HDLC controller core

Jamil Khatib

April 9, 2001

Contents

1 List of authors and changes

Name	Changes	Date	Contact address
Jamil Khatib	Initial release	9-1-2001	khatib@ieee.org
Jamil Khatib	TX interface added, Spec improved	27-1-2001	khatib@ieee.org
Jamil Khatib	External FIFO buffer added	3-2-2001	khatib@ieee.org
Jamil Khatib	Registers and CPU interface added	8-2-2001	khatib@ieee.org
Jamil Khatib	Drop bit, TDM interface are added	9-2-2001	khatib@ieee.org
Jamil Khatib	More design descriptions added	2-4-2001	khatib@ieee.org
Jamil Khatib	FIFO buffers calculations added	9-4-2001	khatib@ieee.org

2 Project Definition

2.1 Introduction

HDLC protocol is used as a data link of most of the current communication systems like ISDN, Frame Relay etc. HDLC is a family of protocols that varies in address size, control field, FCS and no. of data bits.

2.2 Objectives

The aim of this project is to develop the basic HDLC functionalities to be used by many communication systems.

3 Specifications

3.1 System Features Specification

1. Synchronous operation
2. 8 bit parallel back-end interface
3. Use external RX and TX clocks
4. Start and end of frame pattern generation
5. Start and end of frame pattern checking
6. Idle pattern generation and detection (all ones)
7. Zero insertion and removal for transparent transmission.
8. Abort pattern generation and checking (7 ones)
9. Address insertion and detection by software
10. CRC generation and checking (CRC-16 or CRC-32 can be used which is configurale at the code top level)
11. FIFO buffers and synchronization (External)
12. Byte aligned data (if data is not aligned to 8-bits error signal is reported to the backend interface)
13. Q.921, LAPD and LAPB compliant.
14. The core should not have internal configuration registers or counters, instead it provides all the signals to implement external registers.
15. There is No limit on the Maximum frame size as long as the backend can read and write data (depends on the external FIFO size)

16. Bus connection is not supported directly (TxEN and RxEN pins can be used for that reason)

17. Retransmission is not supported when there is collision in the Bus connection mode.

18. This controller is used for low speed application only (relative to the backend bus).

19. Supports connection to TDM core via backend interface and software control for time slot selection and control (signaling ,etc.) generation.

20. Backend interface uses the Wishbone bus interface which can be connected directly to the system or via FIFO buffer.

21. Optional External FIFO buffers, configuration and status registers.

22. The core will be made of two levels of hierarchies, the basic functionality and the Optional interfaces and buffers.

3.2 External Interfaces

3.2.1 Receive Channel

Signal name	Direction	Description
Control interface		
Rst	Input	System asynchronous reset(active low)
Serial Interface		
RxClk	Input	Receive Clock
Rx	Input	Receive Data
RxEn	Input	RX enable (active high)
Back-end Interface		
RxD[7:0]	Output	Receive data bus
ValidFrame	Output	Valid Frame indication during all frame bytes transfer
FrameErr	Output	Error in the received data (lost bits)
Aborted	Output	Aborted Frame
Read	Input	Read byte
Ready	Output	Valid data exists

3.2.2 Back-end interface mapping to Wishbone SoC bus

The HDLC receive backend interface can be used as a slave core or master according to the below mapping. The core supports SINGLE READ Cycle only using 8-bit data bus without address lines. The choice between master and slave is left for the system integrator and must do the configuration and glue logic as defined in the tables.

Signal Name	Wishbone signal
Master Configuration connected to FIFO	
RxClk	CLK_I
Rst	not RST_I
RxD[7:0]	DAT_O(7:0)
ValidFrame	STB_O
ValidFrame	CYC_O
ReadByte	ACK_I and not RTY_I
Ready	WE_O
FrameERR	TAG0_O
Aborted	TAG1_O
Slave FIFO(two-clock domain FIFO)	
Data[7:0]	DAT_I(7:0)
Chip Select	STB_I
STB_I and not FullFlag	ACK_O
FullFlag	RTY_O
Write	WE_I
Slave Configuration	
RxClk	CLK_I
Rst	not RST_I
RxD[7:0]	DAT_O(7:0)
ValidFrame	TAG0_O
ReadByte	not WE_I
Ready	not RTY_O
STB_I and not WR_I	ACK_O
FrameERR	TAG1_O
Aborted	TAG2_O

