channel 9	32
Depth of FIFO for RX channel 10	32	Depth of FIFO for RX channel 11	32
Depth of FIFO for RX channel 12	32	Depth of FIFO for RX channel 13	32
Depth of FIFO for RX channel 14	32	Depth of FIFO for RX channel 15	32

Help OK Cancel

**Figure 2-1 • Core429 Configuration within CoreConsole**



**Configuring Core429\_00**

Configuration

**TX\_FIFO\_DEPTH**

Depth of FIFO for TX channel 0	32
Depth of FIFO for TX channel 1	32
Depth of FIFO for TX channel 2	32
Depth of FIFO for TX channel 3	32
Depth of FIFO for TX channel 4	32
Depth of FIFO for TX channel 5	32
Depth of FIFO for TX channel 6	32
Depth of FIFO for TX channel 7	32
Depth of FIFO for TX channel 8	32
Depth of FIFO for TX channel 9	32
Depth of FIFO for TX channel 10	32
Depth of FIFO for TX channel 11	32
Depth of FIFO for TX channel 12	32
Depth of FIFO for TX channel 13	32
Depth of FIFO for TX channel 14	32
Depth of FIFO for TX channel 15	32

**TX\_FIFO\_LEVEL**

FIFO Level for TX channel 0	16
FIFO Level for TX channel 1	16
FIFO Level for TX channel 2	16
FIFO Level for TX channel 3	16
FIFO Level for TX channel 4	16
FIFO Level for TX channel 5	16
FIFO Level for TX channel 6	16
FIFO Level for TX channel 7	16
FIFO Level for TX channel 8	16
FIFO Level for TX channel 9	16
FIFO Level for TX channel 10	16
FIFO Level for TX channel 11	16
FIFO Level for TX channel 12	16
FIFO Level for TX channel 13	16
FIFO Level for TX channel 14	16
FIFO Level for TX channel 15	16

**Bit Rate Selection**

Select bit rate for RX/TX channel 0 pair	100kb/s or 12.5kb/s
Select bit rate for RX/TX channel 1 pair	100kb/s or 12.5kb/s
Select bit rate for RX/TX channel 2 pair	100kb/s or 12.5kb/s
Select bit rate for RX/TX channel 3 pair	100kb/s or 12.5kb/s
Select bit rate for RX/TX channel 4 pair	100kb/s or 12.5kb/s
Select bit rate for RX/TX channel 5 pair	100kb/s or 12.5kb/s
Select bit rate for RX/TX channel 6 pair	100kb/s or 12.5kb/s
Select bit rate for RX/TX channel 7 pair	100kb/s or 12.5kb/s
Select bit rate for RX/TX channel 8 pair	100kb/s or 12.5kb/s
Select bit rate for RX/TX channel 9 pair	100kb/s or 12.5kb/s
Select bit rate for RX/TX channel 10 pair	100kb/s or 12.5kb/s
Select bit rate for RX/TX channel 11 pair	100kb/s or 12.5kb/s
Select bit rate for RX/TX channel 12 pair	100kb/s or 12.5kb/s
Select bit rate for RX/TX channel 13 pair	100kb/s or 12.5kb/s
Select bit rate for RX/TX channel 14 pair	100kb/s or 12.5kb/s
Select bit rate for RX/TX channel 15 pair	100kb/s or 12.5kb/s

Testbench: User

Core429\_00: (New Connection)  Any:any

**Figure 2-2 • Core429 Configuration within CoreConsole (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 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 SoC

After generating and exporting the core from CoreConsole, the core can be imported into Libero SoC. Create a new project in Libero SoC and import the CoreConsole project from the *LiberoExport* directory. Libero SoC 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 SoC 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 SoC project, it will install the appropriate testbench files. To run the testbenches, simply set the design root to *Core429 instantiation* in the Libero SoC file manager and click the **Simulation** icon in Libero SoC. This will invoke ModelSim and automatically run the simulation.

## Synthesis in Libero SoC

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 SoC. The synthesis window appears, displaying the Synplicity® project. To run Synthesis, click the **Run** icon.

## Place-and-Route in Libero SoC

Having set the design route appropriately and run Synthesis, click the **Layout** icon in Libero SoC 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. Microsemi will supply netlists with alternative parameter settings on request.

**Table 3-1 • FIFO and Label Parameters**

Parameter Name	Description	Allowed Values	Default	Note
FAMILY	Must be set to the required FPGA family.	11 to 24	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 <a href="#">Table 6-3 on page 26</a> and <a href="#">Table 6-7 on page 27</a> .	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.



