DVB-ASI Physical Layer Implementation

XAPP509 (v1.0) February 24, 2005

XILINX®

Xilinx is disclosing this Document and Intellectual Property (hereinafter ìthe Designî) to you for use in the development of designs to operate on, or interface with Xilinx FPGAs. Except as stated herein, none of the Design may be copied, reproduced, distributed, republished, downloaded, displayed, posted, or transmitted in any form or by any means including, but not limited to, electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of Xilinx. Any unauthorized use of the Design may violate copyright laws, trademark laws, the laws of privacy and publicity, and communications regulations and statutes.

Xilinx does not assume any liability arising out of the application or use of the Design; nor does Xilinx convey any license under its patents, copyrights, or any rights of others. You are responsible for obtaining any rights you may require for your use or implementation of the Design. Xilinx reserves the right to make changes, at any time, to the Design as deemed desirable in the sole discretion of Xilinx. Xilinx assumes no obligation to correct any errors contained herein or to advise you of any correction if such be made. Xilinx will not assume any liability for the accuracy or correctness of any engineering or technical support or assistance provided to you in connection with the Design.

THE DESIGN IS PROVIDED ìAS IS" WITH ALL FAULTS, AND THE ENTIRE RISK AS TO ITS FUNCTION AND IMPLEMENTATION IS WITH YOU. YOU ACKNOWLEDGE AND AGREE THAT YOU HAVE NOT RELIED ON ANY ORAL OR WRITTEN INFORMATION OR ADVICE, WHETHER GIVEN BY XILINX, OR ITS AGENTS OR EMPLOYEES. XILINX MAKES NO OTHER WARRANTIES, WHETHER EXPRESS, IMPLIED, OR STATUTORY, REGARDING THE DESIGN, INCLUDING ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE, AND NONINFRINGEMENT OF THIRD-PARTY RIGHTS.

IN NO EVENT WILL XILINX BE LIABLE FOR ANY CONSEQUENTIAL, INDIRECT, EXEMPLARY, SPECIAL, OR INCIDENTAL DAMAGES, INCLUDING ANY LOST DATA AND LOST PROFITS, ARISING FROM OR RELATING TO YOUR USE OF THE DESIGN, EVEN IF YOU HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. THE TOTAL CUMULATIVE LIABILITY OF XILINX IN CONNECTION WITH YOUR USE OF THE DESIGN, WHETHER IN CONTRACT OR TORT OR OTHERWISE, WILL IN NO EVENT EXCEED THE AMOUNT OF FEES PAID BY YOU TO XILINX HEREUNDER FOR USE OF THE DESIGN. YOU ACKNOWLEDGE THAT THE FEES, IF ANY, REFLECT THE ALLOCATION OF RISK SET FORTH IN THIS AGREEMENT AND THAT XILINX WOULD NOT MAKE AVAILABLE THE DESIGN TO YOU WITHOUT THESE LIMITATIONS OF LIABILITY.

The Design is not designed or intended for use in the development of on-line control equipment in hazardous environments requiring fail-safe controls, such as in the operation of nuclear facilities, aircraft navigation or communications systems, air traffic control, life support, or weapons systems (ìHigh-Risk Applicationsî). Xilinx specifically disclaims any express or implied warranties of fitness for such High-Risk Applications. You represent that use of the Design in such High-Risk Applications is fully at your risk.

© 2005 Xilinx, Inc. All rights reserved. XILINX, the Xilinx logo, and other designated brands included herein are trademarks of Xilinx, Inc. PowerPC is a trademark of IBM, Inc. All other trademarks are the property of their respective owners.

DVB-ASI Physical Layer Implementation
XAPP509 (v1.0) February 24, 2005

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
02/24/05	1.0	Initial Xilinx release.

Table of Contents

About This Application Note

This application note describes an implementation of the Digital Video Broadcasting - Asynchronous Serial Interface (DVB-ASI) physical layer in a Xilinx FPGA. DVB-ASI transmitter and receiver designs are discussed as separate modules. A pass-through design, connecting the receiver and transmitter, is also discussed.

This document is organized as follows:

- "Introduction," provides a brief overview of the DVB-ASI specification and the Xilinx physical layer implementation.
- "DVB-ASI Receiver," discusses a standalone DVB-ASI receiver design.
- "DVB-ASI Transmitter," discusses a standalone DVB-ASI transmitter design.
- "Pass-Through Mode," discusses the DVB-ASI receiver and transmitter design configured in pass-through mode.

Additional Resources

Additional information and resources are available at http://www.xilinx.com/support/.

Resource	Description/URL
Tutorials	Tutorials covering Xilinx design flows, from design entry to verification and debugging http://www.xilinx.com/support/techsup/tutorials/index.htm
Answer Browser	Database of Xilinx solution records http://www.xilinx.com/xlnx/xil_ans_browser.jsp
Application Notes	Descriptions of device-specific design techniques and approaches http://www.xilinx.com/xlnx/xweb/xil_publications_index.jsp?category=Application+Notes
Data Sheets	Information on Xilinx device-specific characteristics http://www.xilinx.com/xlnx/xweb/xil_publications_index.jsp
Problem Solvers	Interactive tools that allow you to troubleshoot your design issues http://www.xilinx.com/support/troubleshoot/psolvers.htm
Tech Tips	Tips and patch information for the Xilinx design environment http://www.xilinx.com/xlnx/xil_tt_home.jsp

Conventions

This document uses the following conventions. An example illustrates each convention.

Typographical

The following typographical conventions are used in this document:

Convention	Meaning or Use	Example
`Courier font`	Messages, prompts, and program files that the system displays	`speed grade: - 100`
`Courier bold`	Literal commands entered in a syntactical statement	**`ngdbuild`** *`design_name`*
Helvetica bold	Commands selected from a menu	**File →Open**
	Keyboard shortcuts	**Ctrl+C**
Italic font	Variables in a syntax statement for which values must be supplied	**`ngdbuild`** *`design_name`*
	References to other manuals	See the *Development System Reference Guide* for more information.
	Emphasis in text	If a wire is drawn so that it overlaps the pin of a symbol, the two nets are *not* connected.
Square brackets [ ]	An optional entry or parameter. However, in bus specifications, such as **`bus[7:0]`**, they are required.	**`ngdbuild`** `[`*`option_name`*`]` *`design_name`*
Braces { }	A list of items from which you must choose one or more	**`lowpwr`** **`={on|off}`**
Vertical bar |	Separates items in a list of choices	**`lowpwr`** **`={on|off}`**

Online

The following conventions are used in this document:

Convention	Meaning or Use	Example
Blue text	Cross-reference link to a location in this document	See the section "Additional Resources" for details.
Red text	Cross-reference link to another document	See Virtex-4 User Guide.
Blue, underlined text	Hyperlink to a website (URL)	Go to http://www.xilinx.com for the latest speed files.

Introduction

DVB Project

The DVB Project is an industry-led consortium of over 260 broadcasters, manufacturers, network operators, software developers, regulatory bodies, and others in over 35 countries committed to designing global standards for the delivery of digital television and data services.

DVB-ASI is a serial video communications standard defined by the DVB consortium for use in transporting MPEG-2 encoded video streams. The standard is most commonly used to connect cable head-end equipment that transports MPEG-2 streams. Data is transported at a rate of 270 Mb/s.

DVB-ASI Protocol Stack

The DVB-ASI protocol stack is illustrated in Figure 1-1.

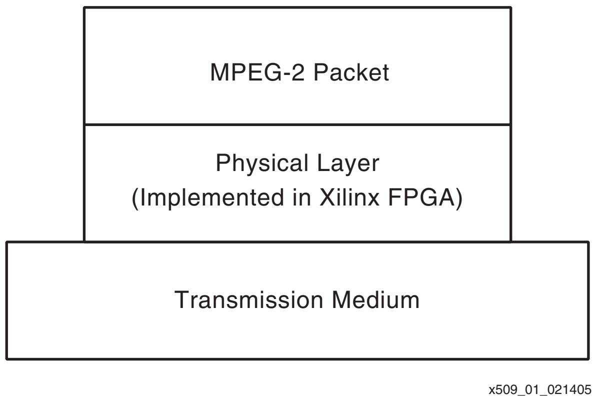


Figure 1-1: **DVB-ASI Protocol Stack**

The section entitled "Reference Design,' in Chapter 4, describes implementation of the physical layer of the protocol. The physical layer employs 8B/10B encoding, which provides DC balancing of the link and a convenient error-checking mechanism. A complete discussion of DVB-ASI specifications is provided in *EN 50083-9: Cabled distribution systems for television, sound and interactive multimedia signals*, available from the DVB Consortium.

DVB-ASI Receiver

Overview

Figure 2-1 shows the DVB-ASI receiver block diagram.

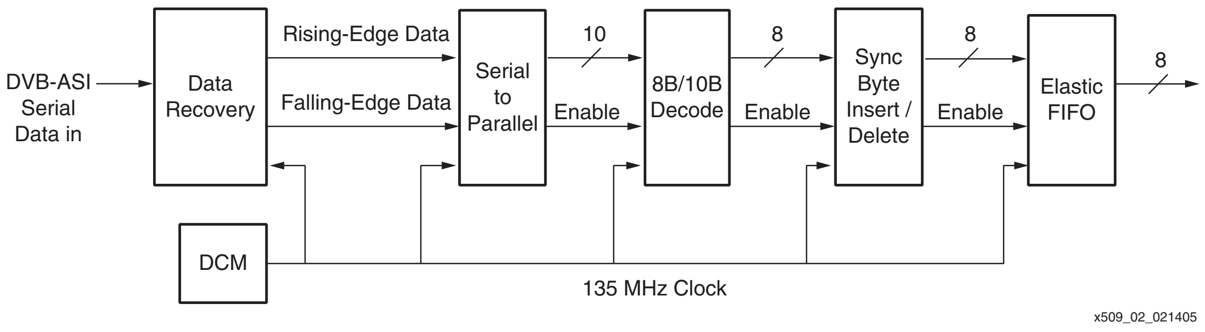


Figure 2-1: **DVB-ASI Receiver Block Diagram**

The DVB-ASI receiver performs two major tasks:

- Frame data in the correct word boundary.
- Decode the 8B10B encoded word.

The receiver is comprised of the following modules:

- Data Recovery Module and Serial-to-Parallel Converter
- 8B/10B Decoder
- Sync Byte Insertion / Deletion Unit
- Elastic FIFO

Table 2-1 shows the DVB-ASI receiver reference design implementation details.

Table 2-1: **DVB-ASI Receiver Implementation Details**

Design Element	Description
Device Used in Implementation	Virtex-II Pro XC2VP4
Board Used in Implementation	Cook Technologies SDV Demo Board
Resource Utilization	200 slices 2 Block RAMs 2 BUFGs 1 DCM
HDL Used	VHDL and Verilog
Synthesis Tool	Synplify 7.6.1
Implementation Tool	Xilinx ISE 6.3

Notes:

1. Synthesis of the reference design has been verified only with Synplify.

Clocking

One digital clock manager (DCM) is used to generate the two required receive clocks: a 135 MHz clock and its inverse. The clocks are routed to global clock buffers for global distribution. Note that only one DCM is required, regardless of the number of DVB-ASI receivers instantiated.

Data Recovery Module

Figure 2-2 shows a block diagram of the data recovery module.