3.2.3 Transmit Channel

Signal name	Direction	Description
Control interface		
Rst	Input	System asynchronous reset(active low)
Serial Interface		
TxClk	Input	Transmit Clock
Tx	Output	Transmit Data
TxEn	Input	TX enable (active high)
Back-end Interface		
TxD[7:0]	Input	Transmit data bus
ValidFrame	Input	Valid Frame indication during all frame bytes transfer
AbortedTrans	Output	Error in the transmitted data (Abort pattern was generated)
AbortFrame	Input	Abort Frame
Write	Input	Write byte
Ready	Output	Can accept new data

3.2.4 Back-end interface mapping to Wishbone SoC bus

The HDLC receive backend interface can be used as a slave core or master according to the below mapping. The core supports SINGLE WRITE Cycle only using 8-bit data bus without address lines. The choice between master and slave is left for the system integrator and must do the configuration and glue logic as defined in the tables.

Signal Name	Wishbone signal
Master Configuration connected to FIFO	
TxClk	CLK_I
Rst	not RST_I
TxD[7:0]	DAT_I(7:0)
Write	ACK_I and not RTY_I
Ready	not WE_O
AbortedTrans	TAG0_O
ValidFrame	TAG1_I
AbortFrame	TAG0_I
Always Active	CYC_O
Always Active	STB_O
Slave FIFO(two-clock domain FIFO)	
Data[7:0]	DAT_I(7:0)
EmptyFlag	RTY_O
Read	WE_I
WE_I and not EmptyFlag	ACK_O
ChipSelect	STB_I
Slave Configuration	
TxClk	CLK_I
Rst	not RST_I
TxD[7:0]	DAT_I(7:0)
ValidFrame	STB_I
Write	WE_I
Ready	not RTY_O
STB_I and WR_I	ACK_O
AbortFrame	TAG0_I
AbortedTrans	TAG0_O

3.2.5 CPU interface

This interface is used when the FIFO and registers are included in the Core. This interface is compatible to WishBone slave bus interface that supports single read/write cycles and block cycles. The interface supports the following wishbone signals.

Signal	Note
RST_I	Reset
CLK_I	Clock
ADR_I(2:0)	3-bit address line
DAT_O(7:0)	8-bit receive data
DAT_I(7:0)	8-bit transmit data
WE_I	Read/write
STB_I	Strobe
ACK_O	Acknowledge
CYC_I	Cycle
TAG0_O	TxDone interrupt
TAG1_O	RxReady interrupt

4 Design description

4.1 Receive Channel

4.1.1 Design notes

Receive channel provides interface to the backend via a simple handshake protocol that can be used to connect the controller to either a shared memory or FIFO buffer. This protocol uses the hand shack protocol of the Wishbone SoC bus.

Receive channel supports only 8-bits aligned data. Each frame starts with a starting flag (01111110) and ends with starting flag (01111110). Since the receipt ion is synchronous only, the channel uses the external clock and a byte must be read from the channel within the first 7 clock pulses after the ready signal is asserted. If no data is read during this period (while ValidFrame signal is active) FrameErr is signaled reported to the backend as long the ValidFrame is active. FrameErr is signaled also when non 8-bit aligned data is received and when FCS error is found.

4.1.2 Timing

4.2 Transmit Channel

Transmit channel provides interface to the backend via a simple handshake protocol that can be used to connect the controller to either a shared memory or FIFO buffer. This protocol uses the handshack protocol of the Wishbone SoC bus.

Transmit channel supports only 8-bits aligned data. Each frame starts with a starting flag (01111110) and ends with starting flag (01111110). Since the transmission is synchronous only, the channel uses the external clock and a byte must be written to the channel within the first 7 clock pulses after the ready signal is asserted. If no data is inserted during this period (while

ValidFrame signal is active) abort pattern is transmitted and reported to the backend via AboredTrans signal as long the ValidFrame is active.

4.2.1 Design notes

4.2.2 Timing

The channel starts accepting data after asserting the ValidFrame signal. This signal can control no of idle pattern bits (e.g. if this signal is de-asserted for 8 bits only a single Idle pattern (8 ones) is inserted). Valid Frame signal must be asserted for 8 clocks after any valid write operation.

4.3 External FIFO and registers

The controller has optional external FIFO buffers, one for data to be transmitted and one for data to be received. Status and control registers are available to control these FIFOs. These two blocks (FIFOs and registers) are built around the HDLC controller core which make them optional if the core is to be used in different kind of applications. The current implementation supports the following configuration: The size of the Transmit and receive FIFOs is (8×128) bits which enables 128 maximum HDLC frame size.

