CARDBUS INTERFACE USER MANUAL

1 Scope

The COM-13xx ComBlock modules are PC cards which support communication with a host computer through a standard CardBus interface. These ComBlock modules can be used as

(a) ready-to-use application-specific ComBlocks, or

(b) development platforms with user-developed code.

This manual addresses both use cases. Its scope is limited to the 32-bit CardBus interface[1].

Users of ready-to-use application-specific ComBlocks should read the following sections: "Architecture", "Windows Device Driver Installation" and on "Applications".

Developers should also read the sections on CardBus component which implements the CardBus interface within the FPGA

The current implementation is subject to the following limitations:

- CardBus interface with 32-bit memory mapped and 8-bit I/O mapped data transfers between ComBlock and host PC
- Windows XP/2000 device driver

.

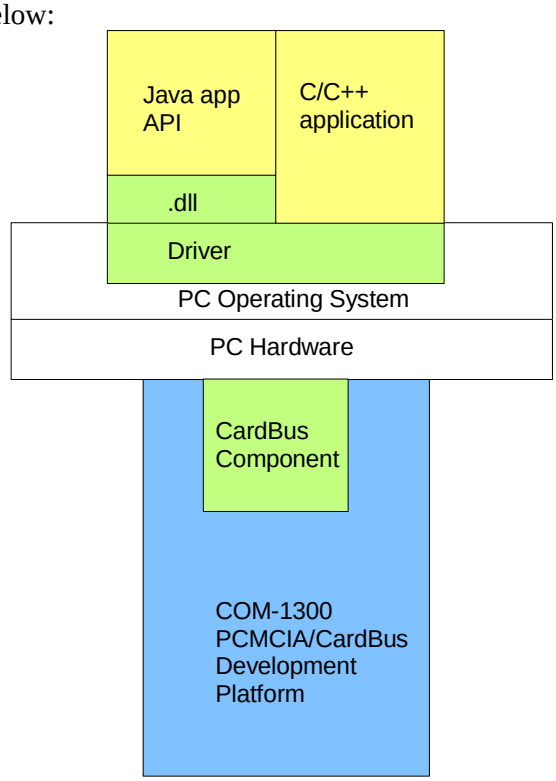
Throughput:

The CardBus interface sustained (average) throughput was measured using one-way data transfer benchmarks as shown below:

Throughput test conditions	***Throughput***
Memory-mapped data transfers:	*40 Mbps* (Read) *102 Mbps* (Write)
I/O-mapped data transfers:	*10 Mbps* (either direction)
Host computer: AMD processor 1.2 GHz. C runtime application, no hard disk data transfers. No other application running.	

2 Architecture

The end-to-end communication architecture between a host computer and the ComBlock module as a CardBus peripheral is illustrated below:



Software Development environment
Blue: supplied hardware
Green: supplied ready-to-use software
Yellow: application-level code examples.

[1] COM-13xx PC Cards also embody the 16-bit PCMCIA interface, as addressed in a separate user manual.

2.1 Host side (PC):

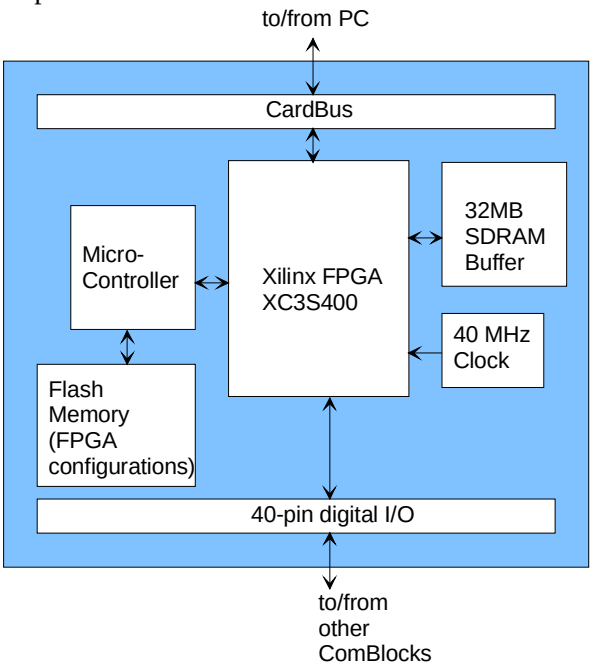
In order for a user to setup a CardBus connection between the host computer and the ComBlock, the user must first create a Java or C/C++ application.

