HDLC Controller Implemented in MachXO, LatticeXP2 and LatticeECP2/M Families

June 2010 Reference Design RD1038

Introduction

HDLC is the abbreviation for High-Level Data Link Control published by the International Standards Organization (ISO). This data link protocol is located at the link layer (layer 2) of the 7-layer OSI reference model. Today, a variety of link layer protocols such as LAPB, LAPD, LLC and SDLC are all based on the HDLC protocol with a few modifications. These single-channel and multi-channel HDLC controller reference designs, targeted for the MachXO™, LatticeXP2™, LatticeECP2™ and LatticeECP2M™ families respectively, can easily be used or modified for these HDLC applications.

Features

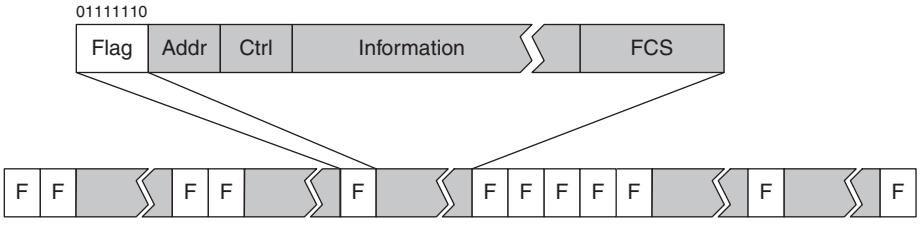
- Parameterizable number of HDLC channels in multi-channel design. Each channel corresponds to a DS0 channel in the TDM (Time Division Multiplexing) PCM (Pulse Code Modulation) highway.
- CRC (Cyclic Redundancy Check) check with parameterizable FCS (Frame Check Sequence) length and arbitrary polynomials.
- Each channel has two separate 8-bit data buses, one for the receiver and another for the transmitter. These buses can be connected to external memory such as FIFOs or memory controller modules for interfacing with the host processor.
- Flag insertion and detection
- Abort generation and detection
- Zero insertion and deletion
- Idle insertion
- Flag sharing between HDLC frames
- Recognize 011111101111110 as two continuous flags
- Conforms to ISO/IEC 3309
- Supports MachXO, LatticeXP2, LatticeECP2/M devices

Functional Description

The HDLC is a bit-oriented protocol with the serial transmission data encapsulated by 01111110 flags. An HDLC frame is composed of the flag and the serial transmission data. There are five fields in an HDLC frame: flag, address, control, information (variable length), and FCS. The FCS is calculated according to the CRC error detecting scheme from the serial bit stream of address, control, and information fields. It is usually a 16-bit or 32-bit pattern used for checking the frame data integrity.

In addition to separating the serial transmission data, the HDLC flag can also be used to fill the time gap when there is no data to be transferred. Figure 1 shows the HDLC frame format and the typical HDLC bit stream.

Figure 1. HDLC Frame Format



The flag pattern, 011111110, is also a possible sequence in other HDLC fields. In order to make the flag unique to the whole bit stream, a zero insertion and deletion technique is applied to the nonflag fields. For data transmission, whenever there are five consecutive 1's being transmitted, an additional redundant zero bit will be inserted immedi-

ately after the five 1's. This is called “zero insertion” or “zero stuffing”. When receiving data, whenever there are five consecutive 1's followed by a zero, the zero will be ignored. This is called “zero deletion” or “zero unstuffing”.

In some cases when there is a priority issue or a problem on the data link, the transmitter may want to abandon the transmission of the current HDLC frame before it is fully transmitted. This is done by asserting the abort sequence, at least seven but fewer than 15 consecutive 1's. If the number of 1's is more than 15, it will be recognized as an idle sequence. Instead of transmitting consecutive back-to-back flags, idle sequence can also be used when no data needs to be transferred. However, the idle sequence is usually used to support half-duplex operation. When the idle sequence is received, the transmission direction will be reversed. The half-duplex operation is not supported in this design.

The designs treat the address, control, information, and FCS fields as transmission data. The receiver strips away the flags of the bit stream and converts these nonflag fields (including the FCS field) from serial bit stream to parallel 8-bit octets. The receiver also checks the correctness of the FCS field and reports the status at the end of the receiving process. Contrary to the receiver, the transmitter converts the address, control, and parallel-to-serial data, then generates the FCS, and finally encapsulates these fields with HDLC flags.

These reference designs support three most commonly used CRC polynomials. Please contact Lattice technical support if you need a special CRC polynomial. The supported CRC polynomials are:

$$\text{CRC-16} = x^{16} + x^{15} + x^2 + 1$$
$$\text{CRC-CCITT} = x^{16} + x^{12} + x^5 + 1$$
$$\text{CRC-32} = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

The multi-channel design contains a programmable number of HDLC channels working in the channelized mode which uses a synchronization pulse to subdivide the serial TDM data stream (PCM frames) into a set of 8-bit time slots. Each time slot corresponds to a DS0 channel. Each DS0 channel associates with one HDLC channel. Also, the synchronization framing bit can be a separate bit in the TDM data stream framing format or be asserted simultaneously with the last data bit in the last time slot. Figure 2 shows the difference between the two.

The single-channel design is a derivation of more generalized multi-channel designs where only a single-channel is used.

