LVDS System Data Framing

XAPP238 (v1.0) December 18, 2000

Summary

This document describes an implementation of a low-overhead data synchronization and framing method to use with the LVDS capability of Virtex-E devices described in **XAPP233**.

Introduction

There are two general methods for synchronizing serial data: exclusive pattern and repetitive pattern. Exclusive pattern synchronization excludes the sync pattern as a data pattern. This method uses very simple sync recognition logic, but at the expense of 50 percent of the data space. Typically, a bit is prefixed to the basic unit of data. For example, if the data unit is a byte, it is expanded to nine bits (eight bits plus a control bit). If the control bit is "0", then the byte is data. If the control bit is "1", then the byte is a control word (sync for example). In this 8-bit data example, the overhead of the sync is greater than 11 percent and the payload is less than 89 percent. (Table 1 shows more comparisons.)

Table 1: **Efficiency Using Exclusive Pattern**

Word Size	Overhead %	Payload %
4-bit data	20.00	80.00
8-bit data	11.10	88.90
16-bit data	05.90	94.10
32-bit data	03.00	97.00

The second method for synchronizing serial data uses a repetitive pattern, allowing the sync pattern to be included as a data pattern. "Sync" is defined as a particular pattern appearing at regular intervals (see Figure 1). For example, sync pattern `0xF00D` is inserted into the data stream every 256 data words. While this method entails complex sync-detection logic that must look at multiple data frames before it can identify the location of the sync word, once synchronized, it is more efficient (see Table 2).

Table 2: **Efficiency Using Repetitive Sync Pattern**

Sync Word Every. . .	Overhead %	Payload %
128 data words	0.78	99.20
256 data words	0.39	99.60
512 data words	0.19	99.80
1024 data words	0.10	99.90

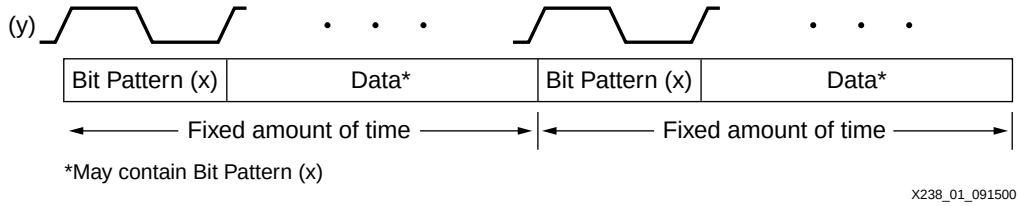

Figure 1: **Generic Repetitive Sync Pattern**

Figure 1 shows an example of a repetitive pattern-based sync. With this scheme, the following two parameters must be specified by the designer:

1. a sync bit pattern (x), for example `0xF00D`
2. a sync repetition rate (y), for example 257 clocks

The sync pattern *x* is inserted every *y* clocks. In a typical repetitive pattern scheme, more than 99 percent of the bits transferred are data.

In real applications, data frames and garbage frames are differentiated by validated checksums or some other application-specific characteristic of the data. For this example application, non-data frames are characterized by containing only the filler pattern (`0xDEAD` and `0xC0DE`).

The sync bit pattern used in this application is `1111 0000 0000 1101` (`0xF00D`) and the sync repetition rate (frame size) is 256 16-bit words plus sync. Figure 2 shows a typical frame bounded by the 16-bit sync words.

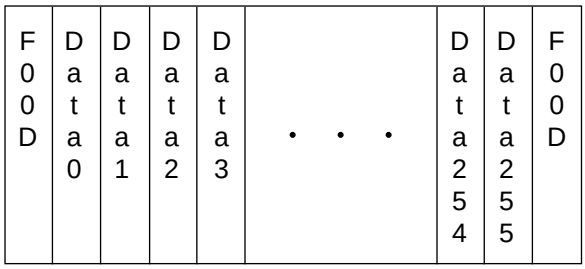

Figure 2: **Typical Frame**

This application note and accompanying reference designs present a general data synchronization solution (using the repetitive sync pattern method) and is a companion to the LVDS reference design presented in **XAPP233**. Together, these documents are the basis of a complete LVDS transmitter and receiver design, with data framing and synchronization, for use in a commercial product. These reference designs are based on a 16-bit data word with a 256-word data frame. While the designs are scalable, permitting users to decide on the appropriate word widths, frame sizes, and sync patterns, this document addresses neither customization nor queue management.

