Core1553 BRT

User Guide

For more information about Actel's products, call 888-99-ACTEL
or visit our Web site at http://www.actel.com

Actel Corporation • 955 East Arques Avenue • Sunnyvale, CA USA 94086
U.S. Toll Free Line: 888-99-ACTEL • Customer Service: 408-739-1010 • Customer Service FAX: 408-522-8044
Customer Applications Center: 800-262-1060 • Customer Applications FAX: 408-739-1540

Actel Europe Ltd. • Maxfli Court, Riverside Way • Camberley, Surrey GU15 3YL • Tel: +44 (0)1276.401452 •
Fax: +44 (0)1276.401490

Actel Japan • EXOS Ebisu Bldg. 4F • 1-24-14 Ebisu Shibuya-ku • Toyko 150 • Japan
Tel: +81 (0)334-457-671 Fax: +81 (0)334-457-668

5029140-0

Core1553BRT

User Guide

Actel Corporation, Sunnyvale, CA 94086

Printed in the United States of America

Part Number: 5029140-0

Release: October 2002

Trademarks

Table of Contents

Introduction 1

Document Organization 2

1 Running Simulation. 3

Verification Environment. 3

2 Running Synthesis. 7

Synthesis for Netlists 7

Synthesis for RTL. 7

3 Running Layout 9

4 Verification Testbench. 11

Interactive Operation 15

Verification Tests 16

5 VHDL Testbench. 19

Using the VHDL QTBEncDec Module 21

6 Verilog Testbench. 27

Using the Verilog QTBENCDEC Module. 29

7 Implementation Hints 35

External Command Word Legality Example. 37

Modifying the Backend Address Map 40

Modifying the Backend Interrupt Vector. 44

Connecting the Backend to Internal FPGA Memory 47

Buffer Management. 47

A VHDL Testbench Procedure and Function Calls 49

B Product Support 53

Actel U.S. Toll-Free Line 53

Customer Service 53

Customer Applications Center 54
Guru Automated Technical Support 54
Web Site. 54
FTP Site 55
Contacting the Customer Applications Center 55
Worldwide Sales Offices 56

Introduction

Core1553BRT provides a 1553B Remote Terminal, compliant with MIL-STD-1553B that you can implement in various Actel FPGA families (including the RTSX-S radiation-tolerant devices). Three versions of the core are available: an evaluation version that allows core simulation with the Actel Libero toolset or Modelsim, a netlist version that provides netlists and pre-compiled test benches and finally, an RTL version with full access to the source code. The directory structure is shown in Figure 1.

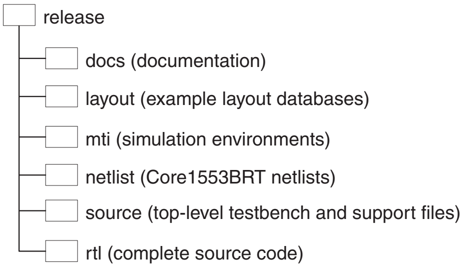
release
docs (documentation)
layout (example layout databases)
mti (simulation environments)
netlist (Core1553BRT netlists)
source (top-level testbench and support files)
rtl (complete source code)

Figure 1. Core1553BRT Directory Structure

All the above directories are provided with the RTL version. The netlist version includes all directories apart from the RTL directories and the evaluation version only includes the *mti* and *docs* directories.

The *netlist* directory contains thirty-two netlists, eight for each supported family. This core release supports the SX-A, RTSX-S, AX and APA families. For each family 12MHz and 16MHz netlist versions are provided with I/O and without I/O pads in both VHDL and Verilog formats. The SX-A netlists may also be used with the SX and RTSX families.

The “with I/O” netlists are used to create the layout databases provided in the *layout* directory; these databases may be used to program an FPGA device. If necessary, you can alter the pin locations within the Designer layout tool.

The “without I/O” netlists are intended for use when the core is instantiated in a user design. During the synthesis process the synthesis tool inserts the I/O pads.

The *source* directory contains the top level VHDL code for the testbenches and also some example support source files to ease design integration.

The *rtl* directory is provided to RTL licensees of the core. This directory contains VHDL and Verilog subdirectories containing the RTL source files.

Document Organization

The *Core1553BRT User Guide* contains the following chapters:

Chapter 1 - **Running Simulation**

Chapter 2 - **Running Synthesis** describes how to run two types of synthesis, netlist and RTL.

Chapter 3 - **Running Layout**

Chapter 4 - **Verification Testbench** describes the VHDL based verification testbench architecture.

Chapter 5 - **VHDL Testbench** describes the customizable VHDL testbench architecture.

Chapter 6 - **Verilog Testbench** describes your customizable Verilog testbench architecture.

Chapter 7 - **Implementation Hints** provides tips on how to integrate the Core1553BRT into your system.

Appendix A - **VHDL Testbench Procedure and Function Calls** provides an overview of Actel's Core1553BRT testbench and a guide to procedures. It also describes the syntax for a variety of existing procedures and functions.

Appendix B - **Product Support**

1

Running Simulation

Core1553BRT has three separate simulation testbenches, a full verification environment for the 1553B core, and two user testbenches (one in Verilog and the other in VHDL). You can modify the user testbenches for Core1553BRT integration in your system. Details of these testbenches are provided in Chapters 4, 5, and 6.

Verification Environment

Seven separate directories provided within the *mti* directory enable you to rerun core verification (Figure 1-1).

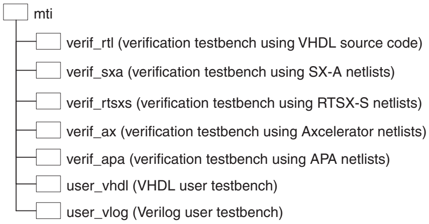
mti
verif_rtl (verification testbench using VHDL source code)
verif_sxa (verification testbench using SX-A netlists)
verif_rtsxs (verification testbench using RTSX-S netlists)
verif_ax (verification testbench using Axcelerator netlists)
verif_apa (verification testbench using APA netlists)
user_vhdl (VHDL user testbench)
user_vlog (Verilog user testbench)

Figure 1-1. mti Library Directories

The Core1553BRT verification testbench is provided as pre-compiled ModelSim library files.

To run the verification testbench:

1. **Start ModelSim.**
2. **Change the directory to the *mti/verif_xxx* directory.** For example:

   ```
   mti/verif_rtl
   ```
3. **Refresh the simulation library with “do refresh.do” command.**
4. **Run the simulation with the “do runsim.do” command.** The simulation starts and prompts for which test to run. Enter 2 to run the complete verification suite (may take several hours) or 1 to run for a short time.

If you use any of the netlist versions then you need to set up the Vital Libraries within the Actel Libero/Designer system (See the *Libero User's Guide*). The *verif_rtsxs* directory uses the SX-A Vital library. Netlist simulation runs are much longer than the RTL simulation run.

RTL licensees can use the compvhdl.do script instead of the "refresh.do" script to recompile the core and testbenches from the provided VHDL source code. The *verif_rtl* directory also has a "compvlog.do" script that simulates the Verilog source code rather than the VHDL source code within the VHDL test harness. If you wish to use the "compvlog" script, you must have a dual-language Modelsim license.

VHDL Testbench

This is provided as a pre-compiled ModelSim library (the top-level source files are provided in the source directory). The pre-compiled files are in the *mti/user_vhdl* directory.

To run the user testbenches:

1. **Start ModelSim.**
2. **Change the directory to the mti/user_vhdl directory.**
3. **Refresh the simulation library with the "do refresh.do" command.**
4. **Run the simulation "do runsim.do".** The simulation starts and runs several 1553B messages.

RTL licensees can use the compvhdl.do script instead of the refresh.do script to recompile the testbenches from the provided VHDL source code.

Verilog Testbench

This testbench is provided as a pre-compiled ModelSim library (the top-level source files are provided in the source directory). The pre-compiled files are in the *mti/user_vlog* directory.

To run the user testbenches:

1. **Start ModelSim.**
2. **Change the directory to the mti/user_vlog directory.**
3. **Refresh the simulation library with the “do refresh.do” command.**
4. **Run the simulation with the “do runsim.do” command.** The simulation starts and runs several 1553B messages.

RTL licensees can use the “compvlog.do” script instead of the refresh.do script to recompile the testbenches from the provided Verilog source code.

2

Running Synthesis

This chapter describes how to run synthesis for both netlists and RTL.

Synthesis for Netlists

The release contains both 16MHz and 12MHz versions of the core for the supported families. Instantiate the Core1553BRT netlist in your design. The top-level entity/module names for the cores are as follows:

- 16MHz - RT1553B12
- 12MHz - RT1553B16

To simplify instantiation the component declarations are provided in the source directory (*rtcomps.vhd* and *rtcomps.v*).