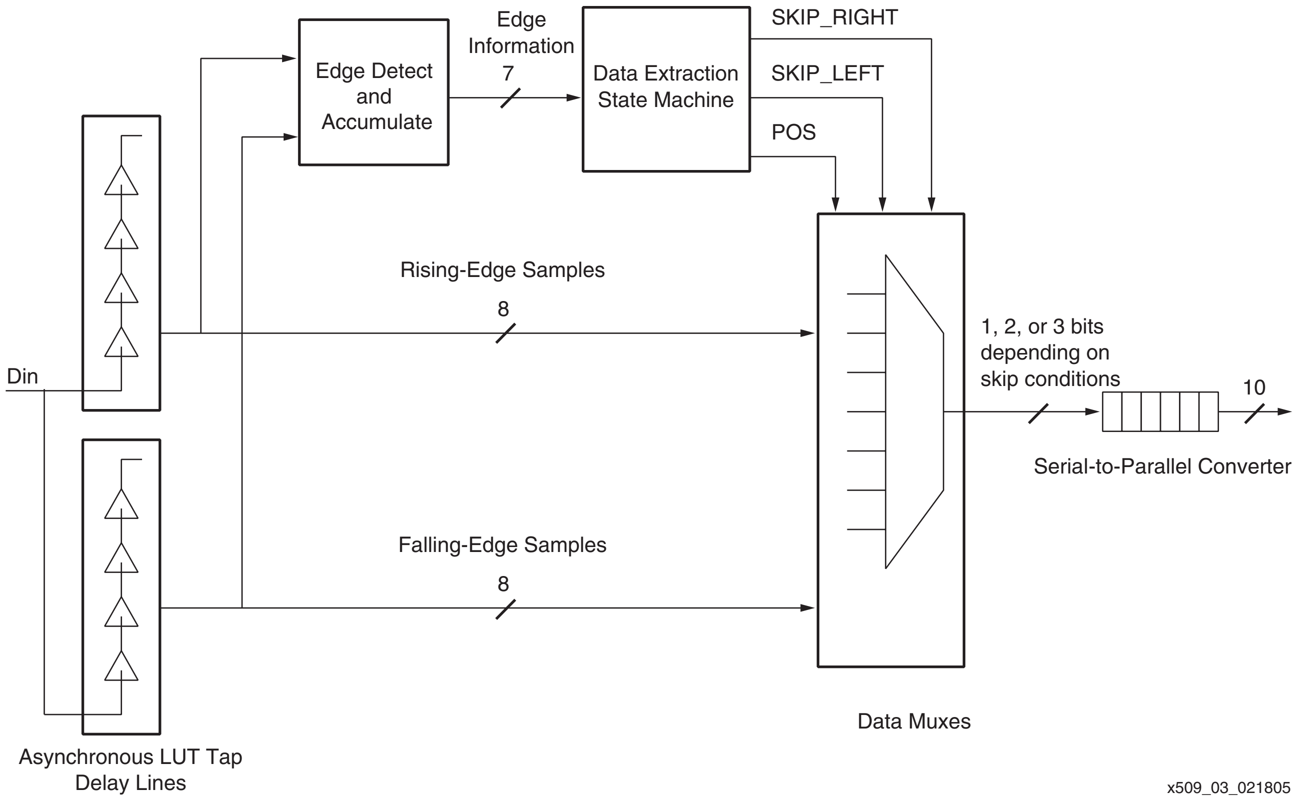

Edge Detect and Accumulate
Edge Information
7
Data Extraction State Machine
SKIP_RIGHT
SKIP_LEFT
POS
Rising-Edge Samples
8
Din
1, 2, or 3 bits depending on skip conditions
10
Serial-to-Parallel Converter
Falling-Edge Samples
8
Data Muxes
Asynchronous LUT Tap Delay Lines
x509_03_021805

Figure 2-2: **Data Recovery Module Block Diagram**

The main purpose of the data recovery module is to recover words from the incoming serial bitstream in the proper word boundary. Module components consist of the following:

- Asynchronous Tap Delay Line
- Data Extraction State Machine
- Serial-to-Parallel Converter

Basic Operation

The DCM generates the system clock, which samples incoming serial data. For the receive clock to capture incoming data correctly, the capture clock must sample data within the data valid window. To avoid sampling outside of the data valid window, the data recovery module continuously monitors its sampling point and determines whether an adjustment is necessary. A data transition edge indicates where the sampling point might be in danger of falling out of the data valid window. Therefore, the data recovery module creates multiple samples of the incoming data each clock period. From the set of samples created, it determines where the transition edges are, and it ultimately positions the sampling point away from the transition edges.

The data recovery module uses a tap delay line constructed from look-up tables (LUTs) that are connected in serial within the FPGA fabric to generate multiple samples of the incoming data stream. Each tap has a combined delay (LUT delay + interconnect delay) of approximately 700 ps. The reference design uses two tap delay lines, each constructed from eight LUTs. Each tap delay line captures every other sample from the incoming bitstream, allowing both tap delay lines to run at half of the incoming data rate. One tap delay line is clocked at 135 MHz (clk135p), while the other is clocked at the inverse (clk135n). The data clocked on the inverse clock domain is eventually brought into the clk135p clock domain.

Edge information is created by pairwise XOR-ing each of the eight samples generated by the tap delay lines. A state machine polls the edge information and decides whether or not to adjust the current sampling point. Since this is a system-synchronous system, when the incoming data rate is slightly different than the system frequency, occasional adjustments to the sampling rate are necessary. If the incoming data rate is faster than the sampling frequency, then an occasional three bits per 135 MHz clock period (instead of the usual two bits per 135 MHz clock period) is sampled to synchronize the data rate. Conversely, if the incoming data rate is slower than the sampling frequency, then an occasional one bit per 135 MHz clock period is sampled to synchronize the data rate.

Implementation Instructions

Proper operation of the data recovery module depends critically on the placement and routing of key routes. The relative location and routing constraints are predetermined and are built into the reference design. The only action the user must take is to set the proper parameters. Placement of the data recovery module is dependent on placement of the data IOBs. Two sets of placement and routing constraints are used for two placement scenarios:

- Data IOBs are placed in top or bottom I/O banks (Banks 0, 1, 4, and 5).
- Data IOBs are placed in right or left I/O banks (Banks 2, 3, 6, and 7).

All placement constraint parameters are found in the `des.vhd` HDL file. Parameters that determine the placement of the data recovery module are the SIDE parameter and the RLOC_ORIG_CONST parameter. The SIDE parameter tells the implementation tool whether to use the constraints for a top-bottom implementation (SIDE = 1) or a left-right implementation (SIDE = 0). The RLOC_ORIG_CONST parameter specifies the RLOC_ORIGIN attribute of the relative location macros that constitute the data recovery module.

To ensure optimum operation, the RLOC_ORIGIN parameter must be located as close to the data IOBs as possible. If possible, for right-left implementations the RLOC_ORIGIN parameter must be in the same CLB row as, or one row above or below, the IOB location. Similarly, for top-bottom implementations the RLOC_ORIGIN parameter must be in the same CLB column as, or one column to the right or left of, the IOB location.

In the case where data IOBs are placed in the top or bottom I/O banks, set the SIDE parameter to 1, and set RLOC_ORIG_CONST to the appropriate value. Figure 2-3 shows the relative location macro configuration for a top-bottom implementation.

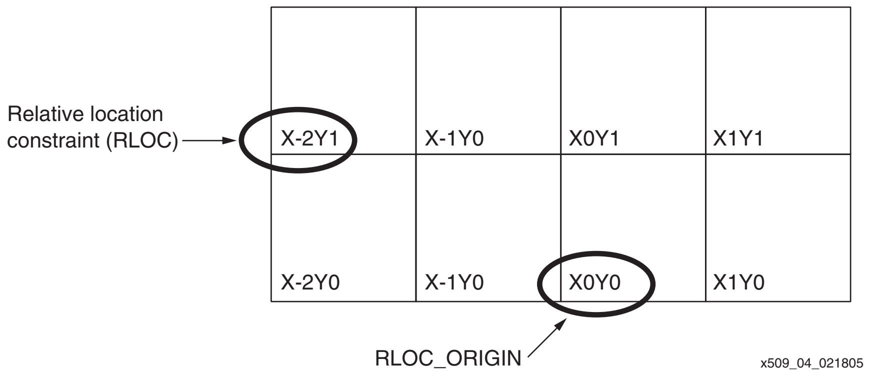


Figure 2-3: **Relative Location of Asynch. Tap Delay Line: Top-Bottom Implementation**

In the case where data IOBs are placed in the right or left I/O banks, set the SIDE parameter to 0, and set RLOC_ORIG_CONST to the appropriate value. Figure 2-4 shows the relative location macro configuration for a left-right implementation.

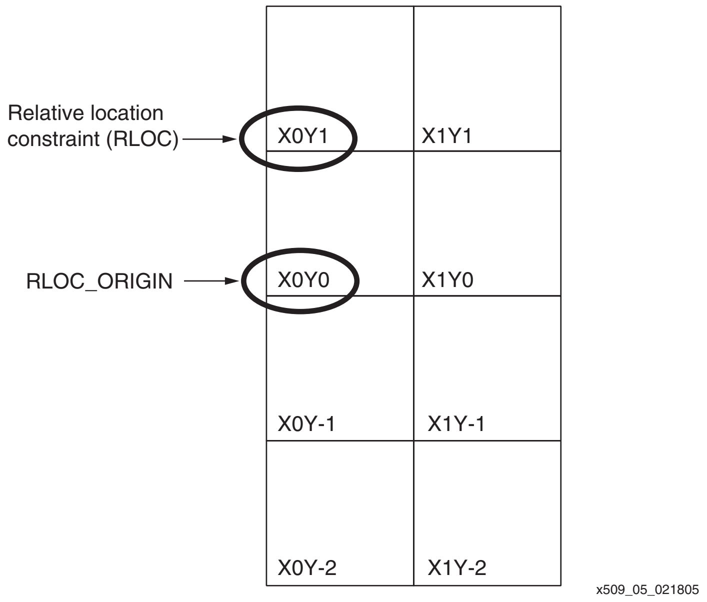
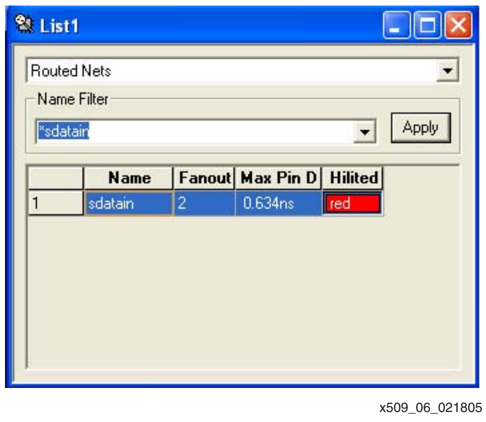


Figure 2-4: **Relative Location of Asynch. Tap Delay Line: Left-Right Implementation**

Checking the routed design in FPGA Editor after place-and-route is complete helps ensure a balanced route from data IOB to the input of the first LUTs on each tap delay line (rising and falling). The difference between both routes must not be more than 500 ps. This can be achieved as follows:

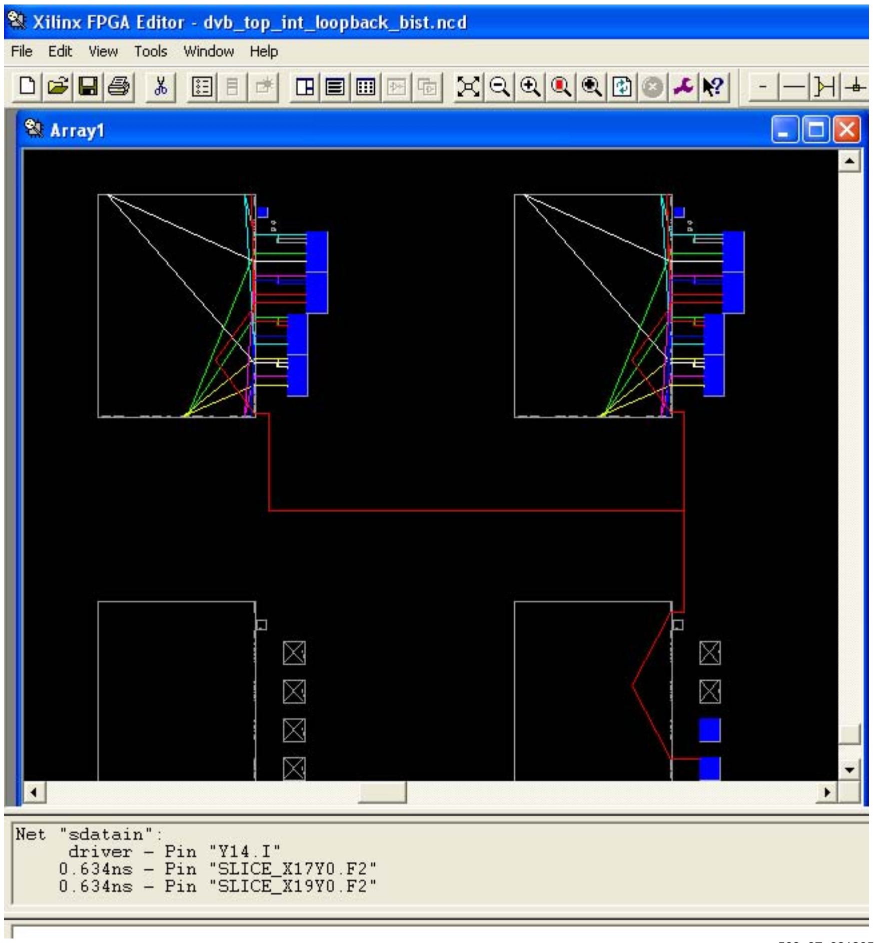
1. Open the routed design in FPGA Editor. The net from IOB to the tap delay line is named `sdatain`.
2. Search for the `sdatain` net in the List window (see Figure 2-5) and highlight it.



	Name	Fanout	Max Pin D	Hilited
1	sdatain	2	0.634ns	red

Figure 2-5: **Search for sdatain Net in List Window**

3. Verify that this is indeed the net that connects the data IOB to the first LUT of the tap delay line. See Figure 2-6.

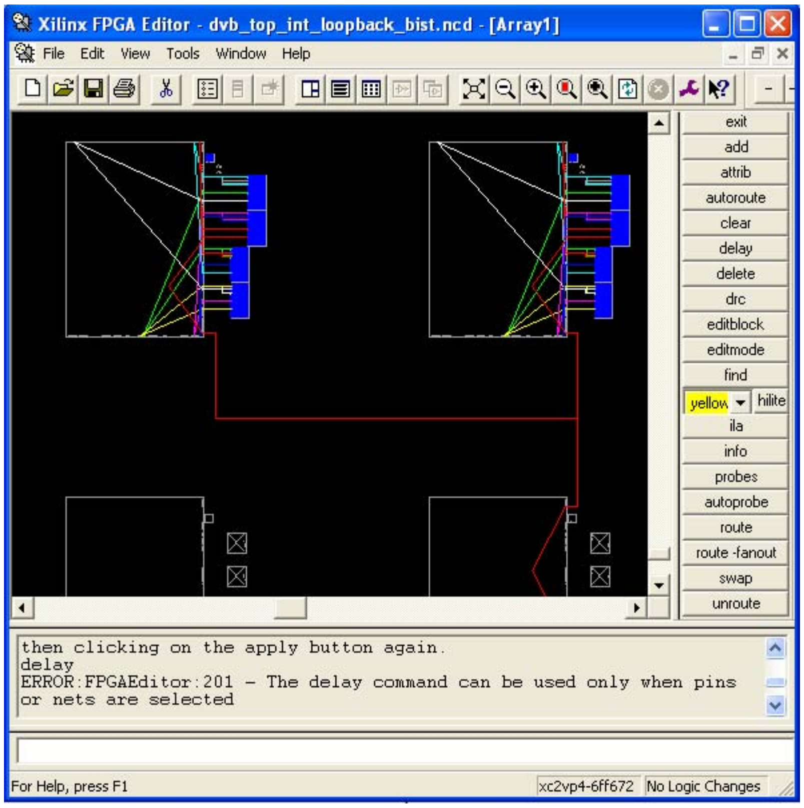

Xilinx FPGA Editor - dvb_top_int_loopback_bist.ncd
File Edit View Tools Window Help
Array1

```
Net "sdatain":
     driver - Pin "Y14.I"
    0.634ns - Pin "SLICE_X17Y0.F2"
    0.634ns - Pin "SLICE_X19Y0.F2"
```

x509_07_021805

Figure 2-6: **Verify IOB to Tap Delay Line Net**

4. With the `sdatain` net highlighted, click on the **Delay** button. See Figure 2-7.



```
then clicking on the apply button again.
delay
ERROR:FPGAEditor:201 - The delay command can be used only when pins
or nets are selected
```

Figure 2-7: **Delay Button**

5. Verify that the delay of the route between the data IOB and the first LUT of both tap delay lines does not differ by more than 500 ps. This timing must be met. If it is not met, change the RLOC_ORIG_CONST parameter until timing is met.

The other parameter the user must set is the SYNC_MODE parameter, also found in the `des.vhd` source file. The SYNC_MODE parameter indicates whether a single sync byte is used to frame the initial data or whether two sync bytes within a 5-byte window are used, as per DVB-ASI specifications. Setting SYNC_MODE to 0 indicates that the receiver frames (locks) the first sync byte it detects. Setting SYNC_MODE to 1 indicates that the receiver frames only when two sync bytes are detected on the same byte boundary within a 5-byte window.

Table 2-2 summarizes user parameters in the design.

Table 2-2: **Table of User Parameters**

PARAMETER	TYPE	DESCRIPTION
RLOC_ORIG_CONST	String	Sets location (RLOC_ORIGIN) of the data recovery module. (1)
SIDE	Bit	Indicates side of the FPGA on which the data recovery module is implemented: 0: Indicates left-right implementation (Data IOBs located in Banks 2, 3, 6, or 7) 1: Indicates top-bottom implementation (Data IOBs located in Banks 0, 1, 4, or 5)
SYNC_MODE	Bit	Indicates whether a single sync byte is used to frame the initial data or whether two sync bytes within a 5-byte window are used: 0: Indicates the receiver frames (locks) the first sync byte it detects. 1: Indicates the receiver frames only when two sync bytes are detected on the same byte boundary within a 5-byte window.

Notes:

1. The location of the data recovery module should be as close to data input pins as possible.

8B/10B Decoder

As mentioned in Chapter 1, "Introduction," the ASI packets are 8B/10B encoded. The 8B/10B code is DC-balanced allowing for cable equalization. Furthermore, 8B/10B transmissions have a limited run length: no more than five consecutive ones or zeroes.

The 8B/10B set of codes provides special comma (K) characters that are useful as packet delimiters. These 10-bit characters are guaranteed not to occur with any input combination. The K28.5 comma character is used as a packet delimiter in the DVB-ASI specifications.

The 8B/10B encoding scheme also provides a convenient error detection scheme, based on the concept of disparity. Disparity is defined as the difference between ones and zeroes in a word. Positive disparity refers to an excess of ones over zeroes. Negative disparity refers to an excess of zeroes over ones. During normal operation, a running disparity is stored, which serves as a disparity record for the aggregate of previously encoded symbols. The disparity of each decoded symbol is added to the running disparity.

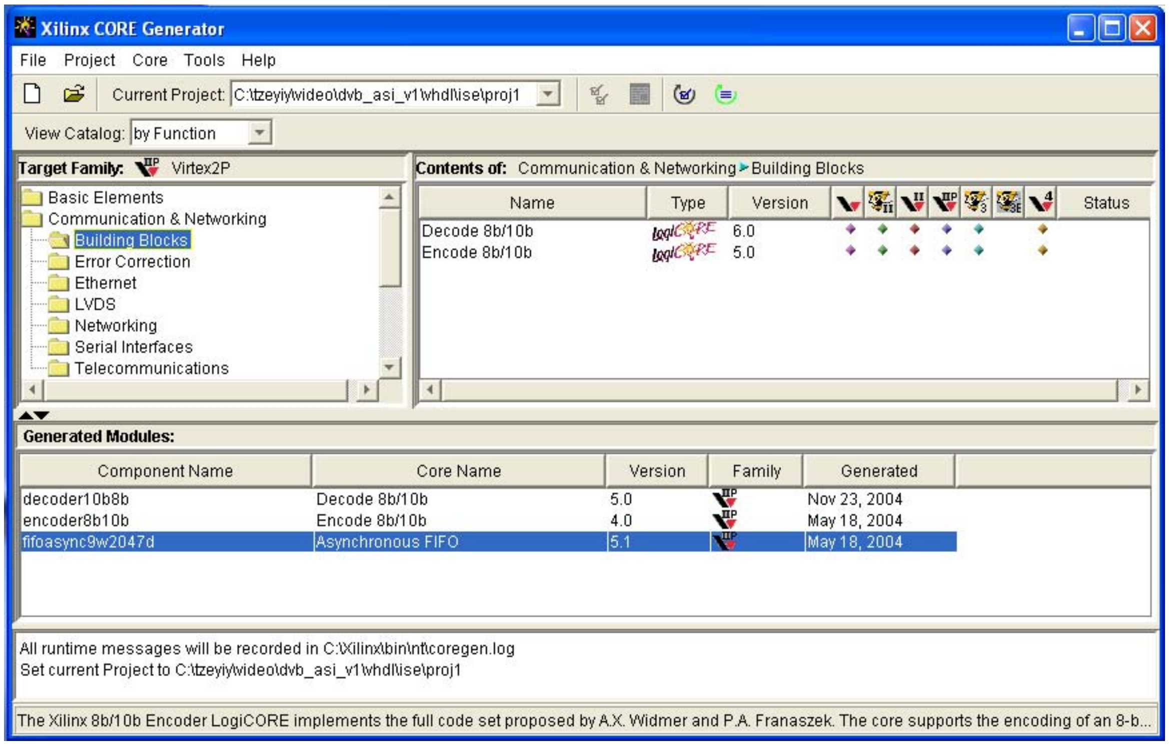
Error detection in the decoder is achieved in two ways: code error and disparity error. A code error occurs when a 10-bit encoded symbol does not match any symbol in the code set. A disparity error occurs when the running disparity does not match a certain value. Using both of these error detection methods enables a very robust error detection scheme. In the reference design, code error and disparity error flags are OR-ed together to set the error_condition flag.

The 8B/10B encoding scheme provides all of these benefits with a very low (only 25%) overhead. The DVB-ASI specifications describe the 8B/10B encoding scheme in greater detail.

Core Generator 8B/10B Decoder Instantiation Instructions

Xilinx provides the 8B/10B decoder core free of charge. The core is available from the Core Generator tool. Instantiating the core in a design is achieved as follows:

1. When the Core Generator tool opens, it prompts the user to select a project to which all Core Generator files will be written. Select the ISE project in which you are working.
2. In Core Generator, navigate to the **Communications & Networking:Building Blocks** folder, where 8B/10B encoder and decoder cores are stored. (See Figure 2-8.)