The Java application calls simple methods described in the Java Application Programming Interface (API) described further in this document.

C/C++ applications can call drivers functions directly as described in the C/C++ Applications described further in this document.

2.2 Peripheral side (ComBlock):

On the peripheral side, the CardBus connection is implemented within the FPGA as illustrated below:

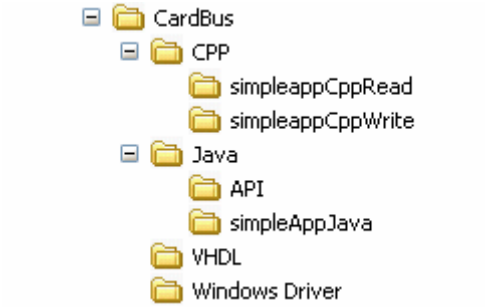


Development Card Hardware Block Diagram

2.3 Supplied Components:

The **CardBus software package** provides software to help users and developers create CardBus communication between the COM-13xx ComBlock module and a host PC. The software components include the following:

- Windows device driver for XP/2000 (.sys and .inf files)
- Java API
- Java simple application code examples
- C/C++ simple application code examples
- config_c, iorw_c, memory_c NGC components for integration within the VHDL code



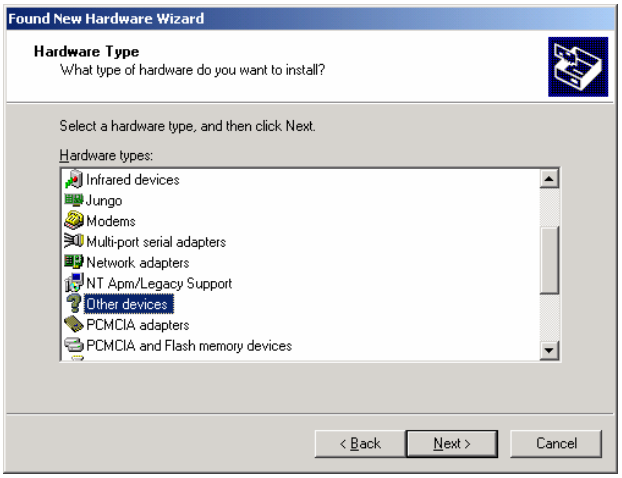
The **CardBus software package** is available in the ComBlock CD and can also be downloaded from www.comblock.com/download/CardBus.zip

3 Windows Device Driver Installation

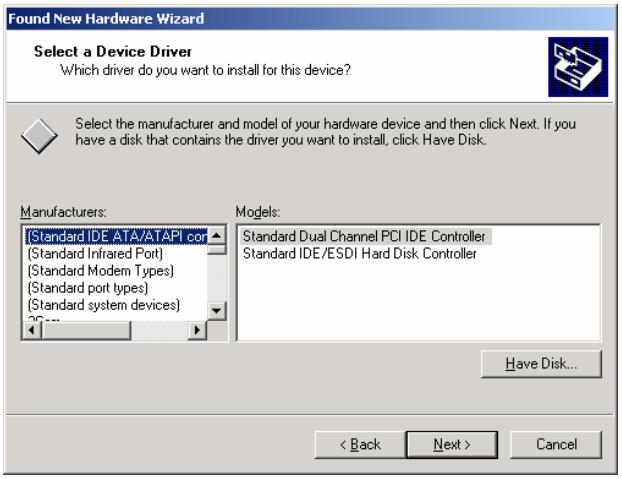
When connecting the COM-1300 CardBus interface for the first time, the user is prompted for new hardware installation. Follow the step by step instructions shown below each screen shot.