General LVDS Operation

Figure 3 shows the seven major elements of a typical LVDS-based system, however, only frame alignment (Table 3, element 3) and frame insertion (Table 3, element 5) are covered in this document and accompanying reference designs. The frame insertion logic contains all the logic necessary to insert framing information into the user data stream. The frame alignment logic is used to realign and frame an LVDS data stream.

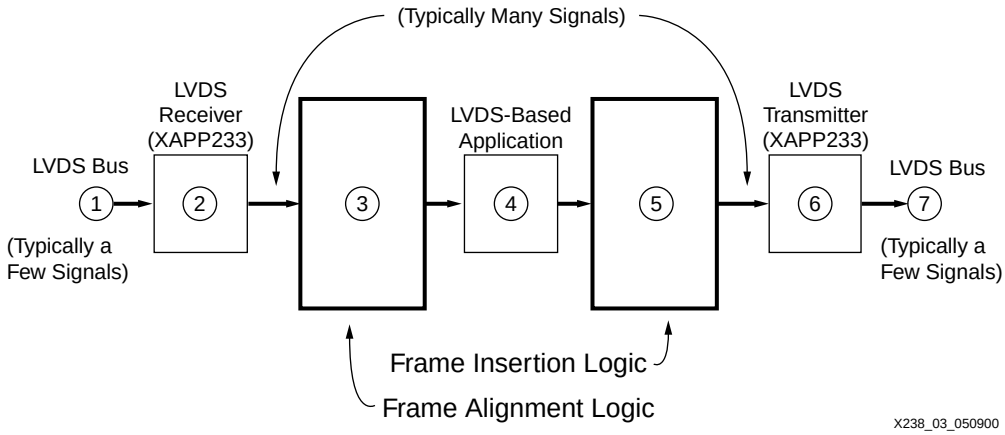

Figure 3: **Typical LVDS-Based System**

Table 3: **Description of Typical LVDS Elements**

Element	Description
LVDS Bus (incoming)	The incoming LVDS data stream, along with at least one clock, is carried on one or more controlled impedance transmission lines. (The clock may be imbedded in the data.)
LVDS Receiver	The LVDS receiver (as described in XAPP233) contains low-voltage differential receivers and high-speed serial-to-parallel logic. (LVDS bus signals are very high speed; the LVDS bus is usually fanned out by the receiver to provide data at a manageable speed.)
Frame Alignment Logic	The frame alignment logic is sometimes as simple as a byte alignment circuit; in this application, it contains both word alignment and payload framing.
LVDS-Based Application	The LVDS-based application can be any function, such as sorting, routing, or storing.
Frame Insertion Logic	The frame insertion logic controls the insertion of sync words and filler data when user data is not available.
LVDS Transmitter	The LVDS transmitter typically contains the high-speed, parallel-to-serial converter and the LVDS line drivers, such as the LVDS transmitter described in XAPP233.
LVDS Bus (outgoing)	The outgoing LVDS bus must have the same number of data and clock signals as the target LVDS receiver.

Frame Insertion

The frame insertion logic simply sends a sync word, followed by a frame of user data or filler data, and then repeats. The start and end of data within a stream is identified by headers, tags, or whatever method suits the user's application. The reference design accompanying this application inserts `0xF00D` every 256 words. When user data is unavailable, a frame of filler data is substituted to maintain the synchronized flow of frames.

Figure 4 shows a frame of filler data that is sent in the absence of user data. Sending filler frames allows the receiving module to maintain (or acquire) lock, even if there is no valid data being sent.

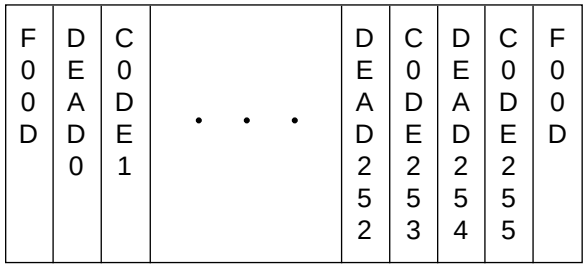

Figure 4: **One Frame with Filler Data**

The frame insertion logic, Figure 5, is simply a multiplexer controlled by a state machine that chooses user data or a constant. The state machine chooses what to send based on the user control input.