Figure 2. TDM System Signals and Timing Diagram

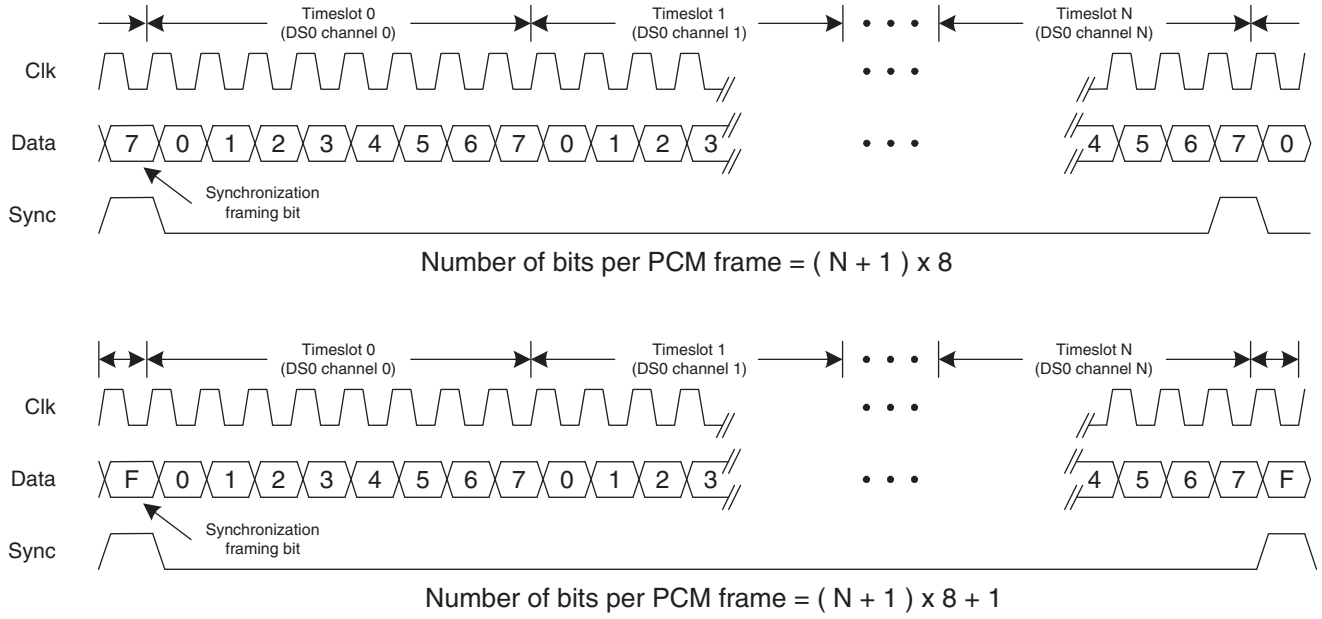
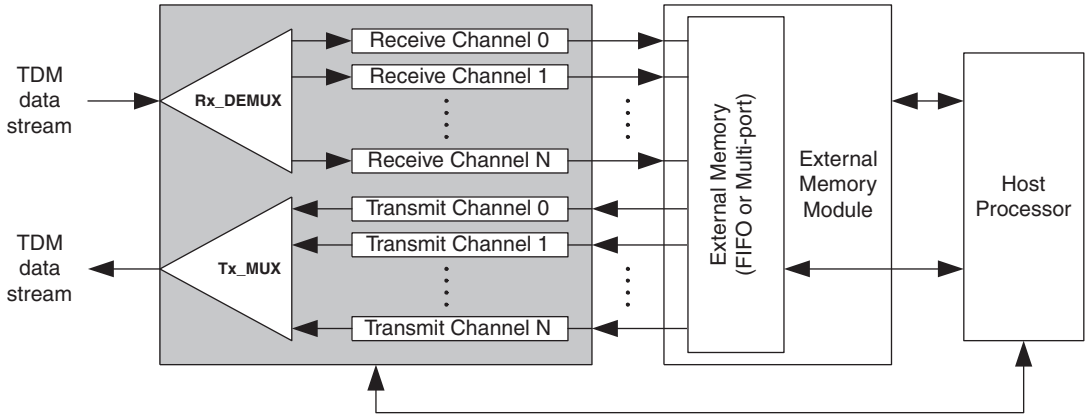