Click on Next.

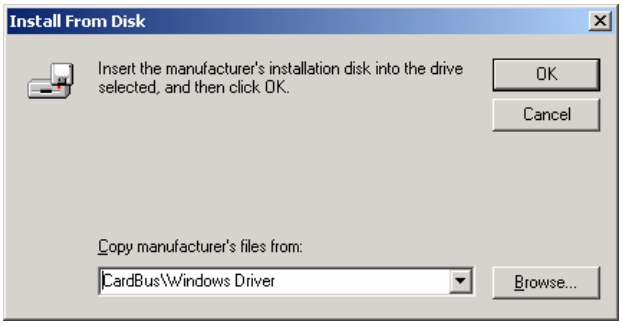
Check “Display a list of known drivers for the device so that I can choose a specific driver”. Click “Next”.



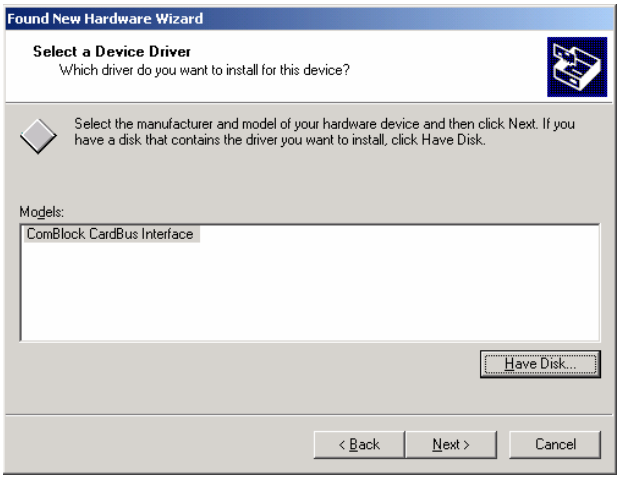
Select “Other Devices”. Go to the Next screen.



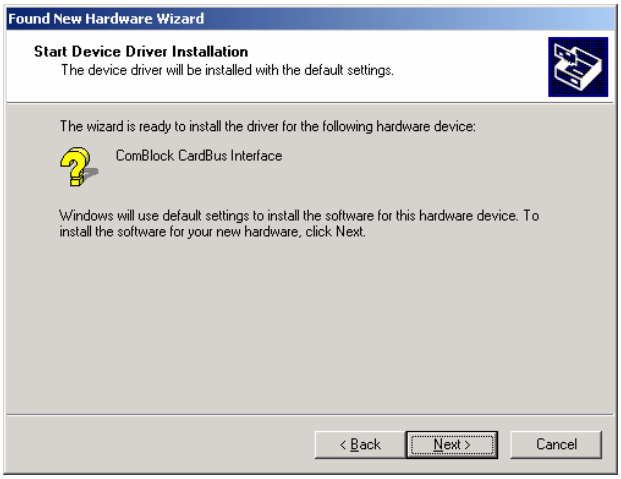
Click on “Have Disk”.



Point to the location where the driver files are and click OK.



Go to the next screen.



Click on Next.

The last window for the New Hardware Wizard should appear, as shown below, for a successfully installed device.

Click on Finish.

At this point, the driver for COM-1300 with CardBus interface has been successfully installed and next time the device is plugged in, the system automatically finds appropriate driver. With the driver installed, the user can talk to the device, using the applications.

4 Applications

4.1 Java API

The Java API is documented in …\Java\API\cardbus.html and can be found in the CardBus software package.

The applications call simple methods to get a handle, dispose a handle, read and write.