## 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
rx_a[RXN–1:0]	In	ARINC receiver input A
rx_b[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: <ul style="list-style-type: none"> <li>FIFO empty</li> <li>FIFO full</li> <li>FIFO is full up to the configured RX_FIFO_LEVEL</li> </ul> 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: <ul style="list-style-type: none"> <li>FIFO empty</li> <li>FIFO full</li> <li>FIFO is full up to the configured TX_FIFO_LEVEL</li> </ul> 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.

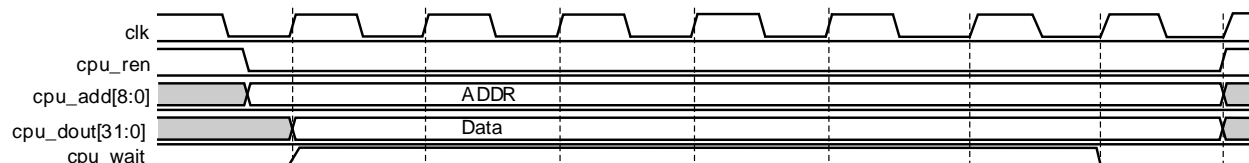
**Table 4-3 • CPU Interface Signals**

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.

## 5 – Timing Diagrams

### CPU Interface Timing for Default Mode

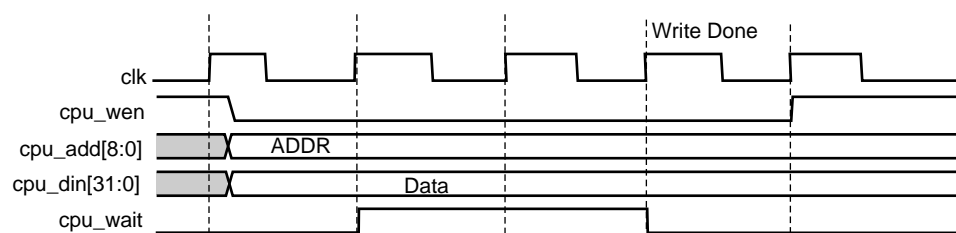
The CPU interface signals are synchronized to the Core429 master clock. [Figure 5-1](#) through [Figure 5-6](#) on [page 24](#) show the waveforms for the CPU interface.



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

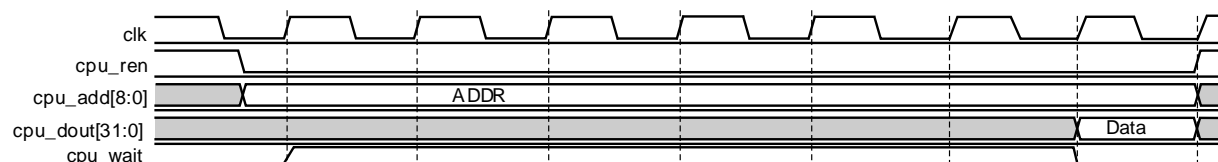
**Note:**

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.



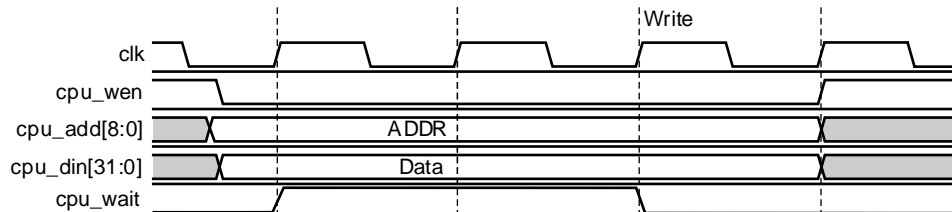
**Figure 5-2 • CPU Interface Control Register Write Cycle**

**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.



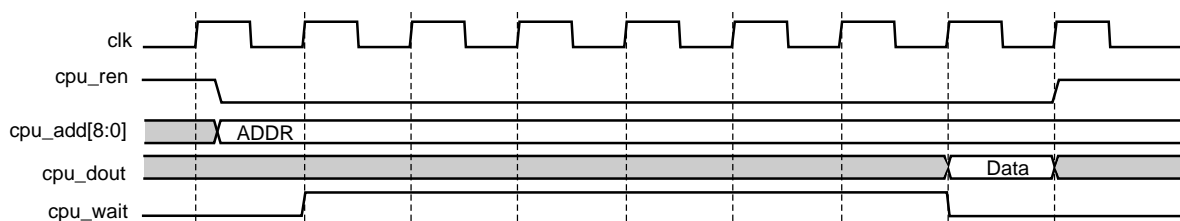
**Figure 5-3 • CPU Interface Data Register Read Cycle**