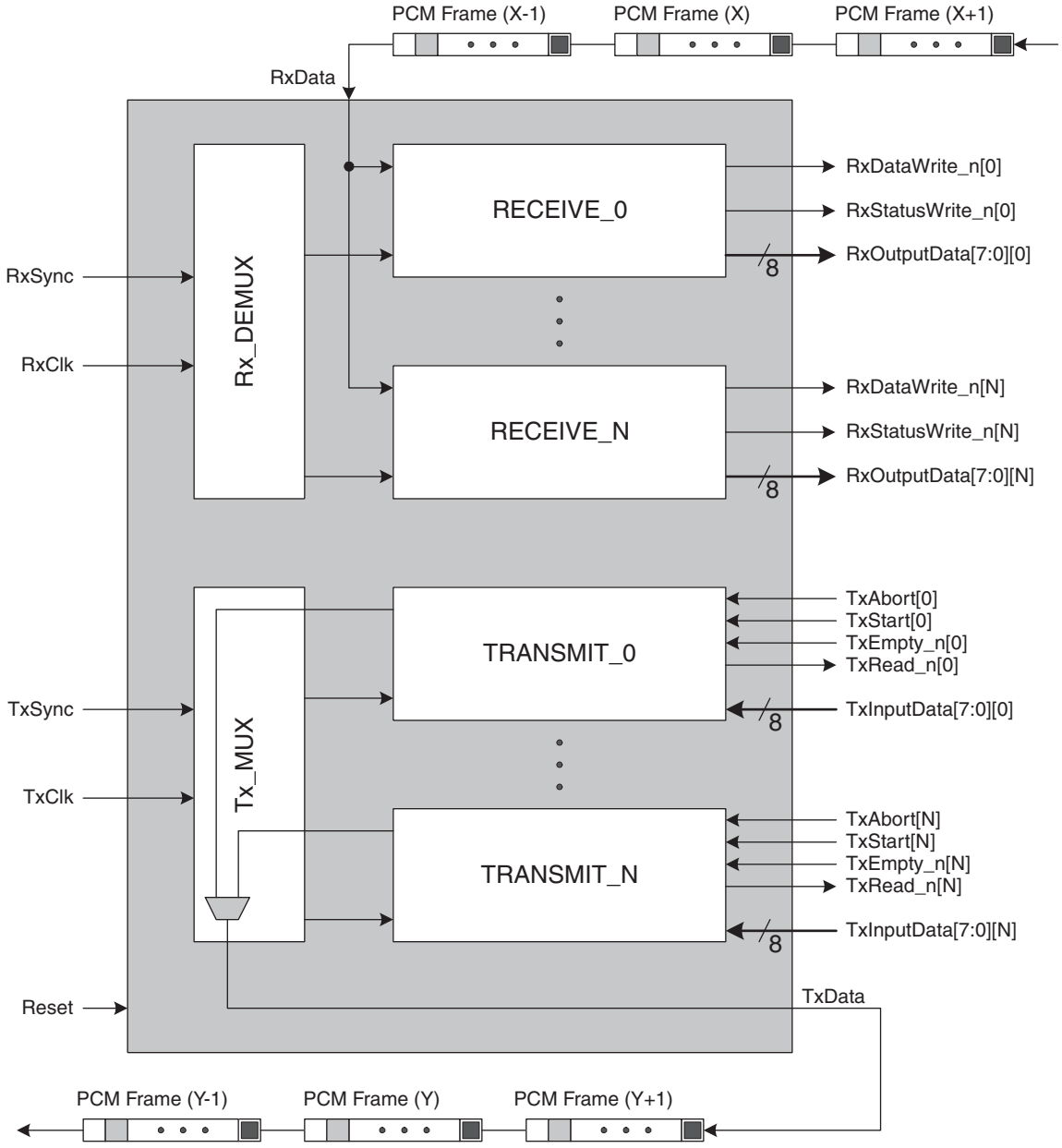
Figure 3 shows how this design may be used in a system. When an HDLC frame is received, the controller will convert the serial bit stream to parallel and write it to the external memory for the host processor to read. For transmission, the host processor writes the frame data into the external memory then triggers the controller to read the data from the external memory and converts it to HDLC serial bit stream. The external memory could be any kind of communication memory such as FIFO or multi-port memory that works as a buffer between the host processor and the HDLC controller.

Figure 3. MC-HDLC Controller in a System



The multi-channel design includes four different modules (i.e. Rx_DEMUX, Tx_MUX, RECEIVE, and TRANSMIT modules). The Rx_DEMUX and Tx_MUX modules will be instantiated just once in the multi-channel design, however, the RECEIVE and TRANSMIT modules will be instantiated as many times as the number of the channels. The block diagram of the multi-channel HDLC design is shown in Figure 4.