The DLL, which links the Java application to the drivers, is provided in the CardBus software package.

The Java application can transfer data buffers in the range of 0 to 2,048 Bytes to/from the CardBus target. When addressing a memory-mapped stream, all transfers are multiple of 4-byte words.

Flow control is implemented by checking the number of bytes actually transmitted at the end of the transaction.

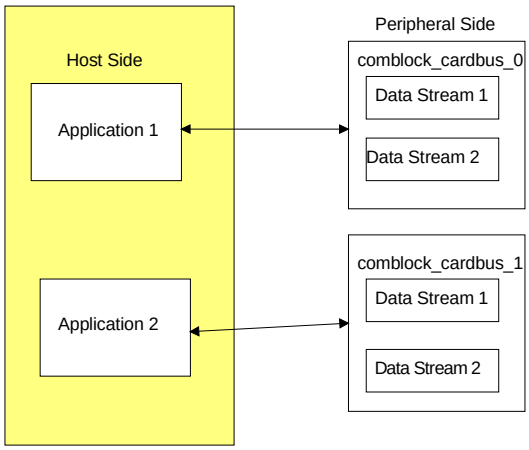
4.2 C/C++ Applications

The applications in C++ contain functions to open the device handle, send/receive data and close the handle.

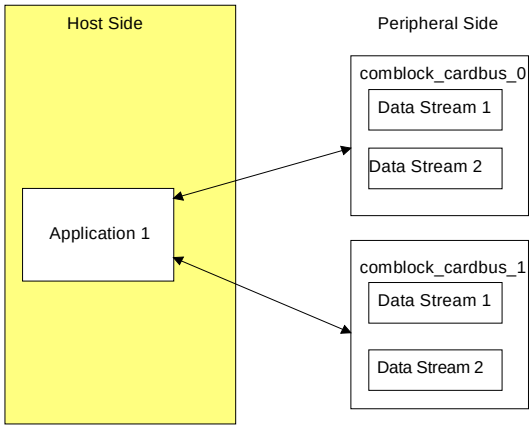
Application examples can be found in the CardBus software package.

4.3 Addressing Multiple ComBlocks

Multiple ComBlocks can be attached to a Host PC. Each ComBlock can be identified by a unique device name assigned when it is attached. The device name would be “comblock_cardbus_X” where X is a number starting with 0 and it depends on the order in which the ComBlock has been attached. The user applications can communicate with any of the ComBlocks exclusively by addressing them with the device name.



Sample communication model 1: Two user applications communicating with two ComBlocks.



Sample communication model 2: One user application communicating with two.

5 FPGA/VHDL/DRIVER Development

This section describes how to create a custom application that makes use of the CardBus medium on the ComBlock COM-1300 FPGA-based

development platforms. This section can be skipped by users of ready-to-use application-specific ComBlock modules.

This section focuses primarily on the peripheral side of the CardBus connection.

5.1 Host <-> Target Communication Methods

In its basic form the CardBus component supports two methods of bi-directional data exchange between host and target:

- One virtual bi-directional data stream is I/O mapped and exchanges 8-bit wide data.
- The other virtual bi-directional data stream is memory-mapped, exchanges 32-bit words and is optimized for maximum throughput.

The intent is to use the I/O mapped data stream to communicate with ComBlock itself, for monitoring and control purposes. The Memory-mapped data stream's intended use is for transferring payload data.

The addressing scheme is the same for both I/O and Memory-mapped data streams:
- Base address is 0.
- Address range used: 0 - 16

Beyond the basic software, developers can create multiple I/O and Memory-mapped data streams by instantiating multiple VHDL components and specifying non-overlapping address ranges.

5.2 Driver Installation

Upon insertion of a COM-1300 (CardBus interface), the bus driver will read the configuration header, in particular, the vendor ID and product ID, from the FPGA. Then it searches through the system registries to find matches with the vendor ID and product ID.