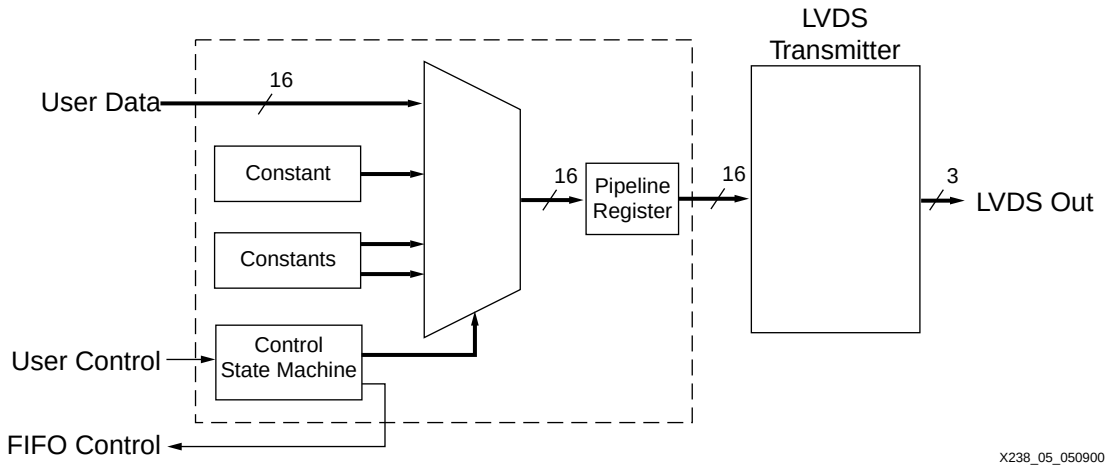

Figure 5: **Frame Insertion Block Diagram**

The frame insertion schematic, Figure 6, consists of five major modules, including a control state machine (RE_CTL), a constant generator for the sync word (SYNCWORD), and a constant generator for the filler data (DEADCODE).

The pipelined one-hot multiplexer (OHM4_1X16) controlled by RE_CTL sends the user data, the filler data, or a sync word. A sync word is sent followed by 256 words of user data; if the user data is not present, the multiplexer sends filler data (`0xDEAD` and `0xC0DE`). A final pipeline register in the outbound data path helps maintain high-speed operation.

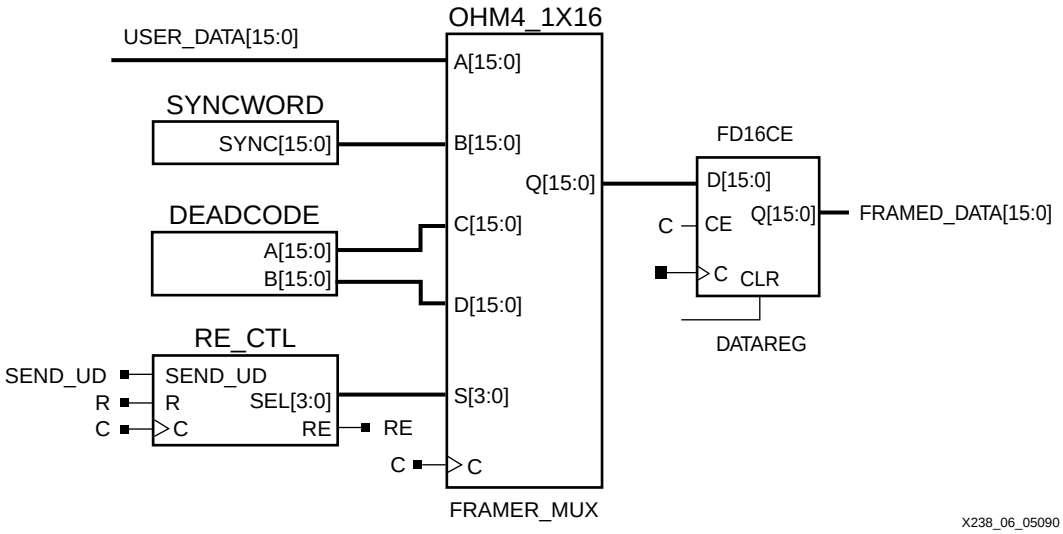

Figure 6: **Frame Insertion Schematic**

RE_CTL State Machine Operation

Referring to the flowchart in Figure 7, the state machine begins in STATE_0 and asserts SEL1, causing the multiplexer to select the sync word as the data to be sent on the next clock.