Use either the Verilog or VHDL netlist without I/O cells. For example, the VHDL netlist for SX-A devices operating at 16MHz is rt1553b_withoutio_sxa_16.vhd, which is found in the netlists directory.

Tie the configuration inputs (WRTTSW, WRTCMD, EXTMDATA, ASYNCIF, TESTTXTOUT, BCASTEN, SA30LOOP & INTENBBR) high or low as desired. During synthesis the synthesis tool optimizes the netlist by removing unnecessary logic. You can also drive these inputs with logic if required, but the core implementation is slightly larger.

Use a clock network to drive the CLK input to the core. Do not use a clock network to drive the RSTINn input, since it is gated internally.

When using Synplicity with a Verilog flow you need to include the Synplicity supplied macro library in the source file list. These libraries are located in the following directories:

```
synplify_install_dir/lib/actel
synplify_install_dir/lib/proasic
```

Use the macro library file that corresponds to your target architecture.

Synthesis for RTL

Instantiate the Core1553BRT top-level entity/module in your design. The top-level entity/module names for the cores are as follows:

- RTL - RT1553B
- 16MHz - RT1553B12
- 12MHz - RT1553B16

You can use an additional top-level option with the RTL version; there is a generic to set the operating frequency to 12MHz or 16MHz. To ease instantiation the component declarations are provided in the source directory (rtcomps.vhd and rtcomps.v)

The VHDL example below shows the included files for a 16MHz core Synplicity project. If the RT1553B is instantiated at the top level in your design then you do not need the RT1553B12 or RT1553B16 files.

```
add_file -vhdl -lib work "rtl/vhdl/core/backend.vhd"
add_file -vhdl -lib work "rtl/vhdl/core/cwlegalsyn.vhd"
add_file -vhdl -lib work "rtl/vhdl/core/decoder.vhd"
add_file -vhdl -lib work "rtl/vhdl/core/encoder.vhd"
add_file -vhdl -lib work "rtl/vhdl/core/RT1553B.vhd"
add_file -vhdl -lib work "rtl/vhdl/core/RT1553B16.vhd"
```

The Verilog example below is associated with a 12MHz core Synplicity project. If the RT1553B is instantiated as the top level in your design then you do not need the RT1553B12 or RT1553B16 files.

```
add_file -verilog -lib work "rtl/verilog/core/backend.v"
add_file -verilog -lib work "rtl/verilog/core/cwlegalsyn.v"
add_file -verilog -lib work "rtl/verilog/core/decoder.v"
add_file -verilog -lib work "rtl/verilog/core/encoder.v"
add_file -verilog -lib work "rtl/verilog/core/RT1553B.v"
add_file -verilog -lib work "rtl/verilog/core/RT1553B12.v"
```

Tie the configuration inputs (WRTTSW, WRTCMD, EXTMDATA, ASYNCIF, TESTTXTOUT, BCASTEN, SA30LOOP & INTENBBR) high or low as desired. During Synthesis the synthesis tool optimizes the netlist, removing the unused logic. You can also drive these inputs with logic if required, but the core implementation is slightly larger.

Use a clock network to drive the CLK input to the core. Do not use a clock network drive the RSTINn input, since it is gated internally.

3

Running Layout

Once synthesized, run the core through the Designer layout tool to produce the programming files for the FPGA. The Core1553BRT meets timing in all supported families, so you can use standard layout flows. Actel recommends that you set the 1553B clock input to 12MHz or 16 MHz and enable timing-driven layout.

The release contains a *layout* directory with a fully placed-and-routed core for each of the four supported families in both the 12MHz and 16MHz core versions. The *netlists* directory contains the matching eight netlists (for example, rt1553b_withio_sxa_16.v). These netlists have IO pads inserted around the core; do not use them when the core is integrated in another design.

4

Verification Testbench

Actel has developed a 1553B verification testbench that you can use to verify the core performance per the 1553B specification. The testbench is coded in VHDL and contains four Core1553B Remote Terminals connected to a bus control function and backend interfaces. A procedural testbench controls the various blocks and implements the tests. The source code is not made available with netlist licenses of the core (Figure 4-1).

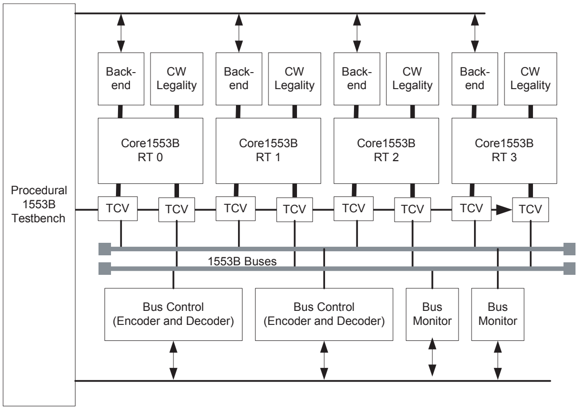

Figure 4-1. Verification Testbench

The testbench includes four remote terminals (RTs) to test core variants, each with different setups of the core configuration inputs (Table 4-1).

Table 4-1. Verification Testbench RT Configuration

RT	**Clock Speed**	**Memory Interface**	**Command Word Legality**[a]
0	12 MHz	Synchronous	Internal
1	16 MHz	Asynchronous	Internal
2	16 MHz	Synchronous	Internal
3	16 MHz	Asynchronous	External

a. All command word legality interfaces are external for netlist simulation.

The testbench uses a command word legality module that disables sub-address 26 & 27 for transmit commands. For receive commands sub-address 25 is disabled and sub-address 27 is only enabled for word counts 1 to 9. Page 37 shows the source code for a command legality module implementing this behavior.

The procedural testbench has direct control of the bus control modules, the transceivers, and the backend interfaces (including all the backend core inputs). This direct control allows the testbench to initialize the backend RAM contents, and to verify the contents after each transfer. The bus controller function has the capability to inject errors and vary the data rate to verify the 1553B decoder behavior.

During operation, the testbench verifies all command, data, and status words. All memory accesses are verified and the resultant transfer status word for every message is checked

During invocation the top-level generics may be set to alter the simulation. These generics are specified in Table 4-2.

Table 4-2. Verification Testbench RT Configuration

Generic	Type	Default Value	Function
ENBUSMON	Boolean	False	Enables the Bus monitor function. The testbench displays every 1553B word that is transmitted on the buses
ENRAMMON	Boolean	False	Enables the RAM monitor function. The testbench displays all the memory reads and writes performed by each of the cores
RUNTEST	Integer	0	Allows the Testbench to run without user input. Forces the Testbench to run the test number as defined by the menus (see below); if RUNTEST is greater than 10, then the testbench runs test number (n-10) and quits (for example, if n=12, then the testbench runs test number (12-10=2), which is the Actel Tests - Standard Mode, and then quits the simulation)
NETLIST	Boolean	False	When TRUE all cores use an external command word legality module. The testbench expects some of the sub-addresses to be disabled. The netlists provided for the core enable all of the sub-addresses in the RT.

When you run the testbench it asks which tests you want to run. The options are as follows (the options are summarized in Table 4-3):

```
# 1553B Test Harness - Actel IP Solutions Group
#  Production Version 22Aug02
#
#  Test Options
#  1 : Quick Run
#  2 : Actel Tests - Standard Mode
#  3 : Actel Tests - Address Mapper Enabled
#  4 : RT Test Plan
#  5 : Actel Tests - Short mode (i.e. fewer tests)
#  9 : Do Everything
#  A : Do Everything, Monitors Off and Quit
#  B : Turn On Bus Monitors
#  R : Turn On RAM Monitors
#  P : Pause Simulation - allows waves to update
#  X : Run for a single 1553B Word; i.e. 20µs
#  M : Send a message    M BUS RT TX SA WC [SEED INC]
#  D : Display RT Memory D RT TX SA
#  S : Set RT Memory     S RT TX SA SEED INC
#  Q : Quit
#
#  Enter Option, for  demo use 1 ?
```

Table 4-3. Verification Testbench Menu Summary

Menu Command	Action
Quick Run	This runs a few simple 1553B command sequences to demonstrate that the core is functioning
Actel Tests - Standard Mode	This runs the complete set of Actel verification tests. For the RTL code (*verif_rtl* directory) this takes several hours depending on the computer used. For the netlist versions the simulation time is significantly longer.
RT test plan	This implements a subset of the protocol tests specified in the MIL-HDBK-1553A handbook. This takes a very long time to run as thousands of 1553B command words need to transmitted and verified.