For the first-time installation of COM-1300, the operating system will discover that the vendor ID and product ID are new to the system registries. The user will be directed through a new hardware installation.

The New Hardware Wizard will check the INF file in the specified directory to see if it matches the vendor ID and product ID the host read from the hardware. If matched, the host will find the required drivers (.sys) defined in the .inf file, and copies the drivers to a location described in the .inf file (C:\WINDOWS\system32\drivers by default).

At this point, the .inf and .sys files will be copied by the operating system, and the system registry will be updated to include this device entry. Next time the device is plugged in, the system automatically finds it in the registry and links the device to the appropriate driver.

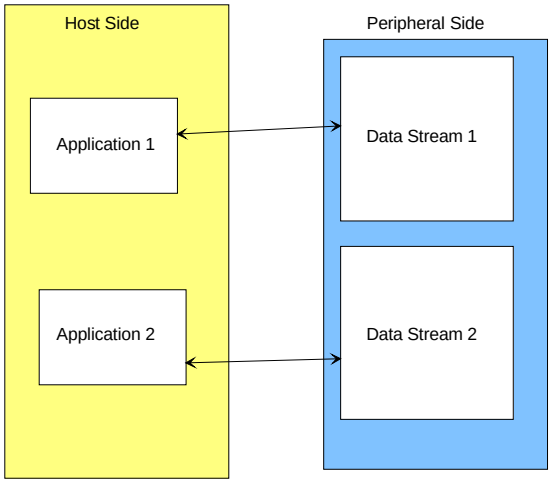
5.3 CardBus Component

The CardBus implementation on the target side is encapsulated within three NCG components, namely, config_c.ngc, iorw_c.ngc and memory_c.ngc. This implementation supports two bi-directional data streams:

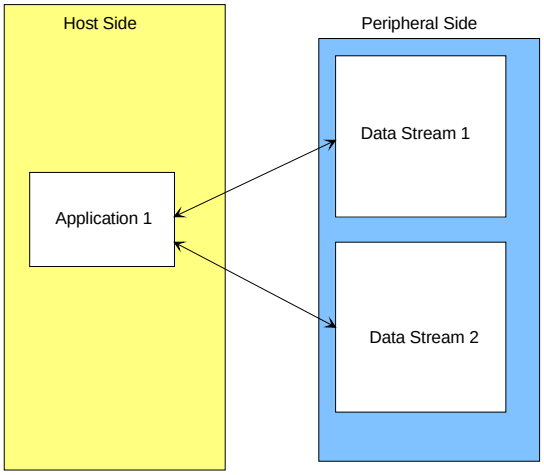
- One virtual data stream that is I/O mapped and exchanges 8-bit wide data.
- The other virtual data stream that is memory-mapped, exchanges 32-bit words

Data is exchanged with the CardBus component through a 16Kbit dual-port (elastic) buffer in each direction.

The data streams are to be used in conjunction with Java or C/C++ applications. The user applications can communicate with either of the two data streams or both.



Sample communication model 3: Two user applications communicating with two independent data streams on a single ComBlock.



Sample communication model 4: One user application communicating with two independent data streams on a single ComBlock.