Xilinx CORE Generator
File Project Core Tools Help
Current Project: C:\tzeyiy\video\dvb_asi_v1\vhdl\ise\proj1
View Catalog: by Function
Target Family: Virtex2P
Basic Elements
Communication & Networking
Building Blocks
Error Correction
Ethernet
LVDS
Networking
Serial Interfaces
Telecommunications
Contents of: Communication & Networking ➤ Building Blocks
Generated Modules:

Name	Type	Version	Status
Decode 8b/10b	LogiCORE	6.0	
Encode 8b/10b	LogiCORE	5.0	

Component Name	Core Name	Version	Family	Generated
decoder10b8b	Decode 8b/10b	5.0		Nov 23, 2004
encoder8b10b	Encode 8b/10b	4.0		May 18, 2004
fifoasync9w2047d	Asynchronous FIFO	5.1		May 18, 2004

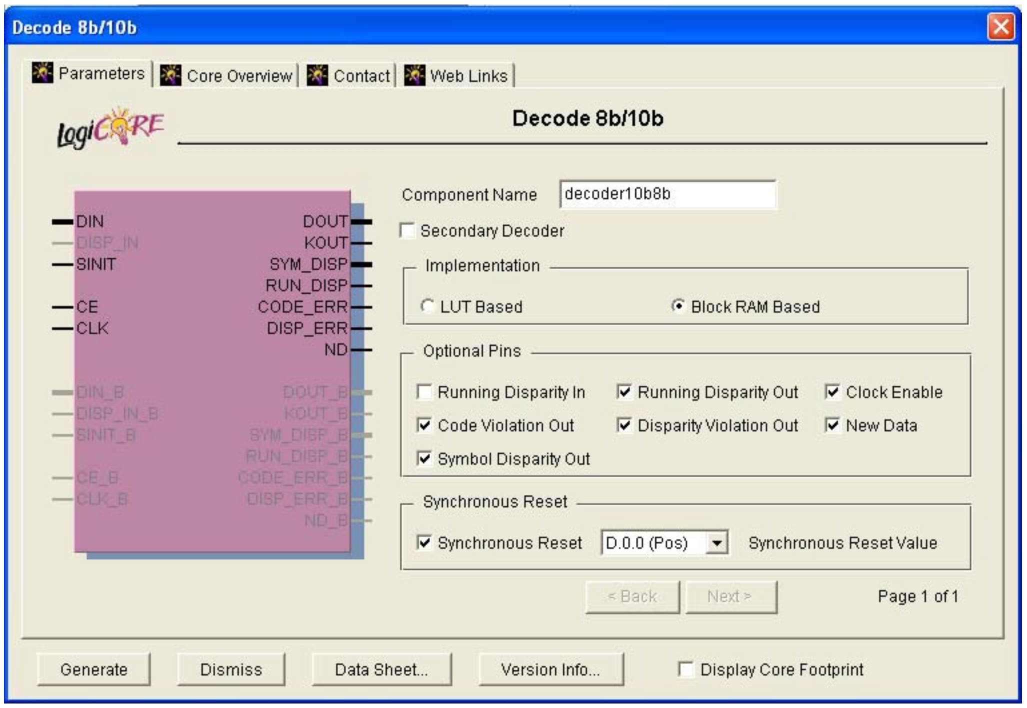
```
All runtime messages will be recorded in C:\Xilinx\bin\nt\coregen.log
Set current Project to C:\tzeyiy\video\dvb_asi_v1\vhdl\ise\proj1
```

The Xilinx 8b/10b Encoder LogiCORE implements the full code set proposed by A.X. Widmer and P.A. Franaszek. The core supports the encoding of an 8-b...

x509_09_021805

Figure 2-8: **Generating 8B/10B Encoder and Decoder from Core Generator**

3. Doubleclick the `Decode 8B/10B` core. A window similar to Figure 2-9 displays. Select parameters as shown, and click **Generate**. The 8B/10B Decoder core is generated.



Component Name: decoder10b8b

☐ Secondary Decoder

Implementation: ○ LUT Based [✓ ● Block RAM Based]

Optional Pins:
☐ Running Disparity In
☑ Running Disparity Out
☑ Clock Enable
☑ Code Violation Out
☑ Disparity Violation Out
☑ New Data
☑ Symbol Disparity Out

Synchronous Reset:
☑ Synchronous Reset
Synchronous Reset Value: D.0.0 (Pos)

☐ Display Core Footprint

x509_10_021805

Figure 2-9: **Generating 8B/10B Decoder from Core Generator**

Elastic FIFO

An elastic FIFO sits in the receive data chain to provide rate-matching between the incoming data rate and the system frequency. The FIFO state machine monitors the FIFO level and, when necessary, adjusts the incoming data stream to keep the FIFO from overflowing or underflowing. Figure 2-10 shows the state machine logic.

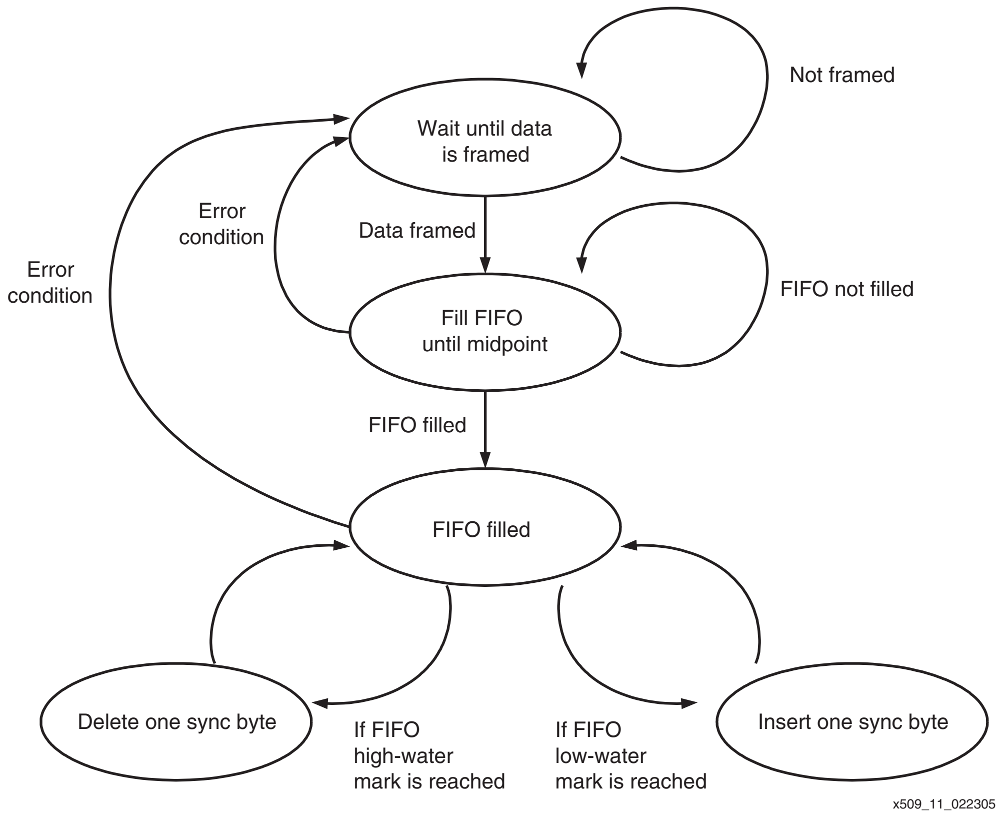


Figure 2-10: **FIFO Control State Machine**

When initialized, the FIFO state machine fills the FIFO to the halfway mark. From then on, the state machine monitors both high-water and low-water mark levels (see Figure 2-11). If the FIFO reaches the high-water mark level (indicating that the incoming data rate is faster than the system / sampling frequency), the state machine deletes one comma character. If the FIFO reaches the low-water mark level (indicating that the incoming data rate is slower than the system / sampling frequency), the state machine adds one comma character.

The FIFO state machine also stuffs the FIFO with comma characters whenever the system loses lock (for example, when a cable is unplugged or a link encounters errors.)

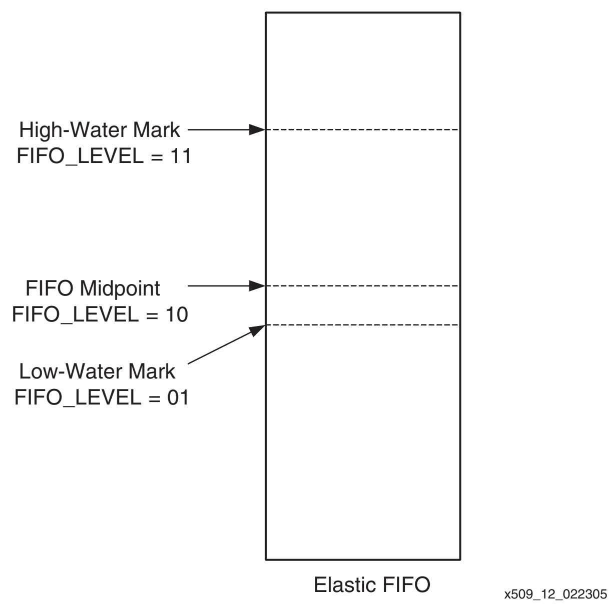
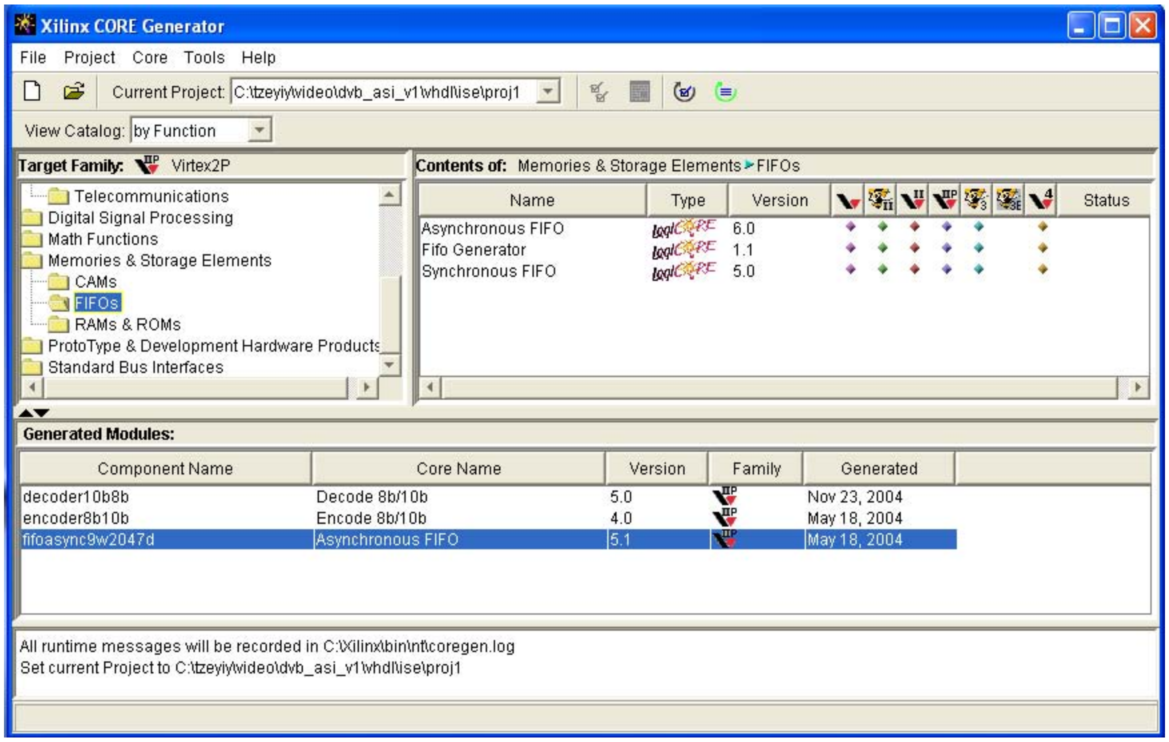


Figure 2-11: **FIFO Levels**

Core Generator FIFO Instantiation Instructions

The FIFO used in the reference design is a 2047 x 9 asynchronous FIFO. Xilinx provides the Asynchronous FIFO core free of charge. The core is available from the Core Generator tool. Instantiating the core in a design is achieved as follows:

1. When the Core Generator tool opens, it prompts the user to select a project to which all Core Generator files will be written. Select the ISE project in which you are working.
2. In Core Generator, navigate to the **Memories and Storage Elements:FIFOs** folder, where FIFO cores are stored. (See Figure 2-12.)