Table 4-3. Verification Testbench Menu Summary (Continued)

Menu Command	**Action**
Do Everything	Runs the three options listed above; i.e. Quick, Actel, and the Test plan
Actel Tests - Address Mapper Enabled	This runs the Standard Mode tests but uses an address mapping function (as on page 41)
Actel Tests - Short Mode	This runs a subset of the Standard Mode tests; the run time is much shorter than for the standard tests
Do Everything & Quit	This runs the three options above and then quits the simulation. Setting the RUNTEST generic to 9 can automatically run this test option
RAM Monitors	The testbench can display a message every time the backend RAM's are written to or read from. This option enables or disables the RAM monitors. When enabled the testbench runs more slowly due to the printing overhead in VHDL
Bus Monitors	The testbench includes 1553B Bus monitors that display every word transmitted on the 1553B buses. This option enables or disables the BUS monitors. When enabled the testbench will run slower due to the printing overhead in VHDL.
Pause Simulation	Exits the simulation and returns to the vsim environment. This may be required to cause the waves window to update. Simulation can be simply restarted using the “run -all” command
Run 1553B Word	Simply allows the simulation to run for 20us.
Send a Message	Allows interactive 1553B message creation
Set RT memory	Allows the values in one of the RT memories to be set
Display RT Memory	Displays the contents of one of the RT memories

Interactive Operation

The verification testbench allows you to create 1553B messages and transmit from the simulator command line. Three commands are provided that support this feature (M, S and D), their parameters are given below. All parameters are decimal integers separated by spaces or commas. If a number begins with a "#", "A-F" or "a-f" then it is interpreted as a hexadecimal value. Basic error checking is performed on the entered values.

Message Parameter M

Consider the following example:

```
M  BUS RT TX SA WC [SEED [INC]]
```

- M - Transmit a message
- BUS - Specifies the Bus to use, 0 or 1
- RT - Specifies the RT number, 0-31
- TX - RT transmit or receive, 1=Transmit 0= Receive
- SA - Sub-address to use, 0-31
- WC - Number of words to transmit or receive, 0-32
- SEED - Sets the first data used for the message to this value. If the RT is to receive data, then the bus controller transmits the data. If the RT is to transmit, then the testbench initializes the RT backend RAM for the appropriate sub-address. If no data is provided then the RT receive data is 0000 and the transmit data is whatever the RT memory contains
- INC - Increments each data word in the message by this value. If not specified defaults to zero so all data words contain the same value

Message Parameter S

Consider the following example:

```
S RT TX SA SEED [INC]
```

- S - Set RT memory
- RT - Specifies the RT number, 0-31
- TX - RT transmit or receive memory, 1=Transmit 0= Receive

- SA - Which Sub-address, 0-31
- SEED - Sets the first data value.
- INC - Increments each data word in the message by this value. If not specified defaults to zero so all data words contain the same value

Message Parameter D

Consider the following example:

```
D RT TX SA
```

- D - Display RT memory
- RT - Specifies the RT number, 0-31
- TX - RT transmit or receive memory, 1=Transmit 0= Receive
- SA - Which Sub-address, 0-31

Verification Tests

The verification testbench includes test procedures to check the following:

Simple Messages BC-RT and RT-BC

- Sub-address Word-Count Combinations

RT-to-RT Messages

Mode codes

- Verifies all codes

Broadcast

- Status word settings
- All mode codes verified

Illegal Commands (legality interface)

- Mode code
- Disabled sub-addresses
- Normal, RT-to-RT and Broadcast
- Internal and External legality logic

Special Features

- Sub-address 30 loopback
- TSW enabled and disabled
- Command word memory write enabled and disabled
- Address Mapping Functions
- Interrupt Vector Extension Functions

Error Conditions

- Variable bit rates +/- 10,000Hz (1%)
- Variable backend GNT and WAIT delays
- Parity and Manchester Encoding errors
- Synchronization pattern corruption
- RT-to-RT illegal command words
- Word count errors
- Word gaps
- Transmitter time-out
- Status Word flag bits
- Superseding command acceptance
- RT address parity logic
- Receive on disabled bus
- Transmitter overrun
- BIT word values
- Loopback logic
- Extra Command and Data Words
- Noise on the data bus
- Superseding Commands on second bus

5

VHDL Testbench

Actel provides an example testbench that you can use as the starting point for design verification of the core in your design. A block diagram of the testbench is shown in Figure 5-1.

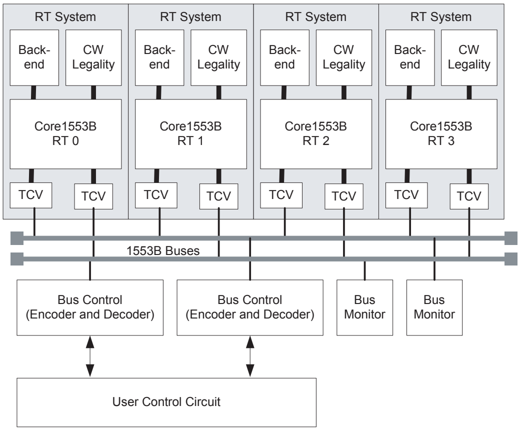

Figure 5-1. VHDL Testbench

The testbench creates an RT System (QRTSystem) by adding the transceivers, backend interface and command legality interface to the core. The top level (Qtbench) includes four of these cores. All the cores in this case are identical. The source code modules used are listed in Table 5-1.

Table 5-1. VHDL Testbench Modules

Entity	Source Provided	Description
Qtbench	Yes	The testbench top level. This contains the bus control function and several remote terminals
QTBEncDec	No	Test block that emulates a bus controller. Transmits 1553B command words and data words, and then decodes the status and data words generated by an RT in response
QBusmon	No	1553B bus monitor. Monitors the bus and reports on the bus traffic
QRTsystem	Yes	Hierarchical block with the core plus transceiver, backend and command word legality blocks to the core
QBusTransceiver	No	This simply takes the six unidirectional signals from the core and models a Transceiver connected to a bus
QTBBackend	No	This connects to core backend interface and provides the following functions: 1) Implements a asynchronous 2Kx16 or 8Kx16 memory block, which provides the 32 receive and transmit sub-addresses 2) Has a built-in address mapping function 3) Loops back the receive sub-address locations to the transmit memory. On a good message received interrupt the correctly received words are copied from RX sub-address to the TX sub-address. If the address mapping function is enabled then the synchronize with data word is copied to the transmit vector word memory location 4) Generates the interrupt acknowledge.
CWLegality	Yes	Implements the external command validity checking

The testbench uses a command-word legality module that disables sub-address 26 & 27 for transmit commands. For receive commands sub-address 25 is disabled and sub-address 27 is only enabled for word counts 1 to 9. Page 37 shows the source code for a command legality module implementing this behavior.

The Core1553B ModelSim library (in the *mti/verif_rtl* directory) contains compiled models for the complete environment. Design source code of the top-level blocks is provided in the *source* directory to enable you to create your own simulation environment using this testbench as a starting point. Examine the source files for QTbench and QRTSystem to obtain a full understanding of the core operation.

The QTbench has a top level generic CMODE that you can set to 0 or 1. When 0, the core is configured with WRTTSW, WRTCMD and EXTMDATA as "100". When 1 these values are "011" and the backend module implements an address mapper function as described in "Implementation Hints" on page 35.

Using the VHDL QTBEncDec Module

You can instantiate the QTBEncDec module in your design and use it to initiate 1553B messages. The top level of the module is shown in the code below, along with a description of these ports (Table 5-2).

```
entity QTBENCDEC is
  port ( CLK16    : in    std_logic;
         RSTn     : in    std_logic;
         STOPCLK  : in    BOOLEAN;
         START    : in    BOOLEAN;
         QMSG     : in    TQMSGREQ;
         BUSY     : out   BOOLEAN;
         QOUT     : out   TQMSGOUT;
         BUSPOS   : inout std_logic;
         BUSNEG   : inout std_logic
       );
end QTBENCDEC;
```

Table 5-2. TBEncDec Port Descriptions

Port	Dir	Type	Function
CLK16	In	std_logic	Clock source for the encoder and decoder. Must be 16MHz
RSTn	In	std_logic	Active low asynchronous reset, must be pulsed low at the start of simulation
STOPCLK	In	Boolean	The encoder has an internal clock generator; when this input is TRUE the clock generator is halted. This allows the simulator to exit gracefully. This input can be tied permanently FALSE to disable the STOPCLK feature

START	In	Boolean	This input is pulsed TRUE to start a 1553B message. If it is not synchronized to a clock and only needs pulsed for a simulation delta cycle, use the following code:
			START <= TRUE;
			wait for 0 ns
			START <= FALSE;

QMSG	In	TQMSGREQ	Input record structure that defines the message that will be transmitted, see below.
BUSY	Out	Boolean	Indicates that the encoder/decoder is busy
QOUT	Out	TQMSGOUT	Output record structure containing message data transmitted on the bus
BUSPOS	Inout	std_logic	Connects to the positive side of the 1553B bus
BUSNEG	Inout	std_logic	Connects to the negative side of the 1553B bus

To initiate a message a simple record structure is set up and the START input is strobed. The module asserts BUSY until the 1553B message is completed and then de-asserts BUSY. The QOUT record structure contains the data sent and received on the bus. The two record structures used here are shown in Table 5-3 and Table 5-4.