The state of SEND_UD is then checked to determine if there is user data to be sent. If there is, the state machine advances to STATE_3 and asserts SEL0 to select the user data. In this state, FIFO read enable (RE) is also asserted, informing the user that words are being taken. These two signals remain asserted until the TX_PITCH counter asserts TC, indicating that 256 words have been sent and returning the state machine to STATE_0.

When in STATE_0, if SEND_UD is false the state machine advances to STATE_1, beginning a two-state loop that causes the multiplexer to send a word of `0xDEAD` followed by a word of `0xC0DE`. This continues until the TX_PITCH counter asserts its TC, once again returning the state machine to STATE_0.

The TX_PITCH counter determines the frame length. In this example it is 257 clocks: 256 clocks for the data and one clock for the sync. This application uses an LFSR (linear feedback shift register) because of its economy of routing resources and inherent high-speed operation, however, any counter will work. The LFSR has a maximum count of 65,535 and can be scaled downward to suit the user's application.

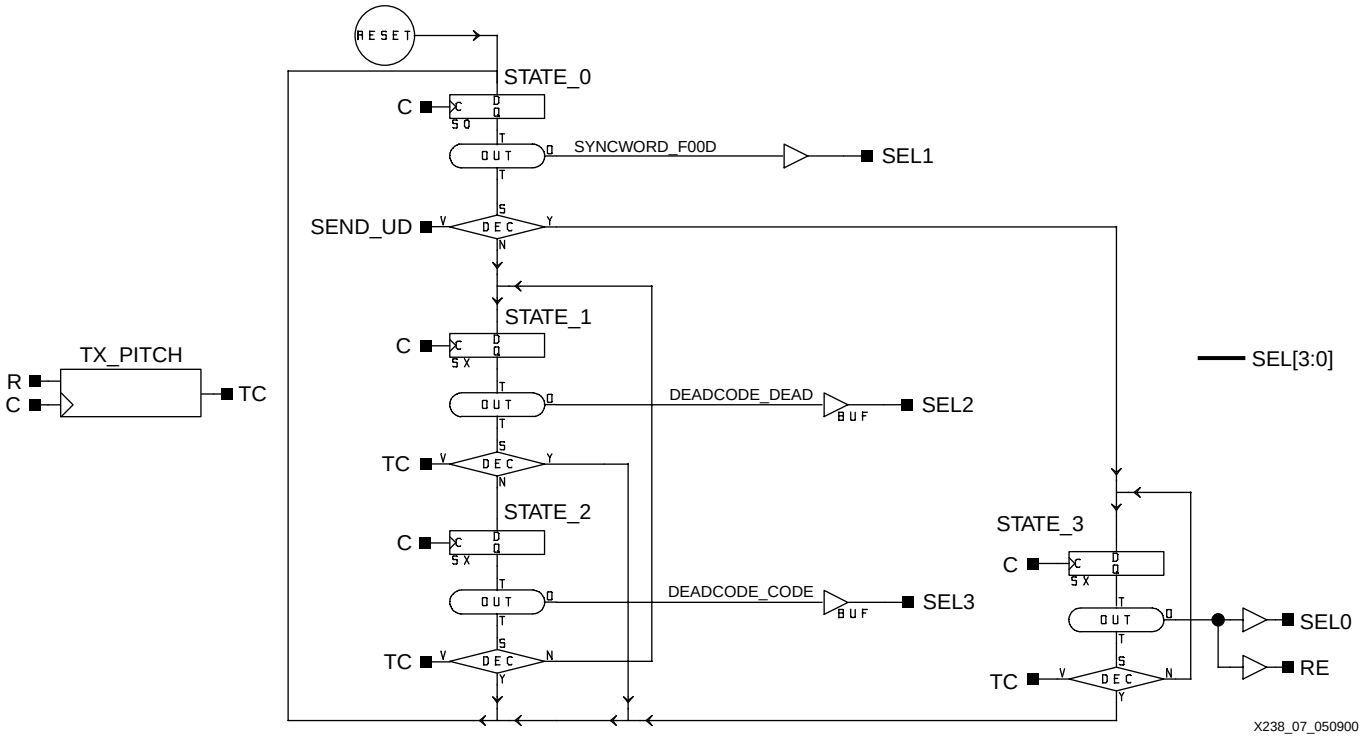

Figure 7: **RE_CTL State Machine Schematic**

Sending User Data

To transmit data, the user queues up 256 words of data and then asserts the SEND_UD signal. The RE_CTL state machine responds by asserting the RE signal (user data read enable). It continues sending user data until it has sent 256 words. After sending another sync word (`0xF00D`), the state machine checks the state of SEND_UD again and determines whether to send another frame of user data or a frame of filler data (`0xDEAD` and `0xC0DE`).