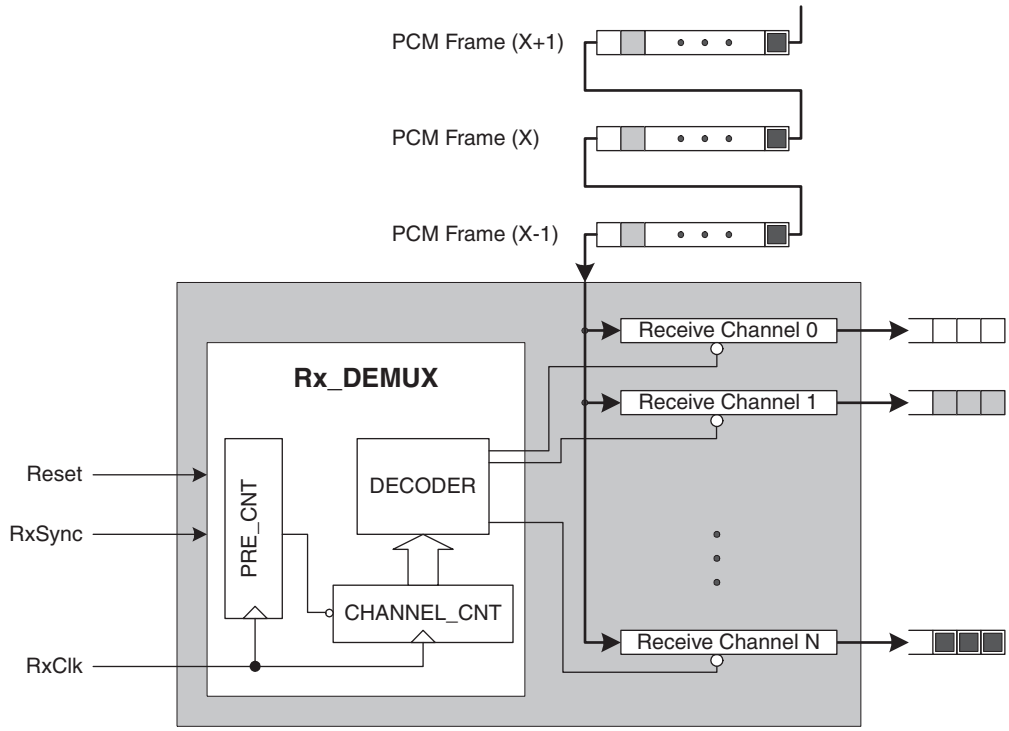
Figure 4. MC-HDLC Block Diagram



Rx_DEMUX Module

The Rx_DEMUX module is used to demultiplex the incoming PCM highway bit stream to the different HDLC receiver channels. Figure 5 shows the block diagram of this module and how it is used in the design. There are two counters, PRE_CNT and CHANNEL_CNT, in this module. Both counters are running at RxClk clock and will be reset to zero synchronously whenever the RxSync is high. The PRE_CNT counter is a fixed 3-bit counter. It enables the CHANNEL_CNT counter to count up one for every eight RxClk clocks. The value of CHANNEL_CNT will be sent to the DECODER sub-module. The DECODER output RxEnable[0:N] will then enable the transmitter of one channel at a time.

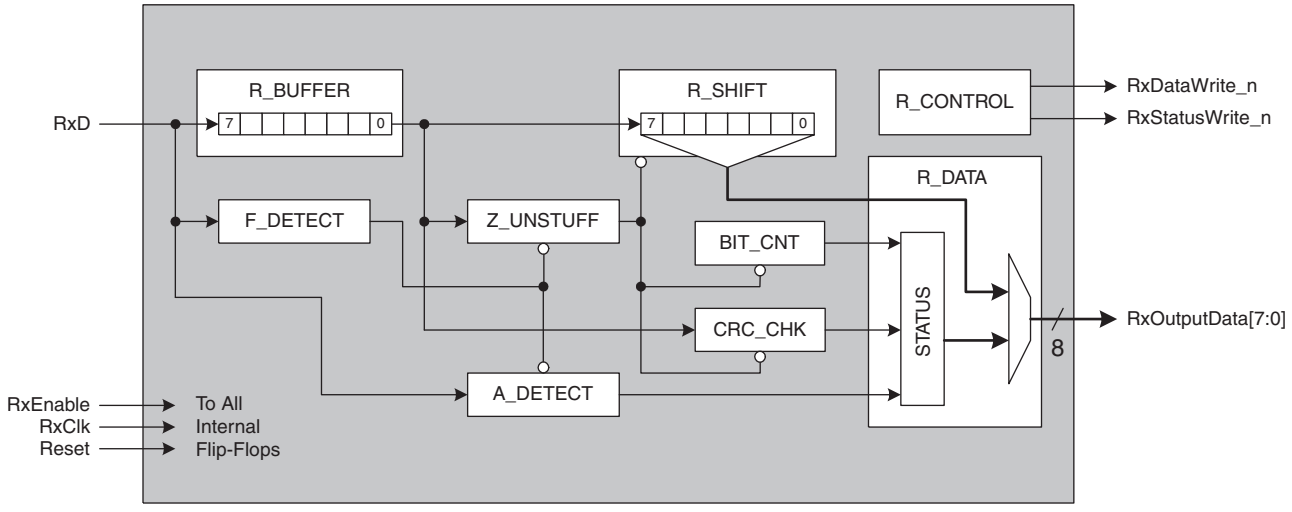
Figure 5. Rx_DEMUX Module in the MC-HDLC Design



RECEIVE Module

The RECEIVE module implements all the required HDLC receiver functions including flag detection, zero unstuffing, abort detection, and CRC checking. The block diagram of this module is shown in Figure 6.

Figure 6. RECEIVE Module Block Diagram



Data Receiving

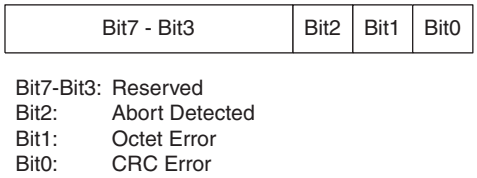
Once the F_DETECT sub-module detects the HDLC flag, after eight RxClk clocks, the Z_UNSTUFF and A_DETECT sub-modules will be enabled for zero unstuffing and abort detection respectively. Once enabled, the Z_UNSTUFF sub-module will keep track of the incoming bit stream and disable the downstream logic for one clock if a zero bit is followed by five consecutive 1's. So, the zero bit inserted to make the 01111110 flag unique will be

unstuffed from the bit stream. The original data without zero bits insertion will then be shifted into the R_SHIFT sub-module. The RxOutputData[7:0] bus will output the R_SHIFT data value once eight bits of data are collected. When this happens, the RxDataWrite_n signal will be asserted for one RxClk clock period to indicate that to the external memory. The FCS data at the end of the receiving HDLC frame will also be transmitted through the RxOutputData[7:0] bus with RxDataWrite_n asserted.

Receiving Frame Status Generation

The BIT_CNT and CRC_CHK sub-modules are used for detecting the error of the receiving HDLC frame. The BIT_CNT sub-module will report the octet error if the total number of bits received after zero unstuffing is not a multiple of eight (i.e. mis-aligned byte count). The CRC-CHK sub-module will check the FCS field to see if there is a CRC error. The RxOutputData[7:0] bus will output these results along with the result of the abort detection. This status will be reported after the entire HDLC frame is received or the abort is detected. The RxStatusWrite_n signal will be asserted for one RxClk clock period to indicate that the value present on RxOutputData[7:0] is the status instead of the data. The bit assignment of this status byte is shown in Figure 7.