Xilinx CORE Generator
File Project Core Tools Help
Current Project: C:\tzeyiy\video\dvb_asi_v1\vhdl\ise\proj1
View Catalog: by Function
Target Family: Virtex2P
Telecommunications
Digital Signal Processing
Math Functions
Memories & Storage Elements
CAMs
FIFOs
RAMs & ROMs
ProtoType & Development Hardware Products
Standard Bus Interfaces
Contents of: Memories & Storage Elements ▸ FIFOs
Generated Modules:

Name	Type	Version
Asynchronous FIFO	LogiCORE	6.0
Fifo Generator	LogiCORE	1.1
Synchronous FIFO	LogiCORE	5.0

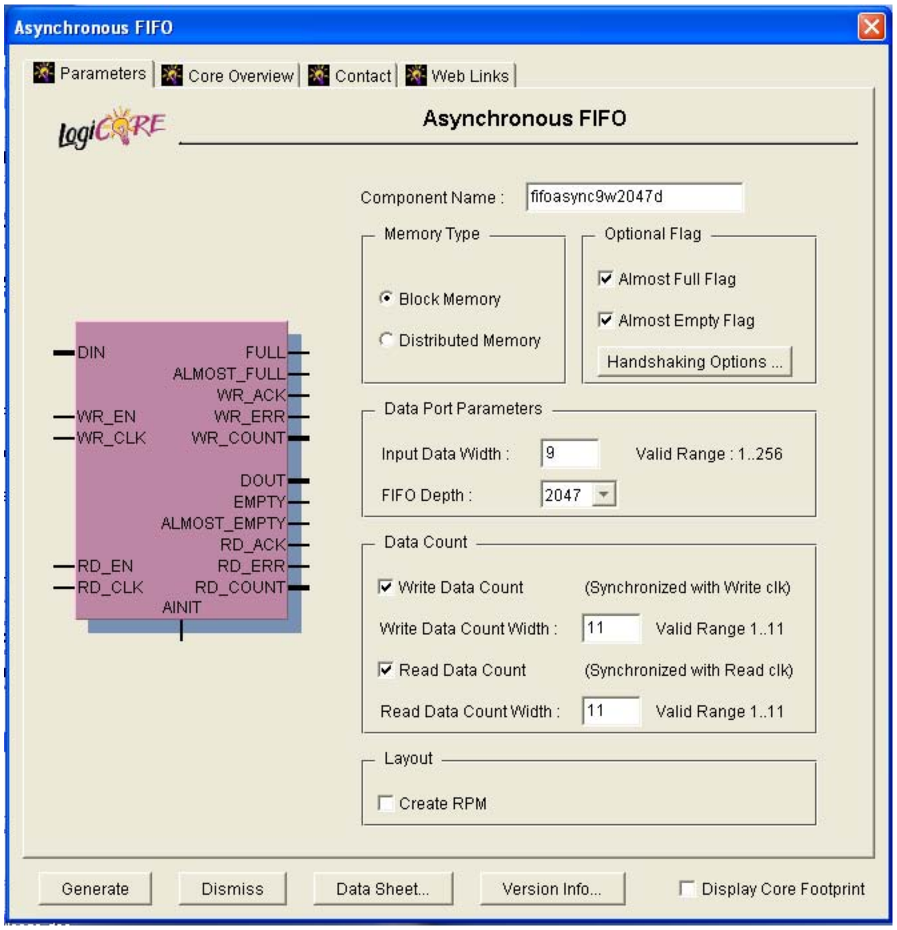
Component Name	Core Name	Version	Family	Generated
decoder10b8b	Decode 8b/10b	5.0		Nov 23, 2004
encoder8b10b	Encode 8b/10b	4.0		May 18, 2004
fifoasync9w2047d	Asynchronous FIFO	5.1		May 18, 2004

```
All runtime messages will be recorded in C:\Xilinx\bin\nt\coregen.log
Set current Project to C:\tzeyiy\video\dvb_asi_v1\vhdl\ise\proj1
```

x509_13_021805

Figure 2-12: **Generating FIFO from Core Generator**

3. Doubleclick the `Asynchronous FIFO` core. A window similar to Figure 2-13 displays.
4. Select parameters as shown and click on the **Handshaking Options...** button.
5. Select parameters and click **Generate**. The Asynchronous FIFO core is generated.



Asynchronous FIFO

Component Name : fifoasync9w2047d

Memory Type: [✓ Block Memory] Distributed Memory

Optional Flag: [✓ Almost Full Flag] [✓ Almost Empty Flag]

Data Port Parameters

Input Data Width : 9 Valid Range : 1..256

FIFO Depth : 2047

Data Count

[✓ Write Data Count] (Synchronized with Write clk)

Write Data Count Width : 11 Valid Range 1..11

[✓ Read Data Count] (Synchronized with Read clk)

Read Data Count Width : 11 Valid Range 1..11

Layout

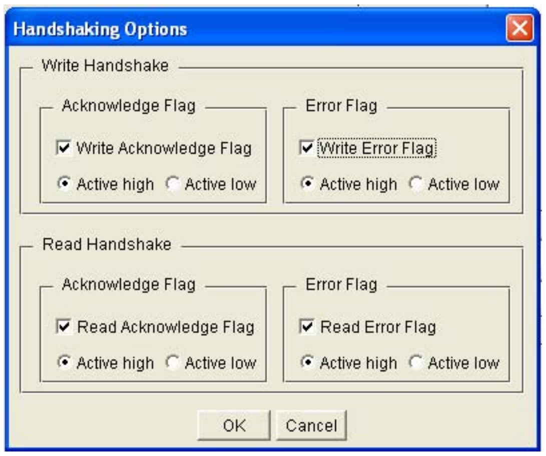
Create RPM

Display Core Footprint

x509_14_021805

Figure 2-13: **Generating FIFO from Core Generator**

Figure 2-14 shows handshaking options.



Write Handshake

Acknowledge Flag: [✓ Write Acknowledge Flag] [✓ Active high] Active low

Error Flag: [✓ Write Error Flag] [✓ Active high] Active low

Read Handshake

Acknowledge Flag: [✓ Read Acknowledge Flag] [✓ Active high] Active low

Error Flag: [✓ Read Error Flag] [✓ Active high] Active low

x509_15_021805

Figure 2-14: **Handshaking Options**

Chapter 3

DVB-ASI Transmitter

Overview

Figure 3-1 shows a block diagram for the DVB-ASI transmitter.

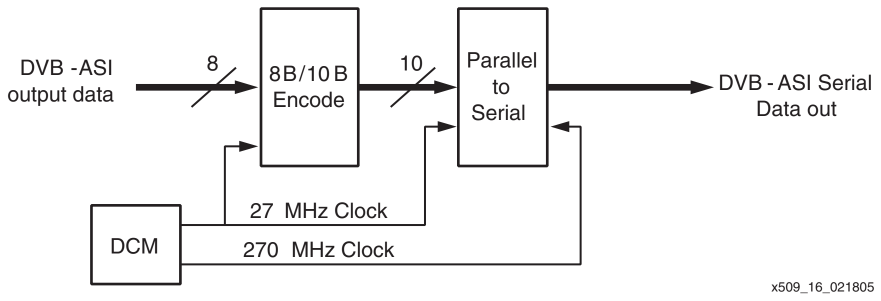


Figure 3-1: **DVB-ASI Transmitter Block Diagram**

The main purpose of the DVB-ASI transmitter is to encode transmit words in 8B/10B format and serialize the data for transmission.

The transmitter is comprised of the following modules:

- 8B/10B Encoder
- Parallel-to-Serial Converter

Table 3-1 shows the DVB-ASI transmitter reference design implementation details.

Table 3-1: **DVB-ASI Transmitter Implementation Details**

Design Element	Description
Device Used in Implementation	Virtex-II Pro XC2VP4
Board Implemented	Cook Technologies SDV Demo Board
Resource Utilization	50 slices 1 Block RAM 2 BUFGs 1 DCM
HDL Used	VHDL and Verilog
Synthesis Tool	XST and Synplify 7.6.1
Implementation Tool	Xilinx ISE 6.3

Clocking

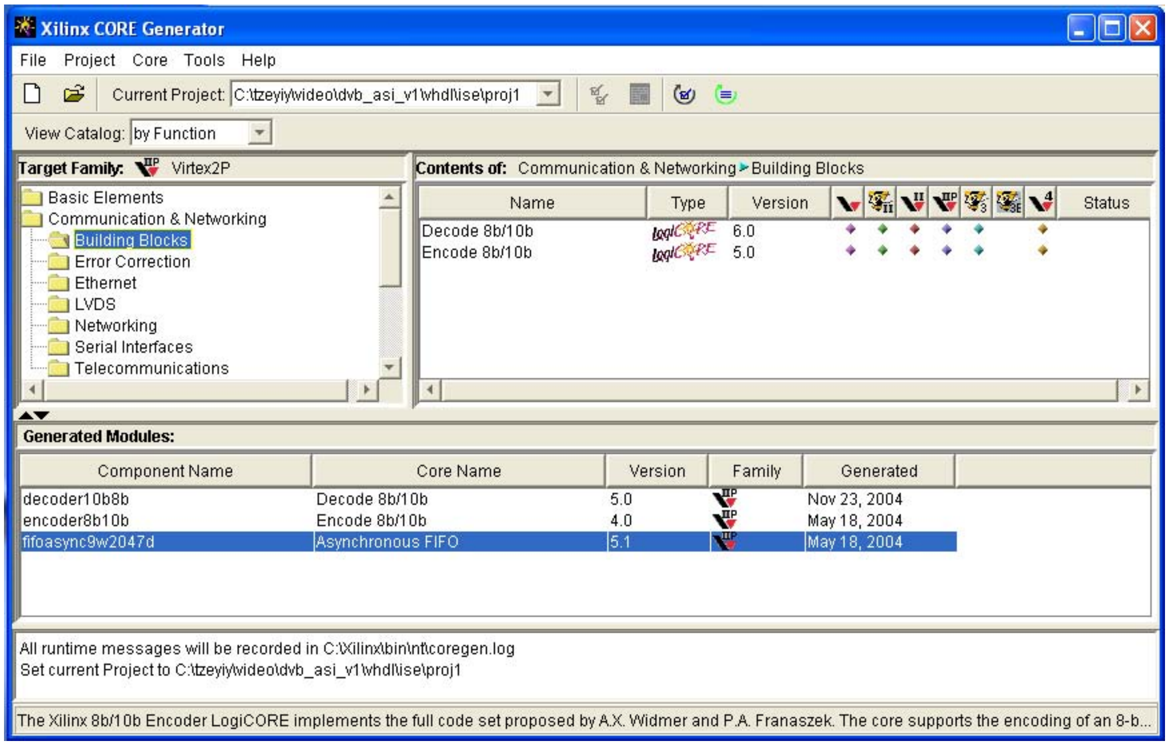
One digital clock manager (DCM) is used to generate the two transmit clocks required: a 27 MHz word rate clock (which is also the system clock) and a 270 MHz serial transmit clock. The clocks are routed to global clock buffers for global distribution. Note that only one DCM is required, regardless of the number of DVB-ASI transmitters instantiated.