**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.



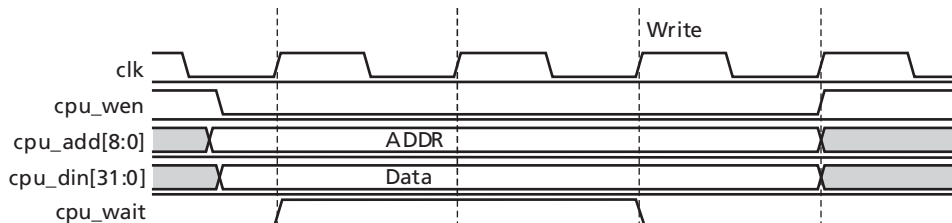
**Figure 5-4 • CPU Interface Data Register Write Cycle**

**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 26).

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 26).

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 32 <sup>nd</sup> bit as parity	0	R/W	0 = 32 <sup>nd</sup> bit is data; 1 = 32 <sup>nd</sup> 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	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 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 32 <sup>nd</sup> bit as parity	0	R/W	0 = 32 <sup>nd</sup> bit is data; 1 = 32 <sup>nd</sup> bit is parity
3	Parity	0	R/W	Parity: 0 = odd; 1 = even

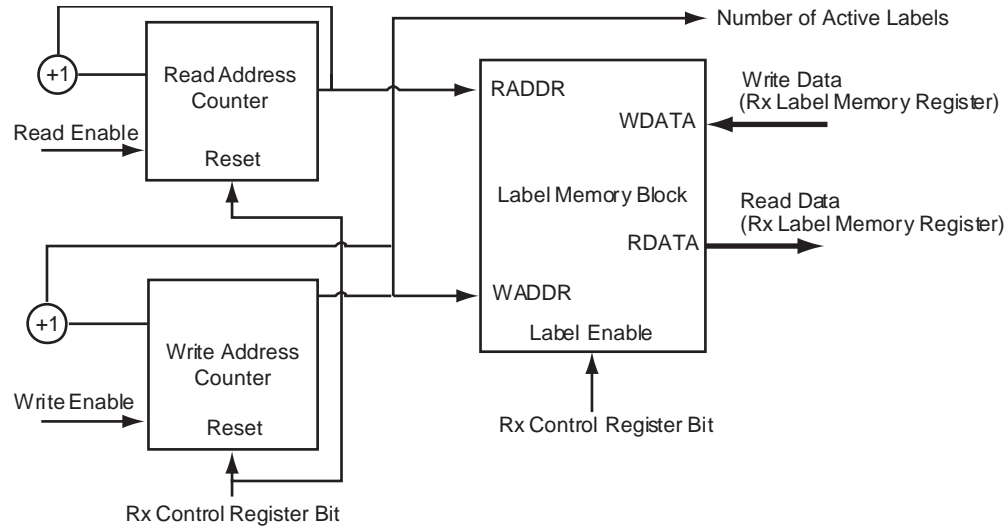
**Table 6-8 • Tx Status Register**

Bit	Function	Reset State	Type	Description
0	FIFO empty	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 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

## 7 – Testbench Operation

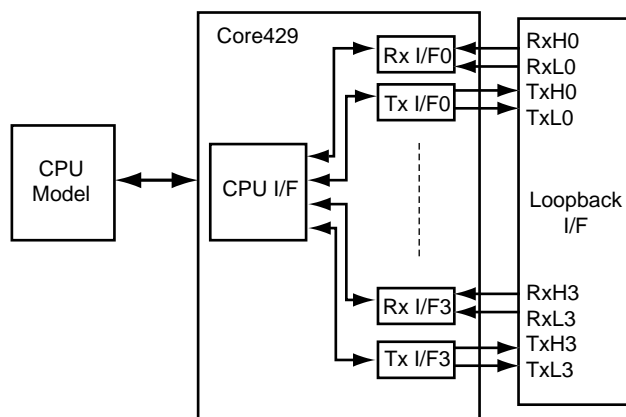
Two testbenches are provided with Core429:

- **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.

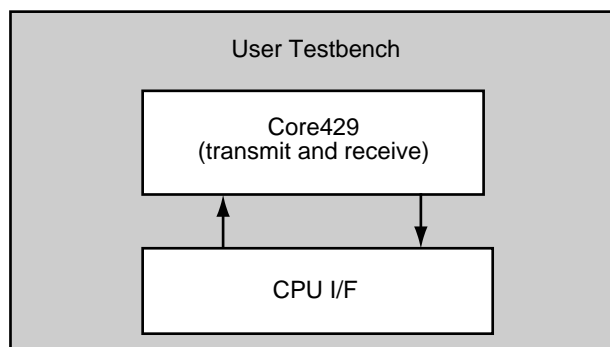
The user testbenches are intended to simplify core integration into the target system ([Figure 7-1](#)). This consists of the core connections to a CPU model and loopback logic that connects the Tx output to the Rx input.