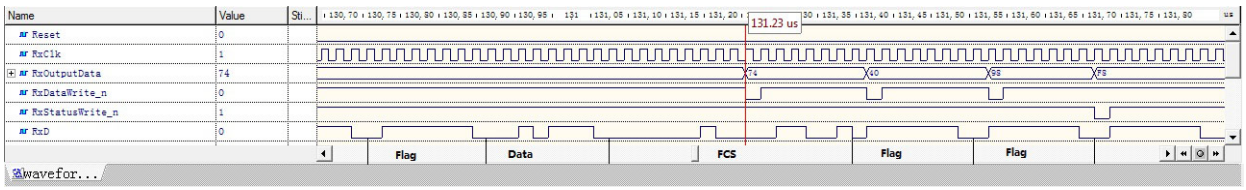
Figure 7. Status Bit Definition



Receive Module Signals and Timings

Both the received data and the status are transmitted through the same bus. The signals RxDataWrite_n and RxStatusWrite_n will be asserted exactly one RxClk clock to indicate what type of information is on the RxOutputData[7:0] bus. The functional simulation timing waveforms of the receiver module are shown in Figure 8. For depiction purposes, this example simulates a single channel instead of a multi-channel receiver.

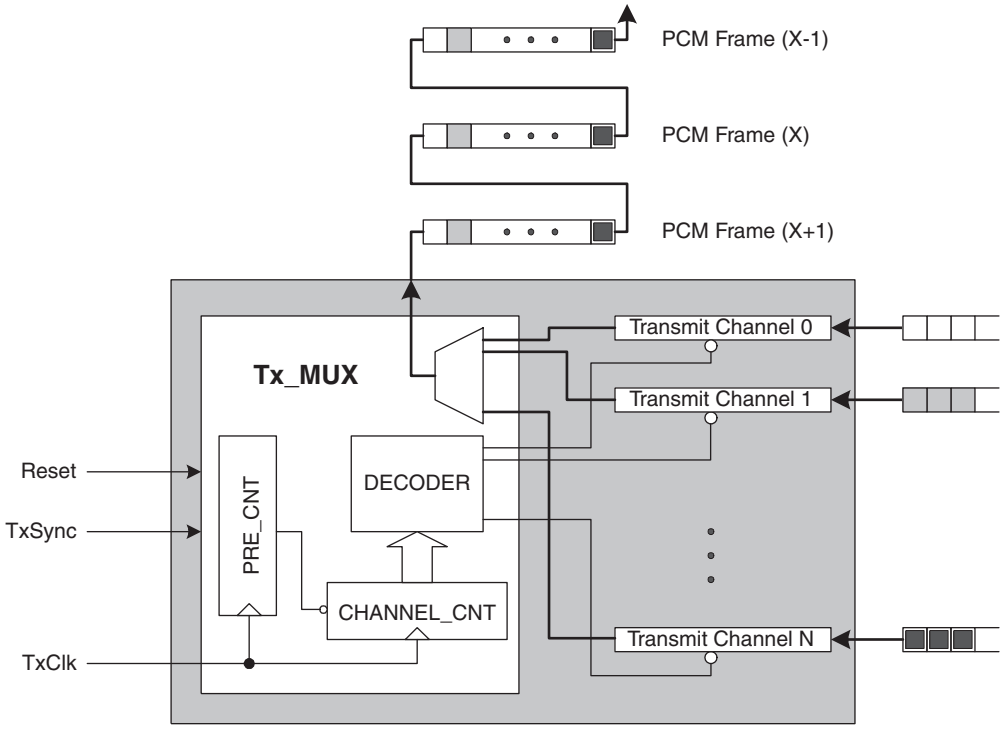
Figure 8. Single-Channel Receiving Timing



Tx_MUX Module

The Tx_MUX module multiplexes the outgoing bit streams of the HDLC transmitter channels to the PCM highway. Figure 9 shows the block diagram of this module and how it is used in the design. Similar to the Rx_DEMUX module, the Tx_MUX module contains the PRE_CNT counter, the CHANNEL_CNT counter, and the DECODER sub-module. In addition, a multiplexer is used for multiplexing the outgoing serial bit streams of the transmitter channels to the PCM high way output.