The transmit buffer is used to prevent underflow while transmitting bytes to the line. All bytes will be available once the transmit is enabled. The Receive buffer is used to provide data burst transfer to the Back end interface which prevents the back end from reading each byte alone. The FIFO size is suitable for operating frequencies 2.048MHz on the serial interface and 50 MHz on the back end interface. Other frequencies can operate if the delay between HDLC frames is less than the delay needed for the back end to empty the internal FIFO (the next calculations is an example to be applied for different frequencies)

7 bits (minimum bits between HDLC Frames) / 2.048MHz = 3.418 us

128 Bytes (Maximum frame size) / 50MHz = 2.56 us

These FIFOs are implemented on Single port memory. Two interrupt lines are used, one to signal transmission done and one to request transfer of received frame to memory. These interrupts are also reflected in Status registers to support polling mode for the controller.

4.4 Registers

All internal registers are 8-bit width.

4.4.1 Transmit

Tx Status and Control Register: Tx_SC Offset Address = 0x0

BIT	7	6	5	4	3	2	1	0
FIELD	N/A	N/A	FCSen	FIFOOverflow	Aborted	TxAbort	TxEnable	TxDone
RESET	0	0	0	0	0	0	0	0
R/W	RO	RO	WO	RO	RO	WO	WO	RO

Tx FIFO buffer register: Tx_Buffer Offset Address = 0x1

BIT	7-0
FIELD	Transmit Data byte
RESET	0x0
R/W	WO

4.4.2 Receive

Rx Status and Control Register: Rx_SC Offset Address = 0x2

BIT	7	6	5	4	3	2	1	0
FIELD	N/A	N/A	N/A	FIFOOverflow	Aborted	FrameError	Drop	RxReady
RESET	0	0	0	0	0	0	0	0
R/W	RO	RO	RO	RO	RO	RO	WO	RO

Rx FIFO buffer register: Rx_Buffer Offset Address = 0x3

BIT	7-0
FIELD	Received Data byte
RESET	0x0
R/W	RO

Rx Frame length: Rx_Len Offset Address = 0x4

BIT	7-0
FIELD	Frame Length
RESET	0x0
R/W	RO

4.5 Transmit Frame

- The CPU should check TxDone (in Tx status register 0x0) bit of it is '1' or wait for TxDone interrupt. TxDone bit is reset to '0' after the first write to Tx FIFO Buffer register (0x1).
- The CPU should write frame data bytes to Tx FIFO buffer register (0x1).
- After writing all data bytes to TX buffer register, the CPU should write '1' to TxEnable to enable data transmission to the line. After

writing to this bit no further write operation to Tx FIFO buffer register is allowed till TxDone is set (all writes will be ignored).

- It is optional for the CPU to check the status bits of Tx status register.

4.6 Receive Frame

- The controller sets RxReady bit in Rx Status and control register (0x2) and sets the TxReady interrupt line to indicate valid frame in internal buffer is available.
- It is recommended that the CPU read the Rx Status and control register (0x3).
- The CPU should read the Frame length register (0x4) to check the size of the frame. The value of this regiter is valid only after the RxReady bit is set and remains valid till the first read from the Data buffer.
- The CPU should read Rx FIFO buffer register (0x3) Frame length times to get all frame bytes. Performing extra reads (read from empty buffer) produces invalid data.
- If the CPU does not read all frame bytes as soon as possible the internal buffer will overflow and FIFOOverflow bit will be set and the current frame should be dropped. No further read operations should be attempted till RxReady bit is set again and RxReady interrupt is signaled indicating new available frame.
- The software can drop entire frame from the Receive FIFO buffer by writing 1 to drop bit in the status and control receive register (0x3). This is suitable for dropping bad frames (for any reason) or frames with incorrect addresses.

4.7 Connection to TDM controller

This controller can get/send data from/to TDM controller through software control. The software configures the TDM controller to select the channel. It adds/removes the address and control information fields of the HDLC frame. Then passes the data field between the two controllers through optional DMA transfer.

4.8 Clocks Synchronization

Since the core can operate in different clock domains (The serial line domain and the backend interface domain), all control signals pass through two flip flops to reduce the metastability. These Flip Flops are clocked with the same clock of the interface that read these signals.