Table 5-3. TMSGREQ Record Structure

Element	**Type**	**Default**	**Function**
RT	INTEGER, 0 to 31	0	Command word RT value. Broadcast is supported when the RT number is 31
TX	INTEGER, 0 to 1	0	Command word TX value
SA	INTEGER, 0 to 31	0	Command word SA value
MCWC	INTEGER, 0 to 32	0	Word count or mode code value. For data transfers values 1 to 32 should be used, for mode codes values 0 to 31
RTRT	Boolean	FALSE	If TRUE a RT-to-RT command pair is transmitted. The transmit command word uses the RT, TX and SA elements
RT2	INTEGER, 0 to 31	0	RT command word value for the receive RT in RT-to-RT messages
SA2	INTEGER, 0 to 31	0	SA command word value for the receive RT in RT-to-RT messages
DATA	PACKET	Unknowns i.e. 'U'	Data to be transmitted for an RT receive message This is an array(0 to 31) of WORD; WORD is std_logic_vector(15 downto 0). Index 0 is used for mode code data. For example: DATA(0) <= "0000111100001111"; DATA(1) <= to_word(16#1234#) DATA <= initdata(16#1000#); DATA <= initdata(16#5555#,0); The first example sets the first data word using standard VHDL assignments to std_logic_vector. The second example uses a function provided in one of the underlying Core1553B packages to set the word to the hexadecimal value 1234. The third example uses another function call to set the complete data pattern to an incrementing value starting at 1000 hex. The final example sets the complete data pattern to 5555 hex; the second argument is the increment between consecutive data words.

Table 5-4. TQMSGOUT Record Structure

Structure Element	Type	Function
OKAY	Boolean	Indicates that the message was correctly processed with no errors
COUNT	INTEGER	Number of words received
CW1	WORD	First command word
CW2	WORD	Second command word. If no second command word it will contain '-' in all 16 bits
SW1	WORD	First status word. If no status word it will contain '-' in all 16 bits
SW2	WORD	Second status word. If no status word it will contain '-' in all 16 bits (only for RTRT messages)
DATA	PACKET	Data packet for the message. When the RT is receiving it will contain a copy of the transmitted data, when transmitting the data the RT transmitted. On RT-to-RT messages the data transferred between the RT's will be provided. For mode codes the data word will in the first location i.e. DATA(0)

The VHDL code below shows a simple code fragment that generates a 10 word BC-to-RT 1 sub-address 5 message with data incrementing from 1200 hex, and then displays the message. The “print_msgout” is a procedure provided in the underlying Core1553B package that displays the message details; it needs to be passed the two record structures described above.

```
signal BUSASTART : BOOLEAN;
signal BUSAMSG   : TQMSGREQ;
signal BUSABUSY  : BOOLEAN;
signal BUSAOUT   : TQMSGOUT;

begin

process
 begin
  BUSAMSG.RT   <= 1;
  BUSAMSG.TX   <= 0;
  BUSAMSG.SA   <= 5;
  BUSAMSG.MCWC <= 10;
  BUSAMSG.RTRT <= FALSE;
  BUSAMSG.DATA <= initdata(16#1200#);

  -- Do the message
  BUSASTART <= TRUE;
  wait for 0 ns;
  BUSASTART <= FALSE;
  wait for 0 ns;
  while BUSABUSY loop
    wait for 1 us;
  end loop;

  -- Display The Data Transmitted
  print_msgout(BUSAMSG,BUSAOUT);
  wait;
end process;
```

6

Verilog Testbench

Actel provides an example Verilog testbench that you can use as the starting point for design verification of the core within your design. A block diagram of the testbench is shown in Figure 6-1.

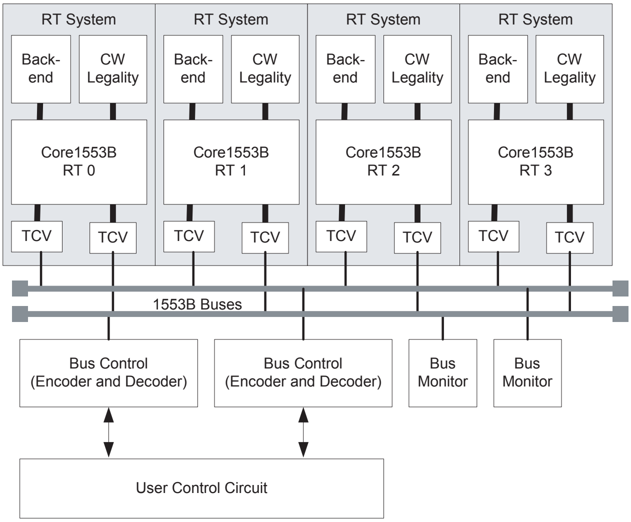

Figure 6-1. Verilog Testbench

The testbench creates an RT System (QRTSYSTEM) by adding the backend interface and command legality interface to the core. The top level (QTBENCH) includes four of these cores. All the cores in this case are identical. Table 6-1 lists the source code modules.

Table 6-1. Verilog Testbench Modules

Module	**Source Provided**	**Description**
QTBENCH	Yes	The testbench top level. This contains the bus control function and several remote terminals
QTBENCDEC	No	Test block that emulates a bus controller. Transmits 1553B command words and data words, and then decodes the status and data words generated by an RT in response.
QRTSYSTEM	Yes	Hierarchical block with the core plus a transceiver, backend and command word legality block to the core.
QTBBACKEND	Yes	This connects to the core backend interface and provides the following functions: 1) Implements an asynchronous 2Kx16 or 8Kx16 memory block providing the 32 receive and transmit sub-addresses 2) Has an in-build address mapping function 3) Loops back the receive sub-address locations to the transmit memory. On a good message received interrupt the correctly received words are copied from RX sub-address to the TX sub-address. If the address mapping function is enabled then the synchronize with data word is copied to the transmit vector word memory location. 4) Generates the interrupt acknowledge
CWLEGALITY	Yes	Implements the external command validity checking

The testbench uses a command word legality module that disables sub-address 26 & 27 for transmit commands. For receive commands sub-address 25 is disabled and sub-address 27 is only enabled for word counts 1 to 9. “External

Command Word Legality Example" on page 37 the source code for a command legality module implementing this behavior.

The Core1553B ModelSim library (in the mti/user_vlog directory) contains compiled models for the complete environment. Design source code of the top-level blocks is provided (in the source directory) to enable you to create your own simulation environment using this testbench as a starting point. Examine the source files for QTBENCH and QRTSYSTEM to obtain a full understanding of the core operation.

The QTBENCH has a top-level parameter CMODE that can be set to 0 and 1. When 0 the core is configured with WRTTSW, WRTCMD and EXTMDATA as "100". When 1 these values are "011" and the backend module implements an address mapper function as described in "Implementation Hints" on page 35.

Using the Verilog QTBENCDEC Module

You can instantiate the QTBENCDEC module in your design and use it to initiate 1553B messages. The top level of the module is shown below, along with a description of these ports (Table 6-2).

```
module QTBENCDEC (CLK, RSTN,
                   BUSPOS, BUSNEG,
                   TXSTROBE, TXCW, TXDATA,
                   RXSTROBE, RXSTAT, RXDATA,
                   BUSY
                  );
input  CLK;
input  RSTN;
inout  BUSPOS;
inout  BUSNEG;
input  TXSTROBE;
input  TXCW;
input  [15:0] TXDATA;
output RXSTROBE;
output [ 3:0] RXSTAT;
output [15:0] RXDATA;
output BUSY;
```

Table 6-2. Verilog TBENCDEC Port Descriptions

Port	**Direction**	**Function**
CLK	in	16MHz clock source for the encoder and decoder. All inputs are sampled on the rising clock edge, and outputs are registered on the rising clock edge
RSTn	in	Active low asynchronous reset, must be pulsed low at the start of simulation
BUSPOS	inout	Connects to the positive side of the 1553B bus
BUSNEG	inout	Connects to the negative side of the 1553B bus
TXSTROBE	in	This input is pulsed high for a clock cycle to load a 1553B word into the encoder
TXCW	in	0: The encoder transmits a data word 1: The encoder transmits a command word
TXDATA	in[15:0]	Transmit word
RXSTROBE	out	Indicates that the RXSTAT and RXDATA outputs contain valid data
RXSTAT	out[3:0]	Provides status information on the RXDATA output; see Table 6-3 for a summary
RXDATA	out[15:0]	Received word
BUSY	out	Indicates that either the encoder or decoder is busy. It is active when Transmit data is queued in the transmit FIFO or is being transmitted

Table 6-3. RXSTAT Value Definitions

Bit	Function	Description
0	Type	0: Data Word 1: Command/Status Word
1	Burst	Indicates that the data word was contiguous with the previous one
2	Error	Indicates that an encoding error was detected in the word
3	From us	Indicates that the QTBENCDEC transmitted the word

The QTBENCDEC contains a 1553B encoder and decoder. A transmit FIFO is provided, enabling 64 words of transmit data to be loaded into the module. The maximum 1553B message length that the encoder is required to transmit is 33 words. Receive data is strobed out of the module. All 1553B data words are output from this module.