**Figure 7-1 • Testbench Diagram**

## User Testbench

An example user testbench is included with the Evaluation, Obfuscated, and RTL releases of Core429. The user testbench is provided in precompiled ModelSim 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 7-2.



**Figure 7-2 • 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 7-2 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 section on page 25](#). As shown in Figure 7-2, 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.

## 8 – List of Changes

### List of Changes

The following table lists critical changes that were made in each revision of the document.

Revision	Changes	Page
Revision 4 (April 2014)	Updated <a href="#">Table 3-1</a> .	19
	Updated <a href="#">Table 6-3</a> .	26
	Updated "Label Memory Operation" section.	28
Revision 3 (February 2014)	Added IGLOO2 to <a href="#">Table 1</a> , <a href="#">Table 2</a> , <a href="#">Table 3</a> , <a href="#">Table 4</a> , and <a href="#">Table 5</a> .	7, 8, 9
	Added IGLOO2, SmartFusion and SmartFusion2 to <a href="#">Table 6</a> .	9
	Updated <a href="#">Table 3.1</a> - FIFO and Label Parameters	19
	Updated <a href="#">Table 6.4</a> - Rx Status Register	26
	Updated <a href="#">Table 6.8</a> - 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)	<a href="#">Table 1 • Device Utilization for One Tx Module (default mode)</a> through <a href="#">Table 5 • Device Utilization for Legacy Mode (two Rx and one Tx)</a> 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, Tx FIFO level flag, and FIFO half full.	Multi
	RxLo and RxHi were corrected to TxLo and TxHi in <a href="#">Figure 1-2 • Core429 Tx Block Diagram</a> .	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 <a href="#">Figure 1-3 • Core429 Line Driver and Line Receiver Interface</a> . This was corrected.	13
	The Evaluation license option was removed from the "Tool Flows" section.	15
	The TXRXRESET parameter was added to <a href="#">Table 3-1 • FIFO and Label Parameters</a> . Notes were added to the table regarding the limit of the label recognition feature and setting the TXRXRESET parameter.	19
	<a href="#">Figure 5-1 • CPU Interface Control/Status Register Read Cycle</a> was replaced.	23
Revision 1 (April 2007)	Revised the allowed values and default columns for CLK_FREQ in <a href="#">Table 3-1 • FIFO and Label Parameters</a> .	19



## 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

---

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, 408.643.6913

### 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

Visit the Customer Support website ([www.microsemi.com/soc/support/search/default.aspx](http://www.microsemi.com/soc/support/search/default.aspx)) for more information and support. Many answers available on the searchable web resource include diagrams, illustrations, and links to other resources on the website.

### Website

You can browse a variety of technical and non-technical information on the SoC home page, at [www.microsemi.com/soc](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](mailto: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](mailto: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](http://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](mailto: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.

---

# Index

---

## A

ARINC 5

## C

clock requirements 13

components

external 5

contacting Microsemi SoC Products Group

customer service 35

email 35

web-based technical support 35

core parameters 19

core version 6

CoreConsole 15 CPU

address map 25

customer service 35

## D

development system 14

device requirements 7

## E

external components 5

## F

functional description 11

## I

I/O signals 21

## L

label memory 27

Libero SoC 18

licenses

Obfuscated 15

RTL 15

line drivers 13

line receivers 13

loopback interface 28

## M

memory requirements 8

Microsemi SoC Products Group

email 35

web-based technical support 35

website 35

## P

parameters

core 19

product support

customer service 35

email 35

My Cases 36

outside the U.S. 36

technical support 35

website 35

## R

register definitions 26

requirements

clock 13

device 7

memory 8

## S

signals 21

## T

tech support

ITAR 36

My Cases 36

outside the U.S. 36

technical support 35

testbench support routines 35

testbenches 31

timing diagrams 23

## W

web-based technical support 35



**Microsemi**

**Microsemi Corporate Headquarters** One  
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  
Email: [sales.support@microsemi.com](mailto: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, setting the world's standard for time; voice processing devices; RF solutions; discrete components; security technologies and scalable anti-tamper products; Power-over-Ethernet ICs and midspans; as well as custom design capabilities and services. Microsemi is headquartered in Aliso Viejo, Calif. and has approximately 3,400 employees globally. Learn more at [www.microsemi.com](http://www.microsemi.com).

© 2014 Microsemi Corporation. All rights reserved. Microsemi and the Microsemi logo are trademarks of Microsemi Corporation. All other trademarks and service marks are the property of their respective owners.