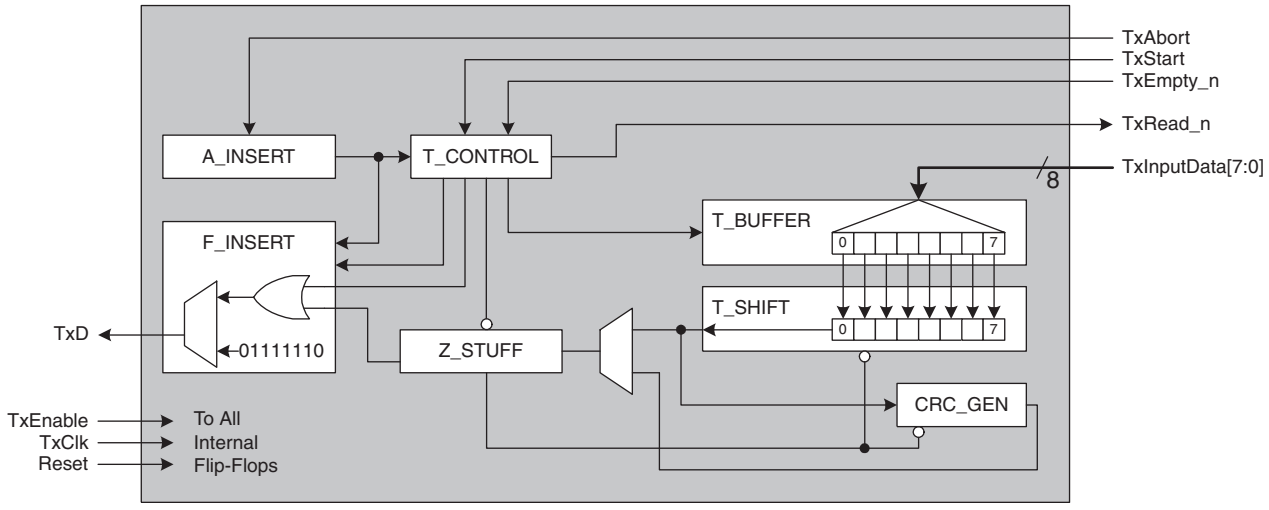
Figure 9. Tx_MUX Module in the MC-HDLC design



TRANSMIT Module

The TRANSMIT module implements all the required HDLC transmission functions such as flag insertion, zero stuffing, abort generation, and FCS generation for CRC check. The block diagram of this module is shown in Figure 10.

Figure 10. TRANSMIT Module block diagram



Data Transmitting

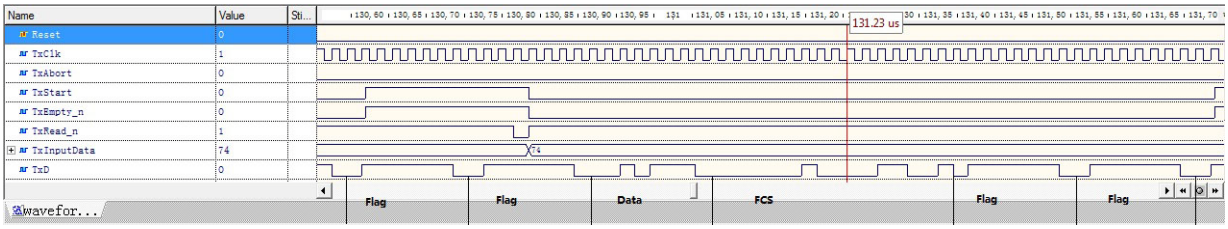
Before the transmission starts, the data needs to be stored in advance in the external memory such as FIFOs. The F_INSERT sub-module will keep asserting HDLC flags until TxStart is asserted. Once a high TxStart is detected, the TRANSMIT module will start reading the first octet from the external memory. It asserts TxRead_n signal for one TxClk clock and then latches the TxInputData[7:0] data into the T_BUFFER at the next TxClk clock. Once the

external memory samples a low TxRead at the rising edge of the TxClk clock, the data needs to be valid before the next TxClk rising edge and satisfy the setup time so that the data can be latched properly into the transmit module's T_BUFFER. The latched data will then be loaded into T_SHIFT and be shifted out of the shift register, through the Z_STUFF and the F_INSERT sub-modules to the TxD output. Before the first octet is completely shifted out through the TxD output, the second TxRead_n will be asserted to get the second octet. And then the third octet, the fourth octet, and so on. When latching the TxInputData bus, the active low signal TxEmpty_n will be examined as well. If it is low, the octet being latched into the T_BUFFER will be considered as the last octet of the current transmission frame. After this last octet is loaded into the T_SHIFT and shifted out, the MUX will switch from the T_SHIFT to the CRC_GEN and then shift the FCS out. The A_INSERT sub-module is used for asserting the aborting sequence (more than eight consecutive 1's) whenever a high TxAbort signal is sampled. The TxAbort signal needs to be asserted for at least one TxClk clock period. Idle assertion requires more than 15 consecutive 1's to be asserted and the TxAbort must be asserted for more than 15 TxClk clocks. The transmission aborting will be discussed later.

Transmitter Module Signals and Timing Waveforms

The functional simulation timing waveforms of the transmitter module are shown in Figure 11. For depiction purposes, the following example simulates a single channel rather than a multi-channel transmitter. CRC-CCITT checking is selected in all timing waveform examples in this document.