Frame Alignment

The frame alignment logic, Figure 8, detects the user-specified sync word within the data stream, using this pattern to determine the bit offset within the stream to the start of the word. It then steers a barrel shifter to correct for this offset.

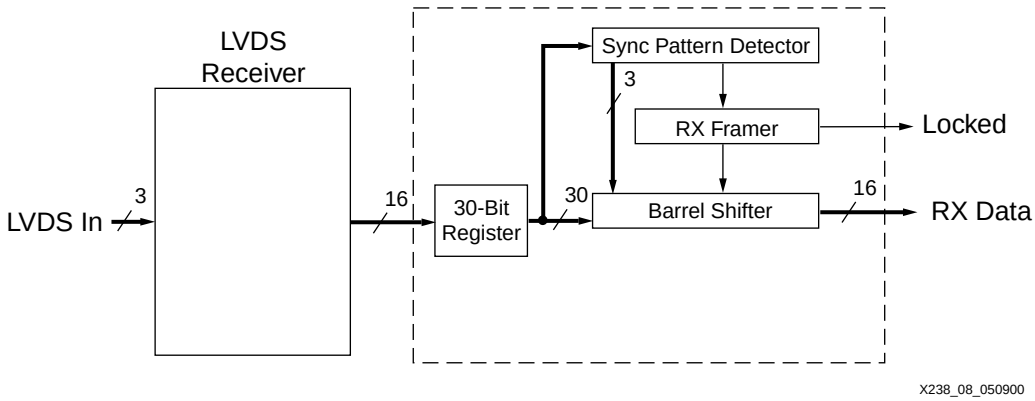

Figure 8: **Frame Alignment Block Diagram**

Figure 9 shows the schematic of the frame alignment logic. The frame alignment logic anticipates the subsequent arrival of additional sync words based on the designated sync interval (256 words in this example) and maintains "locked" status as long as sync words continue to appear in this position. It accomplishes this as follows:

1. Words from the LVDS receiver cascade through a 30-bit register
2. Eight comparators in the SYNC_PAT module simultaneously look for the presence of the sync word on all eight possible bit-pair boundaries
3. When the sync word is detected, SYNC_PAT registers the offset and steers a 16-bit barrel shifter (DATBRLSH) to the correct boundary within the 30-bit buffer
4. RXFRAMER counts the 256 words following the sync word; then, it checks the following word again for the presence of the sync pattern
5. After three consecutive cycles of successfully detecting sync, the RXFRAMER considers the receiver to be locked and asserts the LOCKED signal