5.3.1 Interface

The component is described primarily by its interface definition:

```
entity CARDBUS is
   port (

--// Clocks, reset
CLK_P: in std_logic;
  -- Main processing or I/O clock used outside of this component.
  -- All VHDL user application interface signals are synchronous with
CLK_P
  -- Key assumptions about speed: CLK_P > 8 MHz
ASYNC_RESET: in std_logic;
  -- asynchronous reset at power up. MANDATORY.

--// Host bus adapter interface:
CB_CARD_ADDR_DATA: inout std_logic_vector(31 downto 0);
  -- Address and Data

--// Command
CB_CARD_CC_BE_N: in std_logic_vector(3 downto 0);
  -- CardBus command (defines transaction type) or Byte Enable,

--// System
CB_CARD_CCLK: in std_logic;
  -- CardBus clock signal, 0 to 33 MHz. Global clock.
CB_CARD_CCLKRUN_N: inout std_logic;
  -- Asserted if clock runs normally
  -- deasserted before clock stops or slow down (I/O)
CB_CARD_CRST_N: in std_logic;
  -- Reset signal
  -- Forces CardBus configuration registers to an initialized state

--// Interface Control
CB_CARD_CPAR: inout std_logic;
  -- Parity signal (I/O)
CB_CARD_CFRAME_N: in std_logic;
  -- Data Frame indicator
CB_CARD_CTRDY_N: out std_logic;
  -- Target ready
CB_CARD_CIRDY_N: in std_logic;
  -- Initiator ready
CB_CARD_CSTOP_N: out std_logic;
  -- Target wants to stop the transaction
CB_CARD_CDEVSEL_N: out std_logic;
  -- Device select
  -- Asserted by target upon successful decoding of the address and
command
CB_CARD_CBLOCK_N: in std_logic;
  -- Lock the currently addressed memory target

--// Miscellaneous Signals
CB_CARD_CAUDIO_N: out std_logic;
  -- Card audio output. Not used: FPGA to pull high.
CB_CARD_CSTSCHG_N: out std_logic;
  -- STSCHG#     During memory or I/O interface

--// Error Reporting
CB_CARD_CPERR_N: out std_logic;
  -- Data parity error
CB_CARD_CSERR_N: out std_logic;
  -- System error
CB_CARD_CINT_N: out std_logic;
  -- Interrupt request

--// Arbitration (Bus Master Only)
CB_CARD_CGNT_N: in std_logic;
  -- Bus arbitration grant. Not used BUT must be
  -- pulled high according to specs.
  -- USE as input and PULL-UP
CB_CARD_CREQ_N: in std_logic;
  -- Arbitration request. MUST BE DECLARED AND USED AS
INPUT.
  -- (otherwise alternative assignement for pin N1 conflicts
  -- with proper operation) !!!!!

--// user interfaces
--// Stream1. 32-bit Memory read/write transactions
-- Synchronous with CLK_P clock
DATA1_OUT: out std_logic_vector(7 downto 0);
DATA1_OUT_SAMPLE_CLK: out std_logic;
  -- read DATA1_OUT at rising edge of CLK_P when
  -- DATA1_OUT_SAMPLE_CLK = '1'
  -- Note1: the user is responsible for checking
  -- DATA1_OUT_BUFFER_EMPTY before
  -- reading.
  -- Note 2: When the elastic buffer is not empty, DATA1_OUT is
  -- present at this interface even before requesting it. The request
  -- DATA1_OUT_SAMPLE_CLK_REQ   only moves the read pointer
  -- to the next read location.
DATA1_OUT_BUFFER_EMPTY: out std_logic;
DATA1_OUT_SAMPLE_CLK_REQ: in std_logic;
  -- requests data. If no data is available in the buffer, the
  -- DATA1_OUT_SAMPLE_CLK will stay low.
  -- (flow control)

DATA1_IN: in std_logic_vector(7 downto 0);
DATA1_IN_SAMPLE_CLK: in std_logic;
  -- read DATA1_IN at rising edge of CLK_P when
  -- DATA1_IN_SAMPLE_CLK = '1'
DATA1_IN_SAMPLE_CLK_REQ: out std_logic;
  -- requests data when the input elastic buffer is less than half full.
  -- (flow control)

--// user interfaces
--// Stream2. 8-bit I/O read/write transactions at I/O address 0
-- Synchronous with CLK_P clock
DATA2_OUT: out std_logic_vector(7 downto 0);
DATA2_OUT_SAMPLE_CLK: out std_logic;
  -- read DATA2_OUT at rising edge of CLK_P when
  -- DATA2_OUT_SAMPLE_CLK = '1'
  -- Note1: the user is responsible for checking –
  -- DATA2_OUT_BUFFER_EMPTY before reading.
  -- Note 2: When the elastic buffer is not empty, DATA2_OUT is
  -- present at this interface even before requesting it. The request –
  -- DATA2_OUT_SAMPLE_CLK_REQ
  -- only moves the read pointer to the next read location.
DATA2_OUT_BUFFER_EMPTY: out std_logic;
DATA2_OUT_SAMPLE_CLK_REQ: in std_logic;
  -- requests data. If no data is available in the buffer, the
```

```
  -- DATA2_OUT_SAMPLE_CLK will stay low.
  -- (flow control)

DATA2_IN: in std_logic_vector(7 downto 0);
DATA2_IN_SAMPLE_CLK: in std_logic;
  -- read DATA2_IN at rising edge of CLK_P when
  -- DATA2_IN_SAMPLE_CLK = '1'
DATA2_IN_SAMPLE_CLK_REQ: out std_logic
  -- requests data when the input elastic buffer is less than half full.
  -- (flow control)

--// Test Points
-- Test points are under the shield. 6 at the edge connector.
--TEST_POINTS: out std_logic_vector(6 downto 1)
);
end entity;
```