4.9 Diagrams

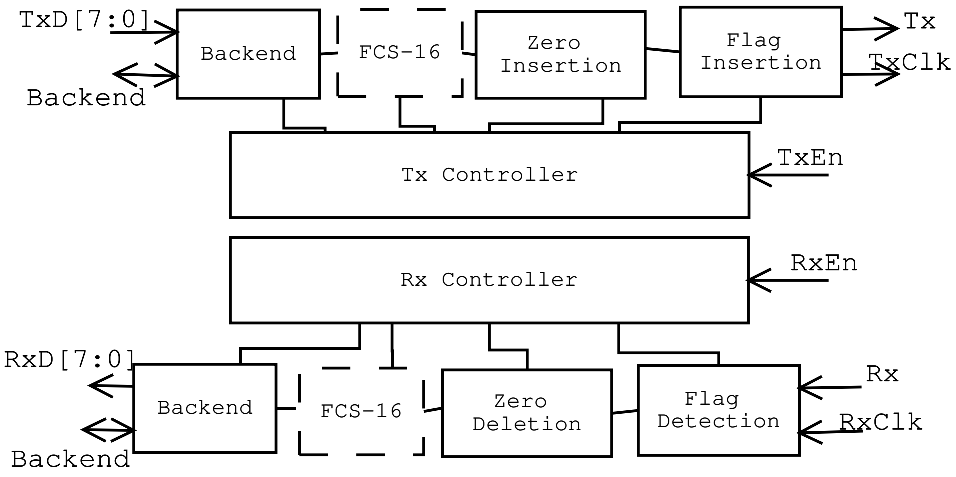

Figure 1: HDLC core

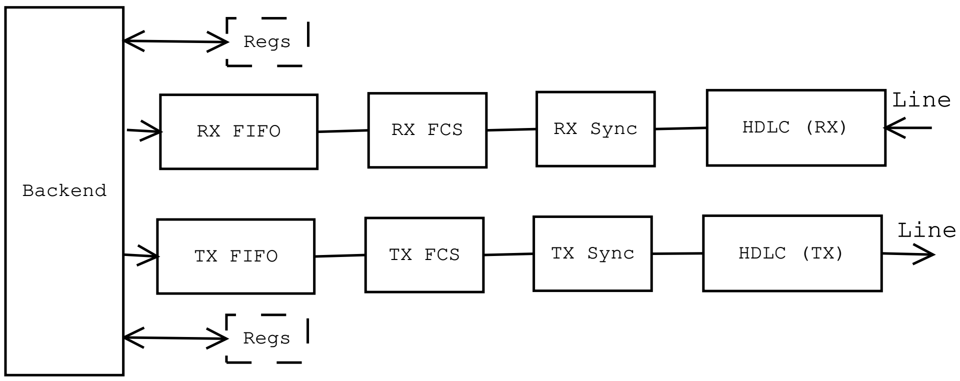

Figure 2: HDLC controller

5 Testing and verifications

Requirement	Test method	Validation method
Interface timing		
Functionality		

5.1 Simulation and Test benches

5.2 Verification techniques and algorithms

5.3 Test plans

6 Implementations

The design is implemented using the VHDL language. The design is divided into three main blocks, serial Receive channel, Serial Transmit channel and the Top blocks. The Receive and Transmit serial channels perform the HDLC functionality. The Top blocks perform the FCS calculation (Which is either FCS-16 or FCS-32), the frame buffering the interface with the back end system and the synchronization between the clocks. The FCS and Buffering can be changed by replacing the corresponding files.

6.1 Scripts, files and any other information

RX	
RxChannel.vhd	Top Rx Channel
Rxcont.vhd	Rx Controller
Zero_detect.vhd	Zero detect and serial to parallel
flag_detect.vhd	Flag detection
TX	
TxChannel.vhd	Top Tx channel
TXcont.vhd	Tx Controller
zero_ins.vhd	Zero insertion and parallel to serial
flag_ins.vhd	Flag insertion
Top	
TxBuff.vhd	Tx buffer
TxFCS.vhd	Tx FCS-16
TxSync.vhd	Tx synchronization
RxBuff.vhd	Rx buffer
RxFCS.vhd	Rx FCS-16
RxSync.vhd	Rx synchronization
WB_IF.vhd	WishBone interface
hdlc.vhd	Top HDLC controller

6.2 Design conventions and coding styles

7 Reviews and comments

8 References