The code below shows a simple code fragment that generates a 30 word BC to RT 1 sub-address 10 message with data incrementing from 4096 decimal.

```
/**************************************************************************/
// The Bus Controller Modules
//

QTBENCDEC BCA
  (.CLK(CLK),
   .RSTN(RSTN),
   .BUSPOS(BUSAPOS),
   .BUSNEG(BUSANEG),
   .TXSTROBE(BCA_TXSTB),
   .TXCW(BCA_TXCW),
   .TXDATA(BCA_TXDATA),
   .RXSTROBE(BCA_RXSTB),
   .RXSTAT(BCA_RXSTAT),
   .RXDATA(BCA_RXDATA),
   .BUSY(BCA_BUSY)
  );

/**************************************************************************/
// Store the data output by the decoder
//
// STAT[3:0] = FROMUS BURST ERROR CW

always @(posedge CLK)
 begin
   if (BCA_INIT==1'b1)  BCA_COUNT = 0;
   if (BCA_RXSTB == 1'b1 )
    begin
      BCA_STORE[BCA_COUNT] = {BCA_RXSTAT, BCA_RXDATA };
      BCA_COUNT = BCA_COUNT+1;
    end
 end

/**************************************************************************/
// Main Test Procedure
//

reg [15:0] CW;
reg [15:0] DW;
reg [ 4:0] RT5BIT;
integer  i;
integer  RT;

initial
  begin
    RSTN = 1'b0;
    BCA_TXSTB = 1'b0;// Used to load a word into the transmitter
    BCB_TXSTB = 1'b0;
    BCA_INIT  = 1'b1;// Used to reset the store pointer on the Decoder
    BCB_INIT  = 1'b1;
    #312700;
    @(negedge CLK);
    RSTN = 1'b1;

    $display("Core1553BRT Verilog Test Harness Production 22Aug02");

    $display("Receive BC-RT 1 SA 10 WC 8 Message on BUS A");
    CW = { 5'b00001, 1'b0, 5'b01010, 5'b01000 };
    transmit_CW(0,CW);
    for ( i=1; i<=8; i=i+1) transmit_DW(0,4095+i);
    wait_to_complete(0);
    display_bus(0);
    #9000000;

    $display("Transmit RT-BC 1 SA 10 WC 6 Message on BUS B");
    CW = { 5'b00001, 1'b1, 5'b01010, 5'b00110 };
    transmit_CW(1,CW);
    wait_to_complete(1);
    display_bus(1);
    #9000000;

    $display("Get the Vector Word from each RT");
    for (RT=0; RT<=3;RT=RT+1)
    begin
        RT5BIT = RT;
        CW = { RT5BIT, 1'b1, 5'b00000, 5'b10000 };
        transmit_CW(0,CW);
        wait_to_complete(0);
        $display("RT %0d Got SW %h Got VW %h",
                      RT,BCA_STORE[1][15:0],BCA_STORE[2][15:0]);
    #9000000;
    end

    #4000000;
    $display(" ");
    $display("Simulation complete");
    $display(" ");
    $stop;
    end

endmodule
```

7

Implementation Hints

You can configure the Core1553BRT to provide backend interfaces for a variety of hardware and software requirements.

The backend interface has been designed to simplify backend hardware design; the core supports both synchronous and asynchronous backends with bus arbitration and variable read and write strobe pulse widths.

To accommodate software requirements, you can modify the backend address map and interrupt vectors with simple address mapping functions implemented in hardware (explained below).

Table 7-1 shows some typical applications and the core configurations required.

Table 7-1. Typical Core Implementations

Type	Description	Core Inputs

Standard-CW	A 2Kx16 memory buffer is used with 32 words of memory allocated for each TX and RX sub-address. The 1553B command word is written to memory locations unused by the mode code sub-addresses. The system can read the command word value to determine the number of words that were received.	WRTTSW = '0'
		WRTCMD= '1'
		EXTMDATA= '0'
		Address Mapper = No
		CW Legality = No

Standard-TSW	A 2Kx16 memory buffer is used with 32 words of memory allocated for each TX and RX sub-address. The Core1553B TSW word is written to memory locations unused by the mode code sub-addresses. The system can read the TSW value to determine the number of words that were received. This implementation provided extra status information such as the bus on which the message was received.	WRTTSW = '1'
		WRTCMD= '0'
		EXTMDATA= '0'
		Address Mapper = No
		CW Legality = No

Table 7-1. Typical Core Implementations (Continued)

Type	Description	Core Inputs

Direct Device	No memory is used; the Core1553BRT backend directly connects to the device. In this case all the unused 1553B sub-addresses should be invalidated. If the device only accepts a fixed number of data words then only that word count should be legal for the sub-address in use. Should an error be detected this will be indicated by the interrupt vector and the data should be discarded. No command words or TSW values are written to memory.	WRTTSW = '0'
		WRTCMD= '0'
		EXTMDATA= '0'
		Address Mapper = No
		CW Legality = Yes

Compatibility Mode	The memory allocation emulates another 1553B remote terminal allowing software drivers to be reused.A backend address mapping function is implemented that reads and writes command and data words to the memory addresses used by the remote terminal that the Core1553BRT is replacing.	WRTTSW = '0'
		WRTCMD= '1'
		EXTMDATA= '1'
		Address Mapper = Yes
		CW Legality = Maybe

External Command Word Legality Example

The core provides three ports (USEEXTOK, CMDVAL and CMDOKAY) that allow the legal command word set to be modified. When USEEXTOK is low, the core internally decides command words are legal (the legal command word set is defined in the CORE1553BRTdatasheet). When USEEXTOK is high, an external block decodes the CMDVAL output and generates a CMDOKAY input to indicate legal command words.

The following VHDL and Verilog code blocks implement an external legality checker that:

- Legalizes mode codes as per the Core1553B datasheet
- Disables transmits from sub-address 26 and 27
- Disables receives to sub-address 25
- Only enables word counts 1 to 9 receives to sub-address 27

The source files for these modules are provided in the source directory.

The core allows 3µs for the legality block to decode CWVAL and generate the CMDOKAY value; this can be implemented within the FPGA, as shown below.

VHDL Example

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity CWLEGALITY is
  port ( CWVAL      : in  std_logic_vector(11 downto 0);
         CMDOKAY : out std_logic
        );
end CWLEGALITY;

architecture RTL of CWLEGALITY is
signal BROADCAST     : std_logic;
signal ISMCODE       : std_logic;
signal TX            : std_logic;
signal SA            : std_logic_vector(4 downto 0);
signal WCMC          : std_logic_vector(4 downto 0);

begin

-- Decode incoming Value
BROADCAST   <= CWVAL(11);
TX          <= CWVAL(10);
SA          <= CWVAL(9 downto 5);
WCMC        <= CWVAL(4 downto 0);
```

```
ISMCODE        <= '1' when ( SA="00000" or SA="11111") else '0';

-- This process decodes the Command Word and sets CMDOKAY for legal command words