Figure 11. Single-Channel Transmitting Timing

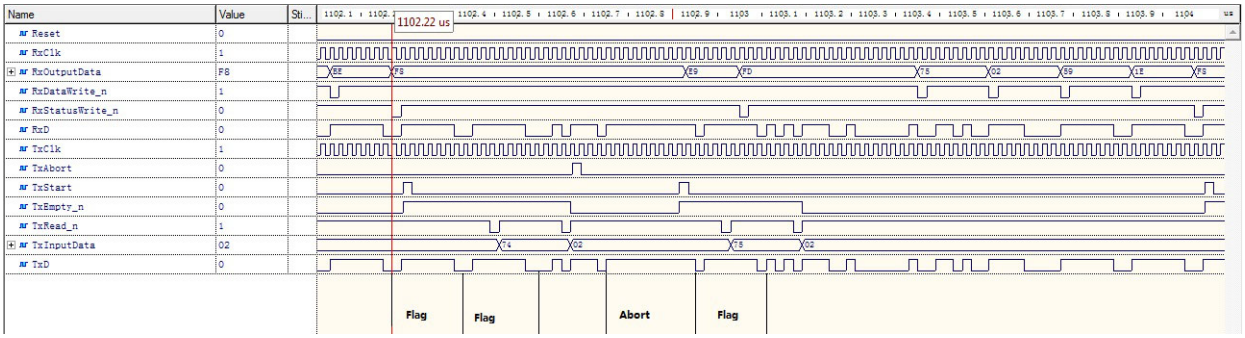


Transmission Abort

Once the host processor begins transmission through the assertion of TxStart, the controller will assert TxRead_n many times to obtain the transmitting octets until a low TxEmpty_n is sampled, indicating that this is the last octet of the frame. Theoretically, the host processor doesn't need to do anything after it asserts the TxStart. However, if the host processor needs to terminate the transmission before the whole HDLC frame is transmitted, the TxAbort signal can be used. The TxAbort signal must be asserted for at least one TxClk clock period. Once asserted, the transmission will be abandoned and an HDLC abort sequence will be transmitted followed by the HDLC flag.

Figure 12 shows both the waveforms of the transmitter and receiver when the abort is issued. The transmitter TxD signal is connected to the receiver RxD signal and both the transmitter and receiver are running the same clock. The following example simulates a single-channel rather than a multi-channel transmitter.

Figure 12. Single-Channel Transmission Abort Timing



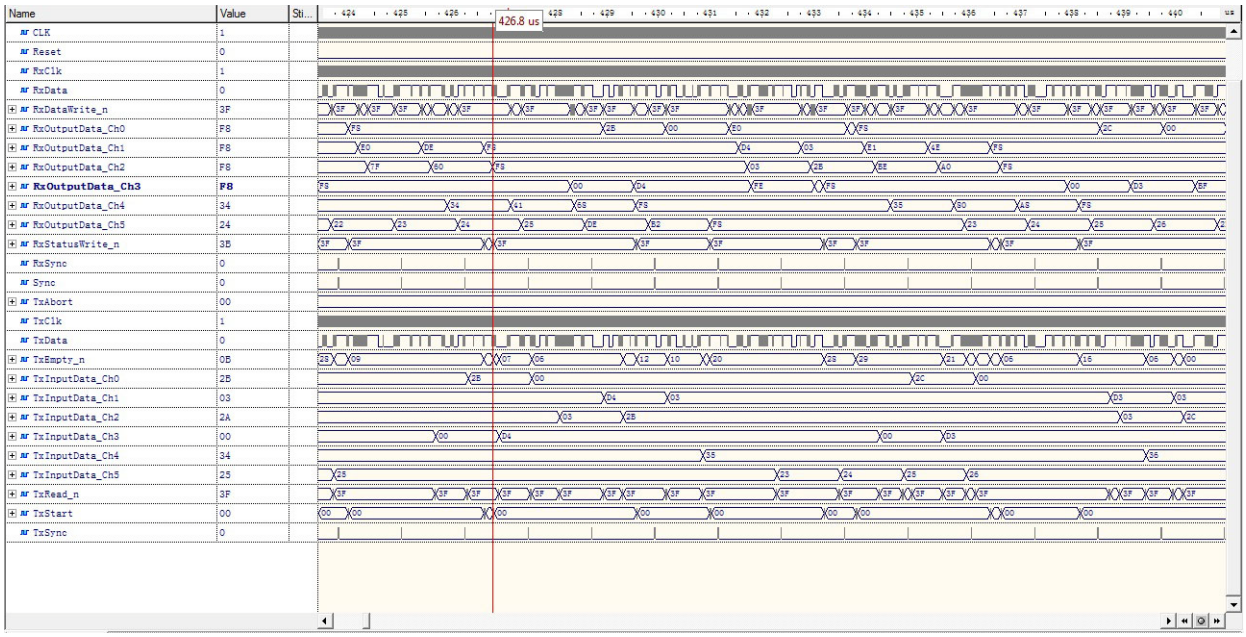
Controller Channel Configuration

This design is a multi-channel HDLC controller. The multi-channel design is a multi-channel HDLC controller. The design is divided into several modules with clean-cut functions. It is very easy to obtain a single-channel HDLC controller by instantiating only the RECEIVE and TRANSMIT modules on the top level. Any combination of receiver and transmitter channels can be obtained by proper instantiations of the RECEIVE and TRANSMIT modules.

When targeting MachXO, LatticeXP2 and LatticeECP2/M devices, a 6-channel HDLC controller, including both receiver and transmitter functions, can be implemented easily into these devices.

Figure 13 shows the timing simulation waveforms of a 6-channel HDLC controller. For depiction purposes, the PCM high way TxData output is connected back to the PCM high way RxData input and both the TxClk and the RxClk are running at the same clock.