5.3.2 Configuration Header

The configuration Header is a data structure stored in non-volatile memory within the CardBus. It is read by the operating system to determine what kind of CardBus is installed, along with its speed, size and the system resources required by the card. A detailed description of the CardBus Configuration Header is provided below. The operating system reads all the Configuration Header registers sequentially starting from address 0.

Offset	***Data (hex)***	***Description and Interpretation***
0	FF	Vendor ID
	FE	
2	01	Device ID
	01	
4	03	Command Register: Device Memory, I/O response enable
	00	
6	00	Status Register
	00	
8	00	Revision ID
9	00	Class code: Generic PCI device
	00	
	00	
12	00	Cache Line Size
13	00	Latency Timer
14	00	Header Type: single function
15	00	BIST
16*	01	BAR 0 (I/O)
	xx	
	xx	
	xx	
20*	00	BAR 1 (Memory)
	00	
	x0	
	xx	
24	00	CardBus CIS pointer
	00	
	00	
	00	
28	00	Subsystem Vendor ID
	00	
30	00	Subsystem ID
	00	
32	00	Expansion ROM Base Address
	00	
	00	
	00	
36	00	Capabilities Pointer
37	00	Reserved
	00	
	00	
40	00	Reserved
	00	
	00	
	00	
44	01	Interrupt Line
45	00	Interrupt Pin
46	00	Min_Gnt
47	00	Min_lat

* ‘x’ represents that the host writes to these bits.

5.3.3 Synthesis Statistics

The FPGA size occupied by the CardBus component is as follows:

Design Summary:
Logic Utilization:
Number of Slice Flip Flops: 270 out of 7,168 3%
Number of 4 input LUTs: 469 out of 7,168 6%
Logic Distribution:
Number of occupied Slices: 366 out of 3,584 10%
Number of Slices containing only related logic: 366 out of 366 100%
Number of Slices containing unrelated logic: 0 out of 366 0%
Total Number 4 input LUTs: 551 out of 7,168 7%
Number used as logic: 469
Number used as a route-thru: 82
Number of bonded IOBs: 120 out of 173 69%
IOB Flip Flops: 44
Number of Block RAMs: 5 out of 16 31%
Number of GCLKs: 2 out of 8 25%

5.4 VHDL code template

A VHDL template project is available on-line at www.comblock.com/download/com1300template_001.zip

The template project (**-C** option) includes:

- Top-level VHDL source code (.vhd), for CardBus interface.
- NGC components for config_c, iorw_c and memory_c, and the SDRAM driver.
- the constraint file (.ucf) listing all pin assignments.
- The Xilinx project with the synthesis and implementation settings.
- The resulting bit files (.mcs) ready to be loaded into flash memory.