PLEGAL:
process(BROADCAST,TX,SA,WCMC,ISMCODE)
 variable OK      : std_logic;
 variable MUXSEL : std_logic_vector(5 downto 0);
 begin

   if (ISMCODE='0') then
      -- Data Transfers
      MUXSEL := TX & SA;
      OK := '0';          -- Default is disabled
      -----------------------------------------------------------------------
      -- This case statement legalizes Data transfers to certain sub addresses
      case MUXSEL is
        when "111010" => OK := '0';                         -- SA 26 Disabled for TX
        when "111011" => OK := '0';                         -- SA 27 Disabled for TX
        when "011001" => OK := '0';                         -- SA 25 Disabled for RX
      when "011011" => if WCMC>0 and WCMC <10 then   -- SA 27 Disabled for RX if WC>9
                                   OK := '1';
                                 end if;
        when others   => OK := '1';  -- legalize all other sub addresses
      end case;
      -----------------------------------------------------------------------
      -- Broadcast transmits are not allowed, overrides above case statement
      if BROADCAST='1' and TX='1' then
        OK := '0';  -- Broadcast Transmit is not allowed
      end if;

    else
      -----------------------------------------------------------------------
      -- This case statement legalizes Mode Codes
      MUXSEL := TX & WCMC;
      OK := '1';       -- Default is OKAY
      case MUXSEL is
        when "100000" => --Dynamic Bus Control
                    OK := '0';        -- since we cant do it we message error
        when "100001" => --Synchronise
        when "100010" => --Transmit Status Word
                    OK := not BROADCAST;
        when "100011" => --Initiate Self Test, we set this because we provide BIT word
                    OK := '1';
        when "100100" => --Transmitter ShutDown
        when "100101" => --Override Transmitter Shutdown
        when "100110" => --Inhibit terminal flag
        when "100111" => --Override inhibit terminal flag
        when "101000" => --Reset Remote Terminal
        when "110000" => --Transmit Vector Word
                    OK := not BROADCAST;
        when "010001" => --Synchronise with data
        when "110010" => --Transmit last command
                    OK := not BROADCAST;
        when "110011" => --Transmit bit word
                    OK := not BROADCAST;
```

```
            when "010100" => --Selected Transmitter Shutdown
                    OK := '0';
            when "010101" => --Override Selected Transmitter Shutdown
                    OK := '0';
            when others   => --All other commands all illegal
                    OK := '0';
        end case;
   end if;
   CMDOKAY <= OK;
end process;

end RTL;
```

Verilog Example

```
module CWLEGALITY (CWVAL, CMDOKAY);
  input[11:0] CWVAL;
  output CMDOKAY;
  reg CMDOKAY;
  wire BROADCAST, ISMCODE, TX;
  wire[4:0] SA, WCMC;

  assign BROADCAST = CWVAL[11] ;  // Decode incoming Value
  assign TX = CWVAL[10] ;
  assign SA = CWVAL[9:5] ;
  assign WCMC = CWVAL[4:0] ;
  assign ISMCODE = (SA == 5'b00000 | SA == 5'b11111) ? 1'b1 : 1'b0 ;

  always @(BROADCAST or TX or SA or WCMC or ISMCODE)
  begin : PLEGAL
    reg OK;
    reg[5:0] MUXSEL;
    if (ISMCODE == 1'b0)
    begin
      MUXSEL = {TX, SA};    // Data Transfers
      OK = 1'b0;
      // This case statement legalizes Data transfers to certain sub addresses
      case (MUXSEL)
        6'b111010 : OK = 1'b0;              // SA 26 Disabled for TX
        6'b111011 : OK = 1'b0;              // SA 27 Disabled for TX
        6'b011001 : OK = 1'b0;              // SA 25 Disabled for RX
        6'b011011 : OK = (WCMC > 0 & WCMC < 10); // SA 27 Disabled for RX if WC>9
        default   : OK = 1'b1;              // legalize all other sub addresses
      endcase
     // Broadcast transmits are not allowed, overrides above case statement
      if (BROADCAST == 1'b1 & TX == 1'b1)
        OK = 1'b0; // Broadcast Transmit is not allowed
    end
    else
    begin
      //---------------------------------------------------------------------
      // This case statement legalizes Mode Codes
      MUXSEL = {TX, WCMC};
      OK = 1'b1;
      case (MUXSEL)
```

```
            6'b100000 : //Dynamic Bus Control
                    OK = 1'b0; // since we cant do it we message error
            6'b100001 : //Synchronize
            6'b100010 : //Transmit Status Word
                    OK = ~BROADCAST;
            6'b100011 : //Initiate Self Test, we set this because we provide BIT word
                    OK = 1'b1;
            6'b100100 : //Transmitter Shutdown
            6'b100101 : //Override Transmitter Shutdown
            6'b100110 : //Inhibit terminal flag
            6'b100111 : //Override inhibit terminal flag
            6'b101000 : //Reset Remote Terminal
            6'b110000 : //Transmit Vector Word
                    OK = ~BROADCAST;
            6'b010001 : //Synchronise with data
            6'b110010 : //Transmit last command
                    OK = ~BROADCAST;
            6'b110011 : //Transmit bit word
                    OK = ~BROADCAST;
            6'b010100 : //Selected Transmitter Shutdown
                    OK = 1'b0;
            6'b010101 : //Override Selected Transmitter Shutdown
                    OK = 1'b0;
            default   : //All other commands all illegal
                    OK = 1'b0;
          endcase
        end
        CMDOKAY <= OK ;
      end
    endmodule
```

Modifying the Backend Address Map

The default setting of the Core1553BRT creates 32 receive and 32 transmit sub-addresses each with 32 words of memory, which in turn requires 2048 words of memory. Receive sub-addresses 0 and 31 are used for storing either the 1553B command word or the TSW value.

You may use an external address mapping function to modify the backend address map to match the software models of legacy 1553B remote terminals, allowing software drivers to be reused (Figure 7-1). You can use the CMDVAL output to generate the mapped address function; however, it must be externally latched using the ADDRLAT signal (ADDRLAT is an active high enable for an external D type flip flop).

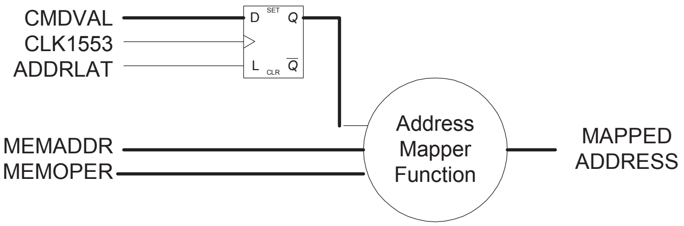

Figure 7-1. External Address Mapper Circuit

A typical address mapping function given below (page 41) re-maps the backend memory map to an 8K-word memory. Each sub-address is allocated 64 words and separate buffers are provided for both broadcast and non-broadcast receive/transmit.

Each non-mode code sub-address consists of 64 words, the command word or TSW value is written to location 0, and the associated data words are stored at locations 32 to 63.

For the mode code sub-addresses, the command word or TSW value is written to location 0 plus the mode code value, and the data word is stored at location 32 plus the mode code value. For the transmit vector word command, the command word is stored in location 16 and the vector word read from location 48. For the synchronize with data command, the command word is stored at location 17 and the data written to location 49. Note that separate address spaces exist for transmit and receive mode codes and broadcast transmit and receive mode codes.

This mapping function is very small; it requires only 23 logic modules in the Actel SXA/RTSX-S families. Consider the code below.

VHDL Address Mapping Function

```
entity ADDRESSMAPPER is
 port ( CLK            : in    std_logic;
        ADDRLAT        : in    std_logic;
        CMDVAL         : in   std_logic_vector(11 downto 0);
        MEMADDR        : in   std_logic_vector(10 downto 0);
        MEMOPER        : in   std_logic_vector( 1 downto 0);
        MAPADDR        : out  std_logic_vector(12 downto 0)
      );
end ADDRESSMAPPER;

architecture RTL of ADDRESSMAPPER is
signal CMDADDR : std_logic_vector(11 downto 0);
begin

process(CLK)
 begin
  if CLK'event and CLK='1' then
    if ADDRLAT='1' then
      CMDADDR <= CMDVAL;
    end if;
  end if;
end process;

process(CMDADDR,MEMADDR,MEMOPER)
variable SA          : std_logic_vector(4 downto 0);
variable WCCW        : std_logic_vector(4 downto 0);
variable WCAD        : std_logic_vector(4 downto 0);
variable MC          : std_logic;
variable BCAST       : std_logic;
variable TX          : std_logic;
variable MSEL        : std_logic_vector( 2 downto 0);
begin
 SA     := CMDADDR(9 downto 5);
 WCCW  := CMDADDR(4 downto 0);
 WCAD  := MEMADDR(4 downto 0);
 BCAST := CMDADDR(11);
 TX    := CMDADDR(10);
 if (SA="00000" or SA="11111") then
   MC := '1';
 else
   MC := '0';
 end if;
 MSEL  := MEMOPER & MC;
 case MSEL is
  when "100"   => MAPADDR <= BCAST & TX & SA & '0' & "00000";  -- CW Data Transfer
  when "101"   => MAPADDR <= BCAST & TX & SA & '0' & WCCW;     -- CW Mode Code
  when "000"   => MAPADDR <= BCAST & TX & SA & '1' & WCAD;     -- DW Transfer
  when "001"   => MAPADDR <= BCAST & TX & SA & '1' & WCCW;     -- DW Mode Code
  when others  => MAPADDR <= ( others => '-');
 end case;
end process;
end RTL;
```

Verilog Address Mapping Function