8B/10B Encoder

Core Generator 8B/10B Encoder Instantiation Instructions

Xilinx provides the 8B/10B encoder core free of charge. The core is available from the Core Generator tool. Instantiating the core in a design is achieved as follows:

1. When the Core Generator tool opens, it prompts the user to select a project to which all Core Generator files will be written. Select the ISE project in which you are working.
2. In Core Generator, navigate to the **Communications & Networking:Building Blocks** folder, where 8B/10B encoder and decoder cores are stored. (See Figure 3-2.)



Current Project: C:\tzeyiy\video\dvb_asi_v1\vhdl\ise\proj1

View Catalog: by Function

Target Family: Virtex2P

- Basic Elements
- Communication & Networking
  - Building Blocks
  - Error Correction
  - Ethernet
  - LVDS
  - Networking
  - Serial Interfaces
  - Telecommunications

Contents of: Communication & Networking ▸ Building Blocks

Name	Type	Version	Status
Decode 8b/10b	LogiCORE	6.0	
Encode 8b/10b	LogiCORE	5.0	

Generated Modules:

Component Name	Core Name	Version	Family	Generated
decoder10b8b	Decode 8b/10b	5.0		Nov 23, 2004
encoder8b10b	Encode 8b/10b	4.0		May 18, 2004
fifoasync9w2047d	Asynchronous FIFO	5.1		May 18, 2004

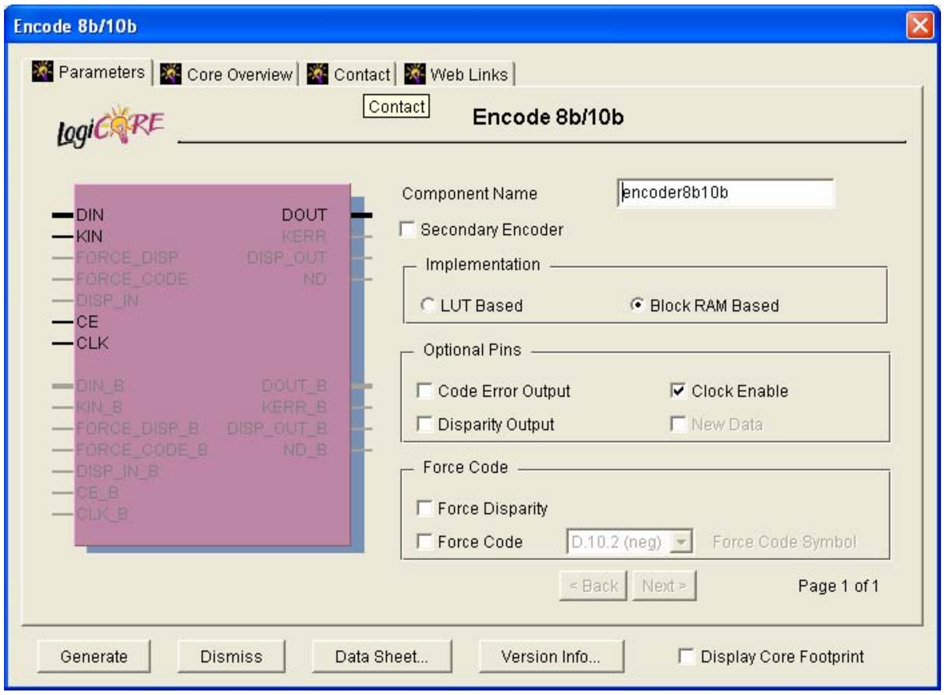
```
All runtime messages will be recorded in C:\Xilinx\bin\nt\coregen.log
Set current Project to C:\tzeyiy\video\dvb_asi_v1\vhdl\ise\proj1
```

The Xilinx 8b/10b Encoder LogiCORE implements the full code set proposed by A.X. Widmer and P.A. Franaszek. The core supports the encoding of an 8-b...

x509_09_021805

Figure 3-2: **Generating 8B/10B Encoder from Core Generator**

3. Doubleclick the `Encode 8B/10B` core. A window similar to Figure 3-3 displays. Select parameters as shown, and click **Generate**. The 8B/10B Encoder core is generated.



Encode 8b/10b

Component Name: encoder8b10b

☐ Secondary Encoder

Implementation: ○ LUT Based [✓ ● Block RAM Based]

Optional Pins: ☐ Code Error Output [✓ Clock Enable] ☐ Disparity Output ☐ New Data

Force Code: ☐ Force Disparity ☐ Force Code D.10.2 (neg) Force Code Symbol

☐ Display Core Footprint

Figure 3-3: **Generating 8B/10B Encoder from Core Generator**

Chapter 4

Pass-Through Mode

Overview

The DVB-ASI transmitter and receiver can be connected together in pass-through mode. Figure 4-1 illustrates the DVB-ASI pass-through design.

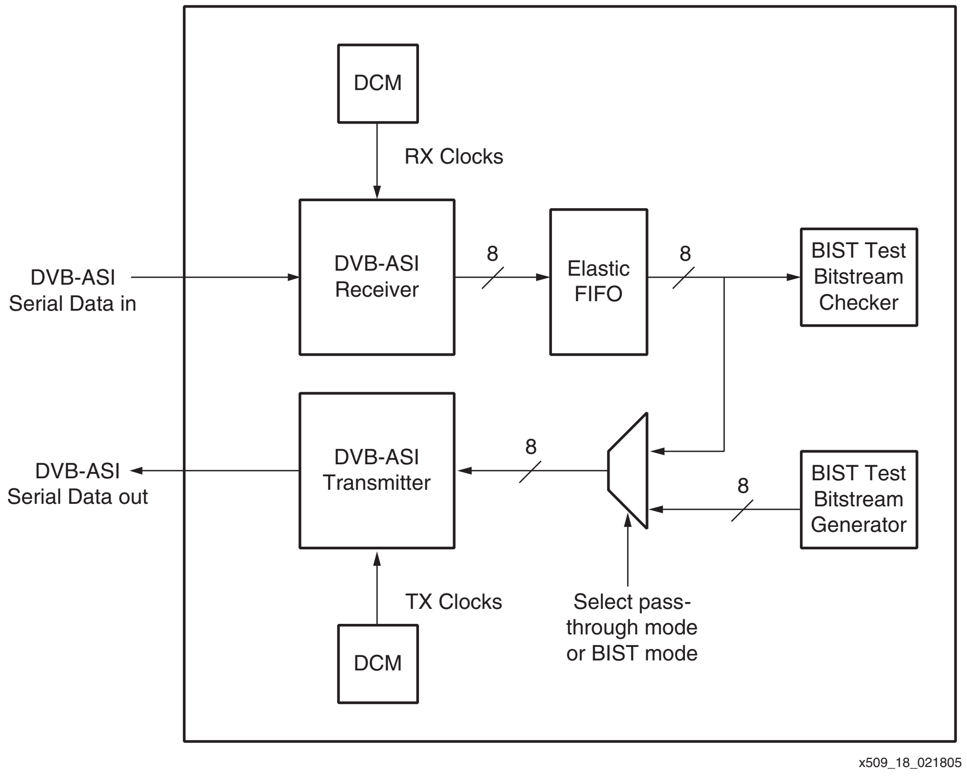


Figure 4-1: **DVB-ASI Block Diagram**

The design is comprised of the following modules:

- LVDS (low-voltage differential signaling) I/Os
- DVB-ASI Receiver
- DVB-ASI Transmitter
- Elastic FIFO
- BIST (Built-In Self Test) Test Bitstream Checker
- BIST (Built-In Self Test) Test Bitstream Generator
- Receive DCM (Digital Clock Manager)
- Transmitter DCM

Modes of Operation

Two modes of operation are available:

- BIST (Built-In Self Test) mode
- Pass-through mode

In BIST (Built-In Self Test) mode, the receiver and transmitter are separate elements and both modules can be run simultaneously. A BIST test bitstream generator and checker provides BIST capabilities.

When configured in pass-through mode, the receiver and transmitter are linked via an elastic FIFO. The FIFO provides rate-matching capabilities between the receiver and transmitter.

Table 4-1 shows the DVB-ASI transmitter reference design implementation details.

Table 4-1: **DVB-ASI Pass-Through Implementation Details**

Design Element	Description
Device Used in Implementation	Virtex-II Pro XC2VP4
Board Implemented	Cook Technologies SDV Demo Board
Resource Utilization	RX: • 200 slices • 2 Block RAMs • 2 BUFGs • 1 DCM TX: • 50 slices • 1 Block RAM • 2 BUFGs • 1 DCM
HDL Used	VHDL and Verilog
Synthesis Tool	Synplify 7.6.1 (1)
Implementation Tool	Xilinx ISE 6.3

Notes:

1. Synthesis of the reference design has been verified only with Synplify.

BIST Test Bitstream Generator and Checker

The BIST test bitstream generator outputs an MPEG-2 compliant packet with a known bitstream that can be checked by the pass-through receiver. Figure 4-2 shows the packet description.

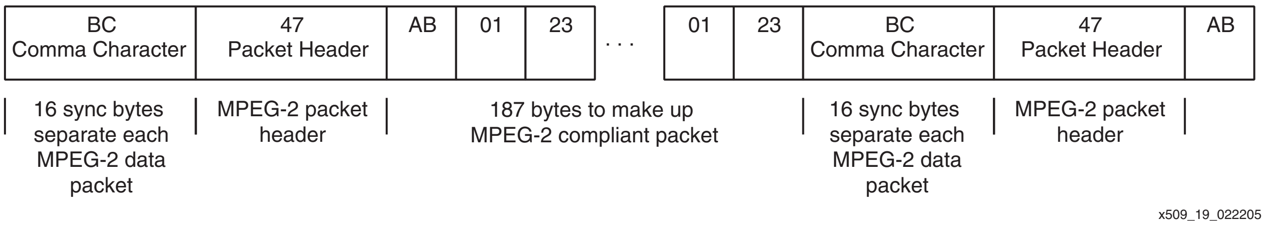

BC
Comma Character
47
Packet Header
AB
01
23
. . .
01
23
BC
Comma Character
47
Packet Header
AB
16 sync bytes separate each MPEG-2 data packet
MPEG-2 packet header
187 bytes to make up MPEG-2 compliant packet
16 sync bytes separate each MPEG-2 data packet
MPEG-2 packet header
x509_19_022205

Figure 4-2: **Test Packet Description**

BIST Parameter Setting

The DVB-ASI specification defines 2 modes of transport:

- Contiguous byte mode
- Interleave byte mode

When data is transmitted in contiguous byte mode, the entire 188-byte MPEG-2 data packet is transmitted as a block separated by sync bytes, as shown in Figure 4-3.

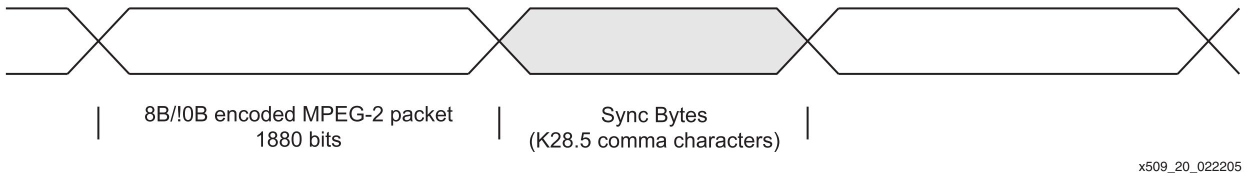


Figure 4-3: **Contiguous Byte Mode**

When data is transmitted in interleave byte mode, each byte in the MPEG-2 packet is alternated with sync bytes, as shown in Figure 4-4.