Figure 13. Multi-channel HDLC Transmitting and Receiving Timing



Pin Descriptions

Name	Type	Description
RxClk	I	**Receive Serial Clock**: This signal provides the clock for the RECEIVE modules and the RX_DEMUX module in this design.
RxData	I	**Receive Serial Data**: Serial data is received at this PCM input port.
RxSync	I	**Receive Serial Sync**: This active high signal provides the synchronization reference for the receiving PCM frame. It can be a bit asserted simultaneously with the last data bit of the PCM frame or be a dedicated bit separated from the data bits of the PCM frame.
TxClk	I	**Transmit Serial Clock**: This signal provides the clock for the TRANSMIT modules and the Tx_MUX module in this design.
TxData	O	**Transmit Serial Data**: Serial data is transmitted through this PCM output port.
TxSync	I	**Transmit Serial Sync**: This active high signal provides the synchronization reference for the transmitting PCM frame. It can be a bit asserted simultaneously with the last data bit of the PCM frame or be a dedicated bit separated from the data bits of the PCM frame.
RxOutputData[N:0][7:0] (for multi-channel) RxOutputData_0(7:0) (for single-channel)	O	**Receiver Data Output**: This is an 8-bit data bus. One for each HDLC channel. The value present on this bus could be either frame data or frame status depending on the waveforms of RxDataWrite_n and RxStatusWrite_n.
RxDataWrite_n[N:0] (for multi-channel) RxDataWrite_n (for single-channel)	O	**Receive Data Write Enable**: This active low output indicates that the value currently present on bus RxOutputData is the HDLC frame data or FCS.
RxStatusWrite_n[N:0] (for multi-channel) RxStatusWrite_n (for single-channel)	O	**Receive Status Write Enable**: This active low output indicates that the value currently present on bus RxOutputData is the HDLC frame status.
TxInputData[N:0][7:0] (for multi-channel) TxInputData_0(7:0) (for single-channel)	I	**Transmitter Data Input**: This is an 8-bit data bus. One for each HDLC channel. The HDLC frame octets to be transmitted are read into the controller through this bus.
TxRead_n[N:0] (for multi-channel) TxRead_n (for single-channel)	O	**Transmit Data Read Enable**: This active low output, one for each HDLC channel, indicates to the external memory module that the controller is going to read the transmission octet in through TxInputData at the next TxClk rising edge.
TxStart[N:0] (for multi-channel) TsStart (for single-channel)	I	**Transmit Start**: This is an active high input, one for each channel. TxStart indicates to the controller that the transmission data of the HDLC frame is ready and the transmitting process can be started. This signal needs to be asserted for at least one TxClk clock period and be negated before the HDLC frame is completely transmitted. Once active, the HDLC transmitter will start asserting TxRead_n to read the octets from the external memory module.
TxAbort[N:0] (for multi-channel) TxAbort (for single-channel)	I	**Transmit Frame Abort**: This is an active high input, one for each channel. TxAbort indicates to the controller that the host wants to abort the transmission of the current HDLC frame. This signal needs to be asserted for at least one TxClk clock period. TxAbort can also be used for the idle assertion by asserting it for more than 15 TxClk clock periods.
TxEmpty_n[N:0] (for multi-channel) TxEmpty_n (for single channel)	I	**Transmit Data Empty**: This is an active low input, one for each channel. TxEmpty_n indicates that the value currently present on the TxInputData bus is the last octet of the HDLC frame. It will be sampled together with TxInputData.
Reset	I	**Master Reset**: This active high reset input will reset all internal registers in the design to their initial state.

Parameter

This reference design provides the following user-programmable parameter.

Name	Description
NumOfChannel	**Number of HDLC Channels**: This defines the total number of HDLC channels.

Implementation

Device Family	Language	Number of Channels	Utilization (LUTS)	Registers	Slices	f_{MAX} (MHz) Tx Clk/Rx Clk
LatticeECP2™ [1]	VHDL	Single channel	104	144	95	>150
		Multi-channel (6)	729	890	629	>150
	Verilog	Single channel	104	144	95	>150
		Multi-channel (6)	734	894	637	>150
LatticeECP2M™ [2]	VHDL	Single channel	104	144	95	>150
		Multi-channel (6)	729	890	629	>150
	Verilog	Single channel	104	144	95	>150
		Multi-channel (6)	734	894	637	>150
LatticeXP2™ [3]	VHDL	Single channel	104	144	95	>150
		Multi-channel (6)	729	890	629	>150
	Verilog	Single channel	104	144	95	>150
		Multi-channel (6)	734	894	637	>150
MachXO™ [4]	VHDL	Single channel	100	144	79	>150
		Multi-channel (6)	674	883	486	>150
	Verilog	Single channel	100	144	79	>150
		Multi-channel (6)	674	883	486	>150

1. Performance and utilization characteristics are generated using LFE2-70E-5F672C, with Lattice ispLEVER® 8.0 SP1 software. When using this design in a different device, density, speed, or grade, performance and utilization may vary.
2. Performance and utilization characteristics are generated using LFE2M50E-5F484C, with Lattice ispLEVER 8.0 SP1 software. When using this design in a different device, density, speed, or grade, performance and utilization may vary.
3. Performance and utilization characteristics are generated using LFXP2-17E-5F484C, with Lattice ispLEVER 8.0 SP1 software. When using this design in a different device, density, speed, or grade, performance and utilization may vary.
4. Performance and utilization characteristics are generated using LCMXO2280C-5FT324C, with Lattice ispLEVER 8.0 SP1 software. When using this design in a different device, density, speed, or grade, performance and utilization may vary.

Technical Support Assistance

Hotline: 1-800-LATTICE (North America)
+1-503-268-8001 (Outside North America)
e-mail: techsupport@latticesemi.com
Internet: www.latticesemi.com

Revision History

Date	Version	Change Summary
September 2008	01.0	Initial release.
June 2010	01.1	Added RTL source code and Verilog support.