```
module ADDRESSMAPPER (CLK, ADDRLAT, CMDVAL, MEMADDR, MEMOPER, MAPADDR);

  input CLK;
  input ADDRLAT;
  input[11:0] CMDVAL;
  input[10:0] MEMADDR;
  input[1:0] MEMOPER;
  output[12:0] MAPADDR;

  reg[12:0] MAPADDR;
  reg[11:0] CMDADDR;

  always @(posedge CLK)
  begin
   if (ADDRLAT == 1'b1) CMDADDR <= CMDVAL ;
  end

  always @(CMDADDR or MEMADDR or MEMOPER)
  begin
    reg[4:0] SA;
    reg[4:0] WCCW;
    reg[4:0] WCAD;
    reg MC;
    reg BCAST;
    reg TX;
    reg[2:0] MSEL;
    SA    = CMDADDR[9:5];
    WCCW  = CMDADDR[4:0];
    WCAD  = MEMADDR[4:0];
    BCAST = CMDADDR[11];
    TX    = CMDADDR[10];
    if (SA == 5'b00000 | SA == 5'b11111)
      MC = 1'b1;
    else
      MC = 1'b0;
    MSEL = {MEMOPER, MC};
    case (MSEL)
     3'b100  : MAPADDR <= {BCAST, TX, SA, 1'b0, 5'b00000} ; // Command Data Transfer
      3'b101  : MAPADDR <= {BCAST, TX, SA, 1'b0, WCCW} ; // Command Mode Code
      3'b000  : MAPADDR <= {BCAST, TX, SA, 1'b1, WCAD} ; // Data Data Transfer
     3'b001  : MAPADDR <= {BCAST, TX, SA, 1'b1, WCCW} ; // Data Mode Code Transfer
      default : MAPADDR <= {13{1'bx}} ;
    endcase
  end
endmodule
```

Modifying the Backend Interrupt Vector

The default setting of the Core1553BRT creates a six-bit interrupt vector indicating whether or not a good message was received and for a transmitter sub-address. If the INTENBBR (Interrupt Enable Bad Block Received) is high, then interrupts are generated for good and bad 1553B messages. When INTENBBR is low interrupts are only generated for messages that are received or transmitted correctly.

An external interrupt mapping function may be used to add extra information to the interrupt vector (Figure 7-2), such as the received word count and whether the command was broadcast. The CMDVAL output can be used to generate this extra information however it must be externally latched using the INTLAT signal (INTLAT is an active high enable for an external D-type flip flop).

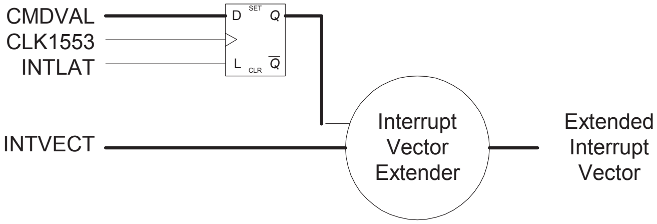

Figure 7-2. External Interrupt Vector Extender Circuit

The interrupt mapping function below creates vector information, indicating broadcast and the received word count value. If an extended interrupt vector is generated then, the need for the system to read the TSW or Command word from the memory is removed; the interrupt vector provides sub-address, word count data and an indication that the message was good. In this case the WRTTSW and WRTCMD inputs can be tied low.

Consider the Verilog Extended Interrupt vector generation example below:

```
module INTVECTEXTENDER (CLK, INTLAT, CMDVAL, INTVECT, MAPVECT);

  input CLK;
  input INTLAT;
  input[11:0] CMDVAL;
  input[6:0] INTVECT;
  output[12:0] MAPVECT;
  reg[12:0] MAPVECT;
  reg[11:0] CMDINT;

  always @(posedge CLK)
  begin
   if (INTLAT == 1'b1)
      CMDINT <= CMDVAL ;
  end

  always @(CMDINT or INTVECT)
  begin
    reg[4:0] SA;
    reg[4:0] WC;
    reg BCAST;
    reg TX;
    reg GBR;
    WC    = CMDINT[4:0];
    BCAST = CMDINT[11];
    GBR   = INTVECT[6];
    TX    = INTVECT[5];
    SA    = INTVECT[4:0];
    MAPVECT <= {GBR, BCAST, TX, SA, WC} ;
  end
```

Or, if you are using VHDL, consider the VHDL Extended Interrupt Generation function example code:

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity INTVECTEXTENDER is
 port ( CLK             : in   std_logic;
        INTLAT          : in   std_logic;
        CMDVAL          : in   std_logic_vector(11 downto 0);
        INTVECT         : in   std_logic_vector( 6 downto 0);
        MAPVECT         : out  std_logic_vector(12 downto 0)
      );
end INTVECTEXTENDER;

architecture RTL of INTVECTEXTENDER is
signal CMDINT : std_logic_vector(11 downto 0);
begin

process(CLK)
 begin
  if CLK'event and CLK='1' then
    if INTLAT='1' then
      CMDINT <= CMDVAL;
    end if;
  end if;
end process;

process(CMDINT,INTVECT)
variable SA          : std_logic_vector(4 downto 0);
variable WC          : std_logic_vector(4 downto 0);
variable BCAST       : std_logic;
variable TX          : std_logic;
variable GBR         : std_logic;
begin
 WC    := CMDINT(4 downto 0);
 BCAST := CMDINT(11);
 GBR   := INTVECT(6);
 TX    := INTVECT(5);
 SA    := INTVECT(4 downto 0);
 MAPVECT <= GBR & BCAST & TX & SA & WC;
end process;

end RTL;
```

Connecting the Backend to Internal FPGA Memory

When implementing the core in ProASIC or Axcelerator devices you can directly connect the core to the ProASIC or Axcelerator memory blocks. Use two memory blocks (1Kx16 each - one for transmit and the other for receive memory) as shown in Figure 7-3.

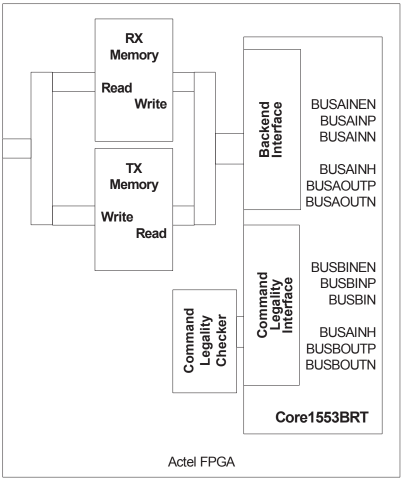

Figure 7-3. Using Internal FPGA RAM Modules

The core must be in Synchronous mode (ASYNCIF inactive (Low)). The MEMGNTn input should be tied active (Low) and the MEMWAITn input inactive (High). This allows the backend to have immediate access to the memory blocks.

Buffer Management

The core implements basic buffer management techniques for the 1553B message protocol; receive and transmit buffers are provided. This approach requires the backend system to empty and fill the buffers promptly. For a broadcast receive command, the core generates an interrupt, indicating that a receive buffer is available as the last data word is written to memory. At the same time, another receive command to the same sub-address could be on the bus, which causes the first word in the memory to be overwritten within 20μs, depending on the backend request grant times. This time could be reduced to less than 5μs if the backend delays the core's access to the memory for the last data word and allows immediate access for the first data word in the following message.

The following implementation (Figure 7-4) implements an extra buffer level in the backend. The core writes the received data to the small 32x16 memory block. At the end of the transfer, the backend RX state machine captures the TSW value, then bursts the complete receive packet into main memory. This system enables the complete message to be copied to main memory, once completely validated. It will stay intact in main memory until the complete following message is received, at least 40μs.

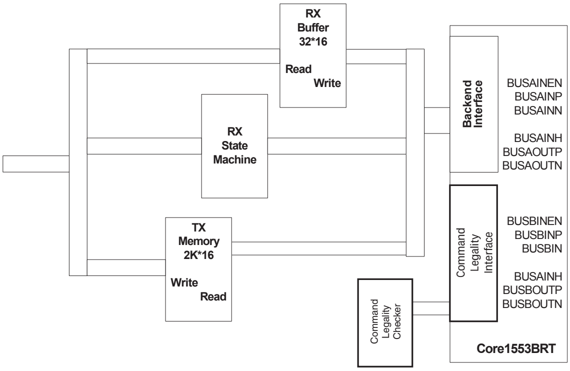

Figure 7-4. Buffer Management Circuit

Buffer management techniques are system dependent and depend on the bus traffic scheduling by the bus controllers. The Core1553BRT implementation is intended to provide a flexible interface that can be adapted to the system requirements.