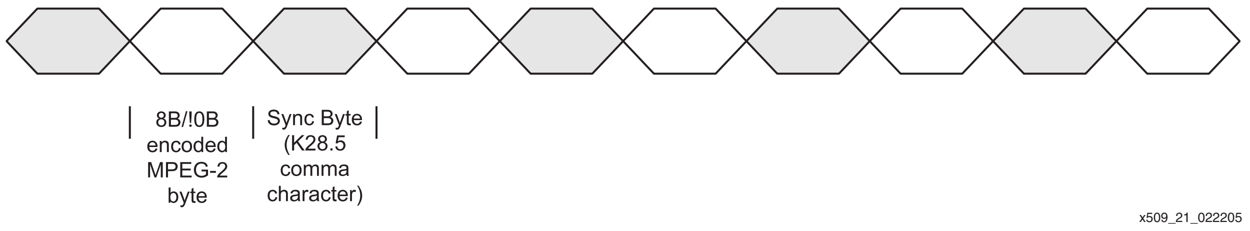


Figure 4-4: **Interleave Byte Mode**

DVB-ASI specifications dictate that each MPEG-2 packet must be preceded by two sync bytes. This enables re-synchronization within one MPEG-2 packet, in case of a link failure. For more details on the DVB-ASI transport mode specification, refer to the DVB-ASI specifications.

In the reference design, the TRANSPORT_MODE parameter controls the transport mode of the test bitstream. This parameter is located in the `dvb_top.v` or `dvb_top.vhd` source file. When this parameter is set to 0, the BIST ASI transmitter transmits in contiguous byte mode (see Figure 4-5). When this parameter is set to 1, the BIST ASI transmitter transmits in interleave byte mode (Figure 4-6).

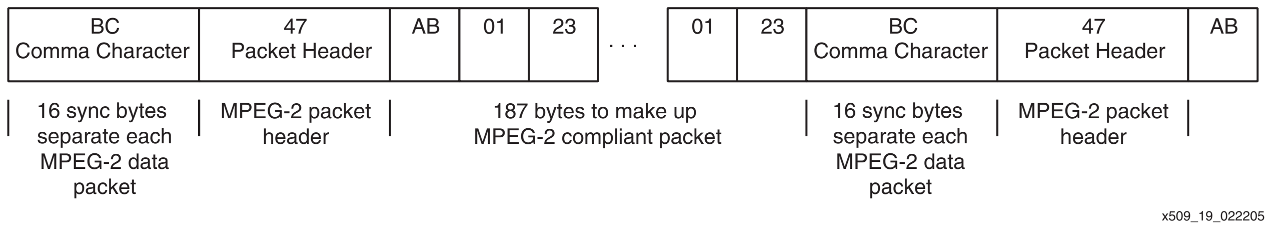

BC
Comma Character
47
Packet Header
AB
01
23
. . .
01
23
BC
Comma Character
47
Packet Header
AB
16 sync bytes separate each MPEG-2 data packet
MPEG-2 packet header
187 bytes to make up MPEG-2 compliant packet
16 sync bytes separate each MPEG-2 data packet
MPEG-2 packet header
x509_19_022205

Figure 4-5: **BIST Test Bitstream in Contiguous Byte Mode**

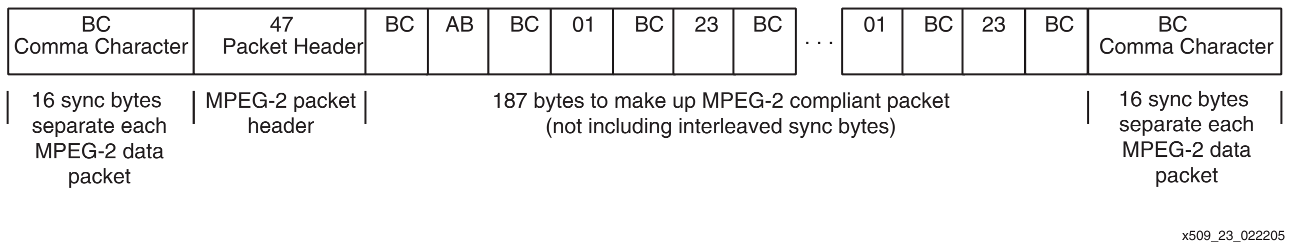

BC
Comma Character
47
Packet Header
BC
AB
BC
01
BC
23
BC
. . .
01
BC
23
BC
BC
Comma Character
16 sync bytes separate each MPEG-2 data packet
MPEG-2 packet header
187 bytes to make up MPEG-2 compliant packet (not including interleaved sync bytes)
16 sync bytes separate each MPEG-2 data packet
x509_23_022205

Figure 4-6: **BIST Test Bitstream in Interleave Byte Mode**

The TX_SEL parameter, located in the `dvb_top.v` or `dvb_top.vhd` source file, selects the transmit data path source. When this parameter is set to 0, the transmit data path is set to pass-through mode, which means the transmitter is sourced from the elastic FIFO on the receiver side. When this parameter is set to 1, the transmit data path is sourced from the BIST test bitstream generator. Table 4-2 summarizes these user parameters.

Table 4-2: **User Parameters**

Parameter	Type	Description
TRANSPORT_MODE	Bit	Indicates ASI transport mode: 0 : Contiguous Byte mode 1 : Byte Interleave mode
TX_SEL	Bit	Indicates transmit data path source: 0 : Pass-Through Mode (data sourced from Receive FIFO) 1 : BIST Mode (data sourced from BIST Test pattern generator)

Reference Design

The reference design is provided in VHDL and Verilog. Figure 4-7 shows the design hierarchy for the VHDL and Verilog design files.

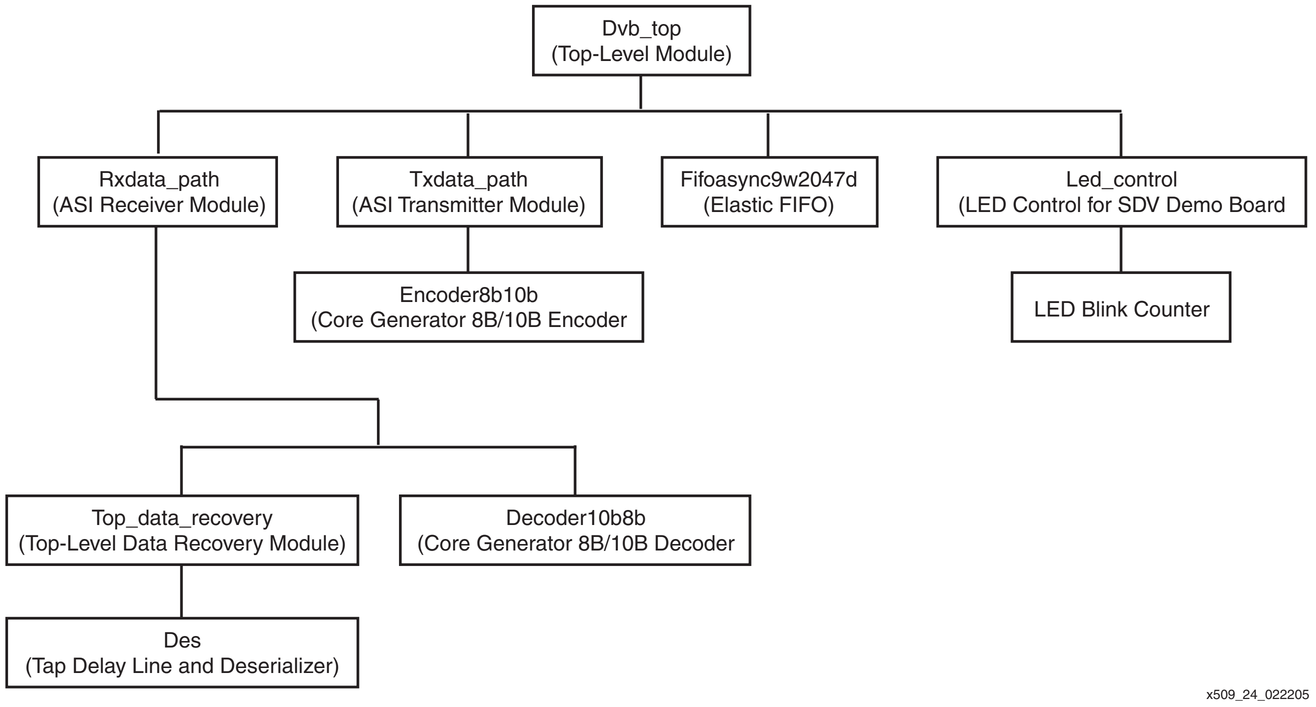


Figure 4-7: **Reference Design File Structure**

The data recovery module (`des.vhd` and `top_data_recovery.vhd`) is provided only in VHDL.

When synthesizing the Verilog design, mixed mode synthesis in Synplicity must be used.

SDV Demo Board Implementation

The DVB-ASI physical layer receiver and transmitter design is verified on the SDV Demo Board, available from Cook Technologies at www.cook-tech.com. The simplified board layout is shown in Figure 4-8. Proper settings for the board are shown in Table 4-3.

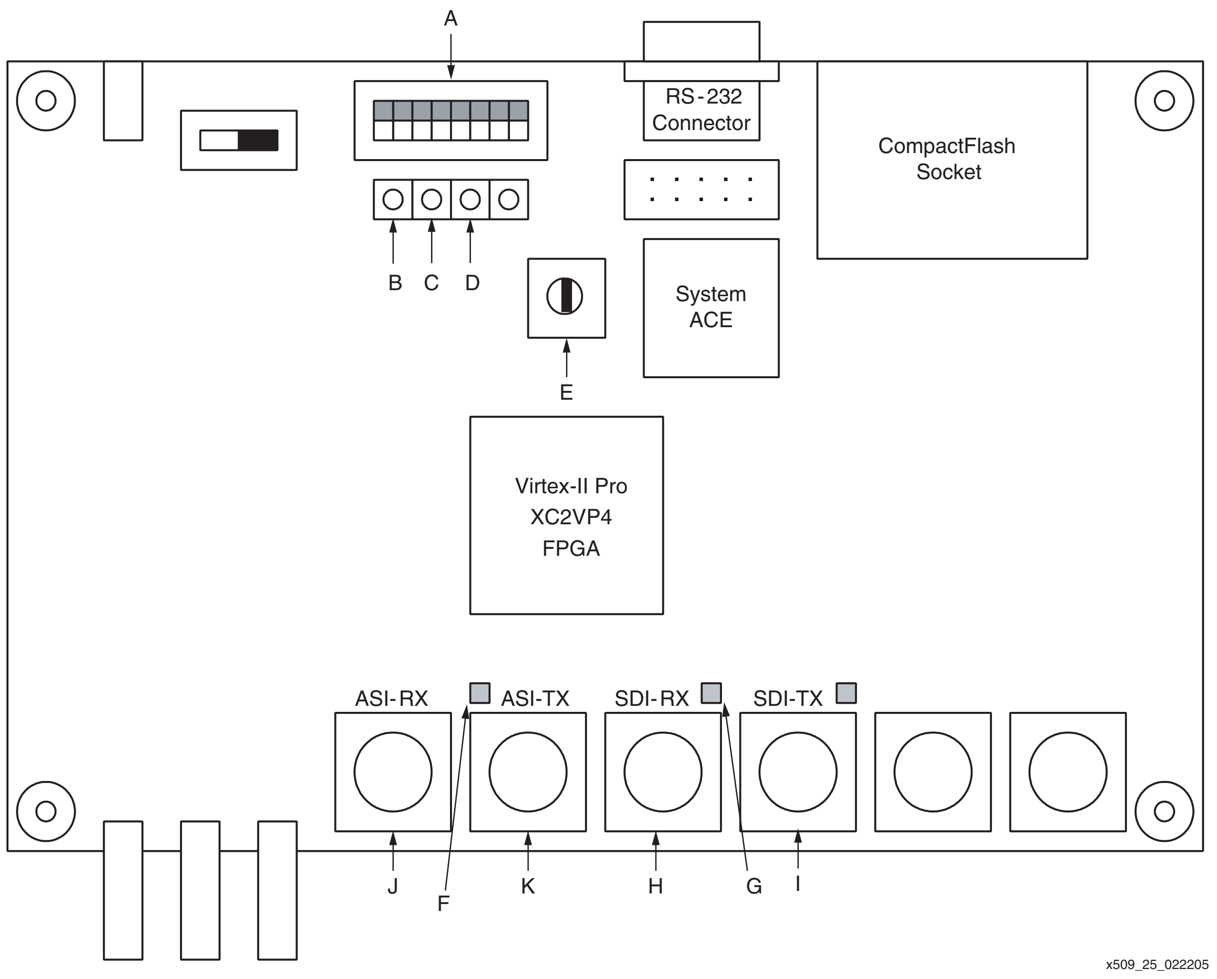


Figure 4-8: **SDV Demo Board Settings**

Table 4-3: **SDV Demo Board Settings**

Item	Description
A	All DIP switches set to ON
B	Push button switch 2 (SW2) System Reset
C	Push button switch 3 (SW3) Force TX Error
D	Push button switch 4 (SW4) Clear error LED
E	Rotary switch set to 0
F	8B/10B Disparity or Code Error LED
G	BIST Error LED
H	SDI-RX input connector: Connect ASI input here
I	SDI-TX output connector: Connect ASI output here
J	ASI-RX input connector: Not used
K	ASI-TX output connector: Not used