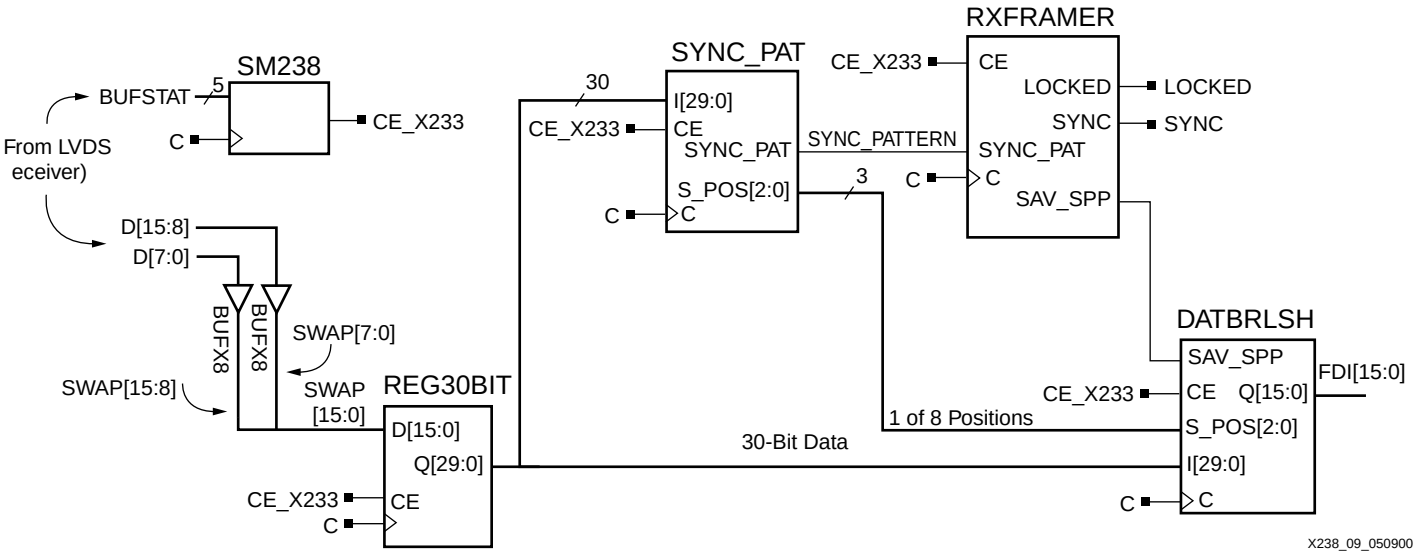

Figure 9: **Frame Alignment Schematic**

Frame Alignment Logic Detail

The frame alignment logic is heavily pipelined to help achieve timing constraints. The incoming 16-bit data is sourced from the low-level LVDS receiver described in **XAPP233**. Each module of the frame alignment logic is described below:

SM238

The SM238 module evaluates the BUFSTAT signals coming from the receiver. When BUFSTAT indicates that the receiver's FIFO has data, SM238 asserts CE_X233, causing the frame alignment logic to process the buffered words. Processing continues until the BUFSTAT signals indicate that the FIFO is almost empty, at which time the SM238 module deasserts CE_X233.

BUFX8s

The **XAPP233** FIFO presents data *byte swapped*. Two BUFX8 components preceding the 30-bit register reorder the bytes.

REG30BIT

The 30-bit register is constructed from two cascaded registers and receives 16-bit input data from the receiver. Both registers update on every clock. All 30 bits are continuously available to the SYNC_PAT and DATBRLSH modules.

SYNC_PAT

The sync pattern recognition circuit decodes 16 bits of sync pattern. It accomplishes recognition by way of eight identical comparators, each left-shifted by two bits from one another to cover all possible positions of sync in the 30-bit field. Using the bit-mapping suggested in **XAPP233**, D0 contains all even bits and D1 contains all odd bits. This reduces the number of possible skew positions within the stream by ensuring that the word starts on an even-bit boundary. The SYNC_PATTERN signal informs RXFRAMER that a sync pattern has been found in the data stream. The position of the sync pattern is sent to the DATBRLSH module as a binary-encoded vector.

RXFRAMER

RXFRAMER is a one-hot state machine that finds frame sync. In its initial state, the state machine waits for the SYNC_PATTERN signal, which indicates that the SYNC_PAT module has decoded the sync pattern. When the sync pattern is found, the state machine initializes a sync pitch counter (RX_PITCH) and waits for TC. The terminal count normally coincides with the next sync pattern. If a sync pattern is indeed present, a possible frame boundary is found and the state machine re-initializes the RX_PITCH counter and waits for a third occurrence of a sync pattern. If the third pattern is found, the state machine assumes detection of a frame sync and asserts LOCKED. It also asserts SAV_SPP (save sync pattern position), which latches the three bits of sync pattern position into the barrel shifter.

DATBRLSH

This module decodes the 3-bit sync pattern position (from SYNC_PAT) and uses it to select the correct alignment of the 16-bit data within the 30-bit field assembled by REG30BIT. The data is then shifted to the correct position and presented to the user application.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
12/18/00	1.0	Initial Xilinx release.