A

VHDL Testbench Procedure and Function Calls

Both the verification and VHDL user testbenches provided with the core include some low-level support routines that you can use to verify your 1553BRT. These are available in the three packages that can be accessed by adding the following four lines to your VHDL source code:

```
Library Core1553B;
use Core1553B.misc.all;
use Core1553B.textio.all;
use Core1553B.test1553b.all;
```

These routines make use of the following types that are also declared in these packages:

```
subtype INT1BIT    is integer range 0 to 1;
subtype INT5BIT    is integer range 0 to 31;
subtype NIBBLE     is std_logic_vector ( 3 downto 0);
subtype BYTE       is std_logic_vector ( 7 downto 0);
subtype WORD       is std_logic_vector (15 downto 0);
type BYTE_ARRAY    is array ( INTEGER range <>) of BYTE;
type WORD_ARRAY    is array ( INTEGER range <>) of WORD;
type PACKET  is array ( INTEGER range 0 to 31) of WORD;
```

The two record structures used in the encoder decoder module are:

```
type TQMSGREQ is
 record
  RT      : INT5BIT;
  TX      : INT1BIT;
  SA      : INT5BIT;
  MCWC    : INTEGER range 0 to 32;
  RTRT    : BOOLEAN;
  RT2     : INT5BIT;
  SA2     : INT5BIT;
  DATA    : PACKET;
end record;

type TQMSGOUT is
 record
  OKAY      : BOOLEAN;
  COUNT     : INTEGER;
  CW1       : WORD;
  CW2       : WORD;
  SW1       : WORD;
  SW2       : WORD;
  DATA      : PACKET;
end record;
```

The following type conversion routines are provided to convert between integers, std_logic and Boolean types:

```
function  to_slv1bit( x: INT1BIT ) return std_logic_vector;
function  to_sl1bit( x: INT1BIT ) return std_logic;
function  to_slv5bit( x: INT5BIT ) return std_logic_vector;
function  to_int1bit( x: boolean ) return INT1BIT;
function  to_int1bit( x: std_logic ) return INT1BIT;
function  to_int1bit( x: std_logic_vector  ) return INT1BIT;
function  to_int5bit( x: std_logic_vector ) return INT5BIT;
function  to_byte   ( x : INTEGER ) return BYTE;
function  to_word   ( x : INTEGER ) return WORD;
```

The following function call is provided to create data patterns. This returns a data packet whose first value is set to the seed value and the subsequent values are incremented by the delta value. If the delta value is zero then all values are the same. For example:

```
function  initdata( SEED : INTEGER; DELTA : INTEGER) return PACKET;
```

A procedure is provided that compares data packets and displays the error when the compare fails. For example:

```
procedure ComparePacket( STR   : STRING;
    EXP       : Packet;
    GOT       : Packet;
    LEN       : Integer;
    ERRCNT    : inout INTEGER );
```

The first parameter is a text string that is printed when a failure occurs; the second parameter is the expected data and the third parameter the actual data. The fourth parameter indicates how many of the words are to be compared (the packet contains 32 words). Finally, the fifth parameter is a global error counter that is incremented if a compare fails.

A procedure is provided that displays the received message record structure. This procedure requires both the message request and output record structures so that it can format the printed data correctly. For example:

```
procedure print_msgout( QMSG : TQMSGREQ; Q : TQMSGOUT);
```

To support general printing, a printf routine is provided, which supports printing std_logic, boolean, integer, and std_logic_vector types. The syntax is slightly different than the c-language ‘printf’ function. For example:

```
  printf("Hello World Decimal %d Hex %x Hex4 %04x Bits %04b A
string %s",
    fmt(256)&fmt(WORD)&fmt(WORD)&fmt(NIBBLE)&fmt(STRING));
```

prints "Hello World Decimal 256 Hex 100 Hex4 0100 Bits 1010 A string thestring".

B

Product Support

Actel backs its products with various support services including Customer Service, a Customer Applications 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.

Actel U.S. Toll-Free Line

Use the Actel toll-free line to contact Actel for sales information, technical support, requests for literature about Actel and Actel products, Customer Service, investor information, and using the Action Facts service.

The Actel toll-free line is (888) 99-ACTEL.

Customer Service

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

From Northeast and North Central U.S.A., call (408) 522-4480.
From Southeast and Southwest U.S.A., call (408) 522-4480.
From South Central U.S.A., call (408) 522-4434.
From Northwest U.S.A., call (408) 522-4434.
From Canada, call (408) 522-4480.
From Europe, call (408) 522-4252 or +44 (0) 1256 305600.
From Japan, call (408) 522-4743.
From the rest of the world, call (408) 522-4743.
Fax, from anywhere in the world (408) 522-8044.

Customer Applications Center

Actel staffs its Customer Applications Center with highly skilled engineers who can help answer your hardware, software, and design questions. The Applications 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 question(s).

Guru Automated Technical Support

Guru is a web-based automated technical support system accessible through the Actel home page (**http://www.actel.com/guru/**). Guru provides answers to technical questions about Actel products. Many answers include diagrams, illustrations, and links to other resources on the Actel web site. Guru is available 24 hours a day, seven days a week.

Web Site

Actel has a World Wide Web home page where you can browse a variety of technical and nontechnical information. Use a Net browser (Netscape recommended) to access Actel's home page.

The URL is **http://www.actel.com**. You are welcome to share the resources provided on the Internet.

Please visit the Intellectual Property portion of our web site. It lists supported cores, documention, and development boards for use with these cores.

There is also a Technical Documentation area on our website that contains information regarding products, technical services, current manuals, and release notes.

You can visit the Product Support area of the Actel website from your Designer software. Click the Product Support button in your Designer Main Window to access the latest datasheets, application notes, and more.

FTP Site

Actel has an anonymous FTP site located at **ftp://ftp.actel.com**. Here you can obtain library updates, software patches, design files, and data sheets.

Contacting the Customer Applications Center

Highly skilled engineers staff the Customer Applications Center from 7:30 A.M. to 5:00 P.M., Pacific Time, Monday through Friday. Several ways of contacting the Center follow:

Electronic Mail

You can communicate your technical questions to our e-mail address and receive answers back by e-mail, fax, or phone. Also, if you have design problems, you can e-mail your design files to receive assistance. We constantly monitor the e-mail 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 e-mail address is **tech@actel.com**.

Telephone

Our Technical Message 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 7:30 A.M. to 5:00 A.M., Pacific Time, Monday through Friday.

The Customer Applications Center number is (800) 262-1060.

European customers can call +44 (0) 1256 305600.

Worldwide Sales Offices

Headquarters

Actel Corporation
955 East Arques Avenue
Sunnyvale, California 94086
Toll Free: 888.99.ACTEL

Tel: 408.739.1010
Fax: 408.739.1540

US Sales Offices

California

Bay Area
Tel: 408.328.2200
Fax: 408.328.2358

Irvine
Tel: 949.727.0470
Fax: 949.727.0476

Newbury Park
Tel: 805.375.5769
Fax: 805.375.5749

Colorado

Tel: 303.420.4335
Fax: 303.420.4336

Florida

Tel: 407.977.6846
Fax: 407.977.6847

Georgia

Tel: 770.277.4980
Fax: 770.277.5896

Illinois

Tel: 847.259.1501
Fax: 847.259.1575

Massachusetts

Tel: 978.244.3800
Fax: 978.244.3820

Minnesota

Tel: 651.917.9116
Fax: 651.917.9114

New Jersey

Tel: 609.517.0304

North Carolina

Tel: 919.654.4529
Fax: 919.674.0055

Pennsylvania

Tel: 215.830.1458
Fax: 215.706.0680

Texas

Tel: 972.235.8944
Fax: 972.235.965

International Sales Offices

Canada
235 Stafford Rd. West,
Suite 106
Nepean, Ontario K2H 9C1

Tel: 613.726.7575
Fax: 613.726.8666

France
Actel Europe S.A.R.L.
361 Avenue General de Gaulle
92147 Clamart Cedex

Tel: +33 (0)1.40.83.11.00
Fax: +33 (0)1.40.94.11.04

Germany
Lohweg 27
85375 Neufahrn

Tel: +49 (0)8165.9584.0
Fax: +49 (0)8165.9584.1

Italy
Via de Garibaldini, No. 5
20019 Settimo Milanese,
Milano, Italy

Japan
EXOS Ebisu Building 4F
1-24-14 Ebisu Shibuya-ku
Tokyo 150

Tel: +81 (0)3.3445.7671
Fax: +81 (0)3.3445.7668

Korea
30th Floor, ASEM Tower,
159-1 Samsung-dong,
Kangam-ku,
Seoul, Korea

Tel: +82.2.6001.3382
Fax: +82.2.6001.3030

United Kingdom
Maxfli Court,
Riverside Way
Camberley,
Surrey GU15 3YL

Tel: +44 (0)1276.401452
Fax: +44 (0)1276.401490