For this demonstration, SD-SDI connectors are used instead of DVB-ASI compliant connectors. This is a common practice in the industry. Advantages of using the SD-SDI connectors include the following:

- Ability to use the National Semiconductor CLC014 cable equalizer on the SDI-RX connector (not available on the ASI-RX connector)
- Ability to drive two standards (SD-SDI and DVB-ASI) from the same pair of connectors

The CLC014 cable equalizer allows the user to drive up to 300 m of cable, instead of only 70 m on the ASI-RX connector.

For more details about the SDI I/Os on the SDV Demo Board, refer to the board documentation available from Cook Technologies at www.cook-tech.com.

The default HDL code parameter settings are listed in Table 4-4:

Table 4-4: **Default Parameter Settings**

Parameter	File Location	Setting
RLOC_ORIG_CONST	`des.vhd`	X18Y0: Locks the placement of the tap delay line to the bottom side of the FPGA close to the SDI-RX pin.
SIDE	`des.vhd`	1: Indicates a bottom implementation.
SYNC_MODE	`des.vhd`	1: Receiver locks only when two sync bytes on the same byte boundary are found in a 5-byte window.
TRANSPORT_MODE	`dvb_top.v` `dvb_top.vhd`	0: Indicates contiguous byte mode.
TX_SEL	`dvb_top.v` `dvb_top.vhd`	0: Indicates pass-through mode (data from Receive FIFO).

LVDS I/O Standards

When implementing the design on the SDV Demo Board with an Engineering Sample (ES) part, the LVDS_25_DCI I/O standard must be followed for all LVDS inputs. When implementing the design on the SDV Demo Board with a Production part, the LVDS_25_DT I/O standard must be followed for all LVDS inputs. The LVDS I/O standards for input are set in the `dvb_top.ucf` user constraint file.

In the `dvb_top.ucf` file, use the following constraints for ES silicon:

```
INST "DATAIN_IBUFDS" IOSTANDARD=LVDS_25_DCI;
INST "CLKIN_IBUFGDS" IOSTANDARD=LVDS_25_DCI;
```

For Production silicon, use the following constraints:

```
INST "DATAIN_IBUFDS" IOSTANDARD=LVDS_25_DT;
INST "CLKIN_IBUFGDS" IOSTANDARD=LVDS_25_DT;
```

Clocking

The reference design uses the 54 MHz clock source on the SDV Demo Board to derive all requisite clocks for both the receiver and transmitter. One DCM is used to generate the receive clocks and one DCM is used to generate the transmit clocks. Note that only one DCM each is required for the receiver and transmitter, regardless of how many receiver and transmitter modules are instantiated. Figure 4-9 and Figure 4-10 show DCM settings for both the receiver and transmitter clocks.

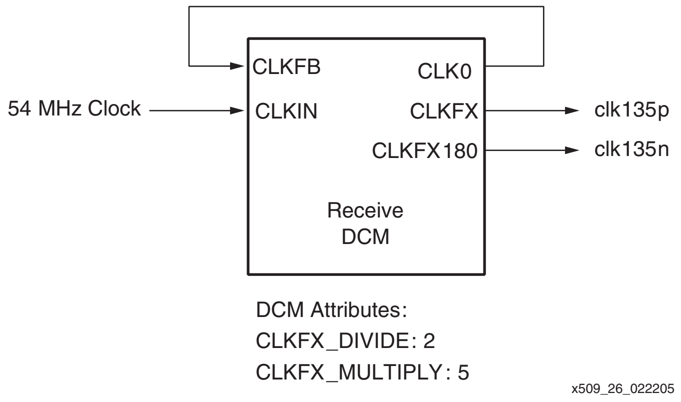


Figure 4-9: **Receiver DCM Settings**

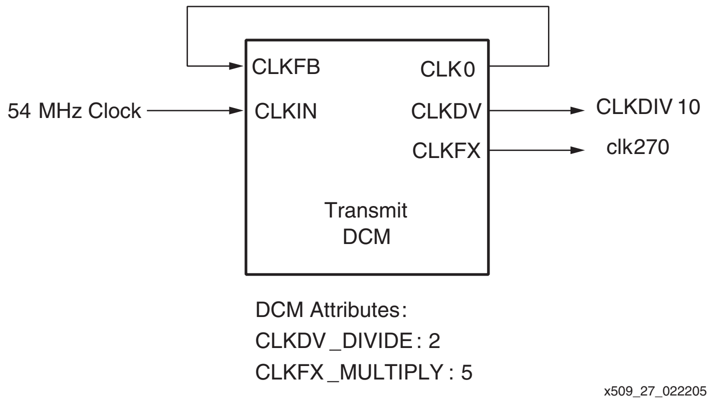


Figure 4-10: **Transmitter DCM Settings**

Error Detection

The reference design detects two types of errors: 8B/10B and BIST errors. 8B/10B errors are simply an OR function of the disparity error and code error flags of the 8B/10B decoder core. Whenever the receiver encounters a disparity error or code error, the 8B/10B error LED is asserted. The receiver automatically goes into reframe mode and starts looking for sync bytes to which it can reframe. However, the error LED remains on and must be cleared by pushing the **Clear Error LED** button (SW4).

The BIST error LED is asserted when the receiver detects a BIST error. This LED remains on until the **Clear Error LED** button (SW4) is pushed.

When the design is first loaded onto the SDV Demo Board, either of the two error LEDs might light up. Push the **Clear Error LED** button (SW4) to clear them.

BIST Test

The BIST test is performed as follows:

1. Set the TRANSPORT_MODE parameter to either contiguous byte mode or interleave byte mode.
2. Set the TX_SEL parameter to 1.
3. Implement the code, and load the bitstream onto the SDV Demo Board.
4. Connect the BNC connector of the SDI-TX output to the SDI-RX input with a 75-ohm cable, as shown in Figure 4-11.

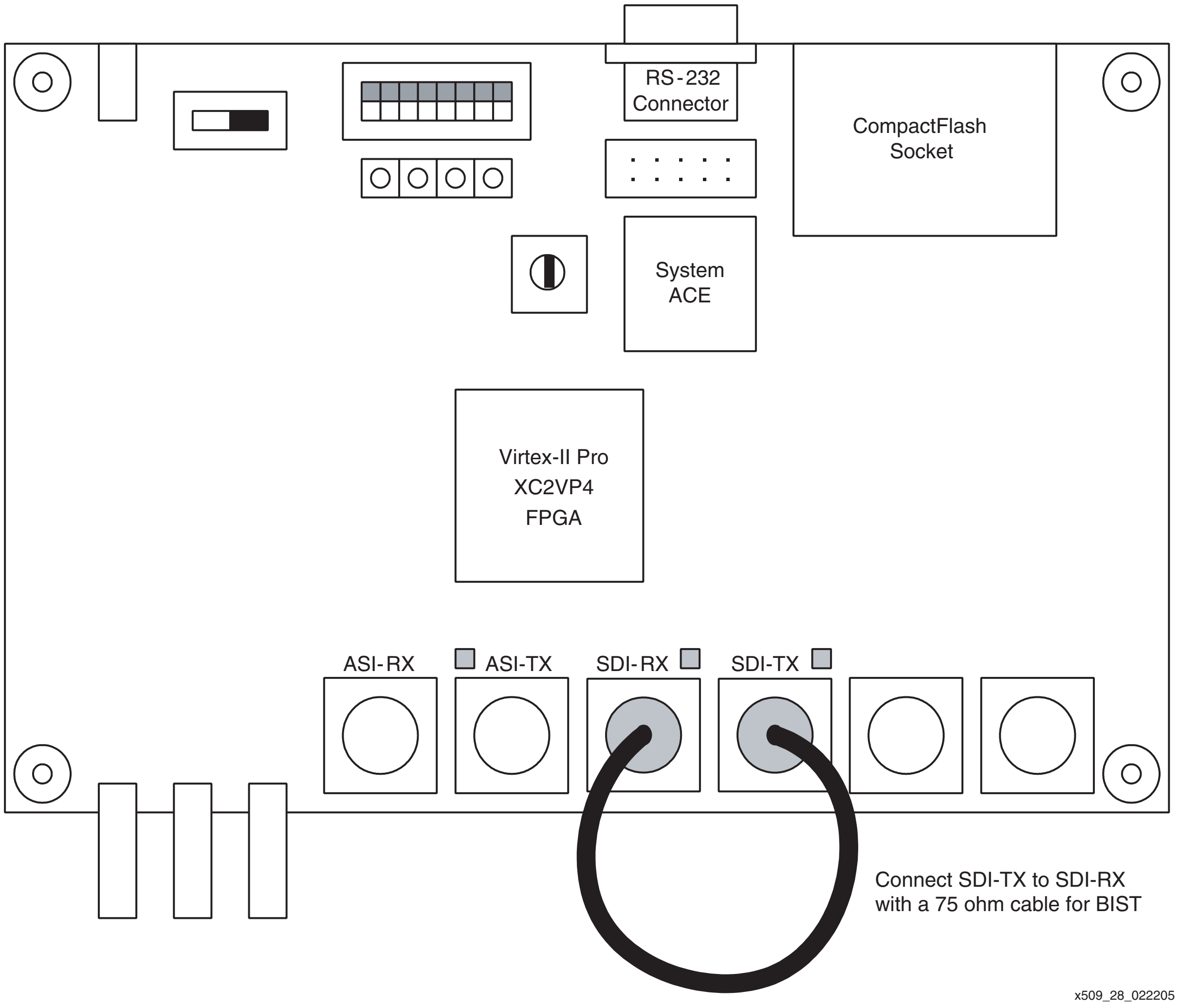


Figure 4-11: **BIST Setup**

5. Upon downloading the bitstream, if either of the error LEDs light up immediately, clear the LEDs by pushing the **Clear Error LED** button (SW4).
6. The BIST operation is running. Monitor the error LEDs for any transmission errors.

Conclusion

This application note describes an implementation of the DVB-ASI physical layer. Using parameters, users can easily configure the reference design in either pass-through mode or BIST mode. The reference design is implemented on the SDV Demo Board, available from Cook Technologies at www.cook-tech.com.

The reference design has been verified to synthesize with Synplify.