

Implementation of a Single-Channel HDLC Controller on FPGA

Hichem Semira
Department of Electronics,
BadjiMokhtar
University,Annaba- B.P.12,
Algeria.

Mohamed Benouaret
Department of Electronics,
BadjiMokhtar
University,Annaba- B.P.12,
Algeria.

Saliha Harize
Department of Electronics,
BadjiMokhtar
University,Annaba- B.P.12,
Algeria.

ABSTRACT

HDLC are the high level data link control procedures established by ISO, They are widely used in digital communication and are the bases of many other data link control protocols. The objective of this paper is to implement a Single-Channel HDLC Controller on an Altera FPGA. All the modules such as the transmitter and the receiver are designed and implemented using VHDL programming language and illustrated with a detailed schema. The software tools used in this work include Altera Quartus II 8.1 and ModelSim Altera 6.1g. The target circuit is the Cyclone II EP2C35F672C6.

Keywords

HDLC, Data Link Control Layer, Altera FPGA, bit stuffing/unstuffing, CRC-16, CRC-32, Flag.

1. INTRODUCTION

To successfully transmit data over any network, a protocol is required to manage the flow of the transmitted data. HDLC (High Level Data Link Control) protocol is one of the most widely used protocol in the field of communication. HDLC is a classical bit-oriented protocol whose variants have been in use for decades in many applications. The HDLC is defined as International Standardization Organization's recommendations ISO/IEC 13239:2002 [1]. It provides a variety of options that the two communicating stations may be able to negotiate before data exchange. The main goal of the HDLC protocol is to achieve a reliable and efficient communication between two adjacent machines at the data link layer.

HDLC procedures are commonly performed by ASIC (Application Specific Integrated Circuit) devices or software programming. ASIC devices are simple, but they lack the flexibility in different applications and it is difficult for ASIC devices to meet all the requirements of various versions of the HDLC protocol. Software programming of HDLC procedures is flexible and can be used in many different HDLC applications by simple modification. However, the programs are heavy consumer of the processor resources and time when running. As the programmable logic devices are widely in use, HDLC procedures can be implemented in FPGA. By adopting hardware processing technology FPGA devices can be programmed repeatedly. So, using FPGA to replace ASIC devices for HDLC implementation is a proper choice. Therefore, for faster implementation, several researchers have designed and developed HDLC controllers according to the requirements of the basic communication system using FPGA [2-13].

This paper presents the design of a FPGA-based HDLC protocol controller The VHDL modeling of a single-channel

HDLC protocol transmitter/ receiver and its implementation using ALTERA FPGA Cyclone II EP2C35F672C6 as the target technology are discussed. In the design process the description of the controller as indicated in [14] with some modifications is followed. Across the paper, HDLC controller components are described with detailed block diagrams to assist in understanding the concepts of their modeling in VHDL.

2. THE PRINCIPLE OF HDLC PROTOCOL

HDLC uses the term "frame" to indicate an entity of data (or a protocol data unit) transmitted. Figure 1 below is a graphical representation of an HDLC frame with an information field [15].

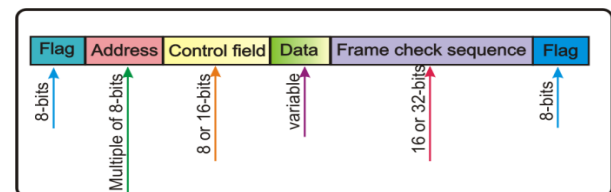


Figure1: HDLC Frame structure

- 1) Flag: An HDLC frame is composed of the flag and the serial transmission data. This figure does not include the bits inserted for transparency. Transparency (Zero stuffing) is performed on the HDLC packets whenever five contiguous « 1 » bits are transmitted. The fields are transmitted from left to right, least significant bit first. Each frame starts and ends with a flag sequence "01111110" (0x7E) used for frame. Synchronization, interframe time filling shall be accomplished by transmitting contiguous flags between frames. In the case of a continuous transmission, the two consecutive frames may share the same flag. The closing flag of the first frame is used as the opening flag of the second one.
- 2) Insertion and deletion of "0": In order to make the flag unique to the whole bit stream, a zero insertion and deletion technique is applied to the no flag fields. For data transmission, whenever there are five consecutive 1's being transmitted, an additional redundant zero bit will be inserted immediately 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".

- 3) Address: Address field is used to identify the addresses of the receivers.
- 4) The control field indicates the type of commands or responses, and contains sequence numbers, where appropriate. The control field shall be used:
 - a) To convey a command to the addressed data station(s) to perform a particular operation, or
 - b) To communicate a response to such a command from the addressed data station.
- 5) Information (variable length): Information may be any sequence of bits.
- 6) Frame Check Sequence (FCS): The FCS is calculated according to the CRC error detecting scheme from the serial bit stream of the address, control, and information fields excluding the opening flag. It is usually a 16-bit or 32-bit pattern used for checking the frame data integrity. The length of the designated portion of the frame being protected by the FCS checking mechanism is determined by negotiation or known by a priori knowledge. The most commonly CRC polynomials are:

$$\text{CRC-16} = x^{16} + x^{15} + x^2 + 1$$

$$\text{CRC-CCITT} = x^{16} + x^{12} + x^5 + 1$$

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

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. At anytime, the transmission of a frame can be aborted by sending the Abort flag, which is (01111111). The receipt of seven contiguous "1" bits shall be interpreted as an abort flag and the receiving data station shall ignore the frame. A data link channel is in an idle state when a continuous "1" state is detected that has persisted for at least 8 bit times. Detection of the idle state at the data link layer shall

be considered to indicate that the remote data station has relinquished its right to continue transmission.

3. THE DESIGN AND IMPLEMENTATION OF HDLC

The focus of the HDLC controller development is on full-duplex communication application. The design of HDLC communication module is implemented using the VHDL language. It is divided into two main blocks which are the HDLC transmitter and the HDLC receiver.

3.1 The transmitter module

Before the transmission starts, the data needs to be stored in advance in an external memory such as FIFOs. The transmitter module includes a parallel-to-serial conversion unit to convert the data issued from a memory, a FCS generation unit for CRC check implemented in parallel [16], a "0" inserting unit; a flag generation unit and a Tx_control unit which is responsible for generating all the necessary internal control signals required by the different units. All units work synchronously on the rising edges of the Txclk clock.

As shown in Figure 2, the initial state of the state machine is TxReady where the Txreset signal is high. In this state all the Flip-Flops are initialized and the transmitter module sends an idle sequence. If the FIFO is not empty, which means there is data to be sent, the host processor asserts the TxStart signal for one Txclk clock and the state machine jumps into the TxSynchro state to send the FLAG as explained in Figure 3. Note that this module functions as a multiplexer to generate the sequences Idle, Flag and Abort based on shift registers, and it is controlled by the FLAGGEN bus signal.

The Tx_control module is notified by the Start_FLAG signal generated by the Flag generator module to load the data from FIFO. This signal is used to trigger a counter to manage and maintain synchronization between the signals Txload, Load_Data and FLAGGEN.

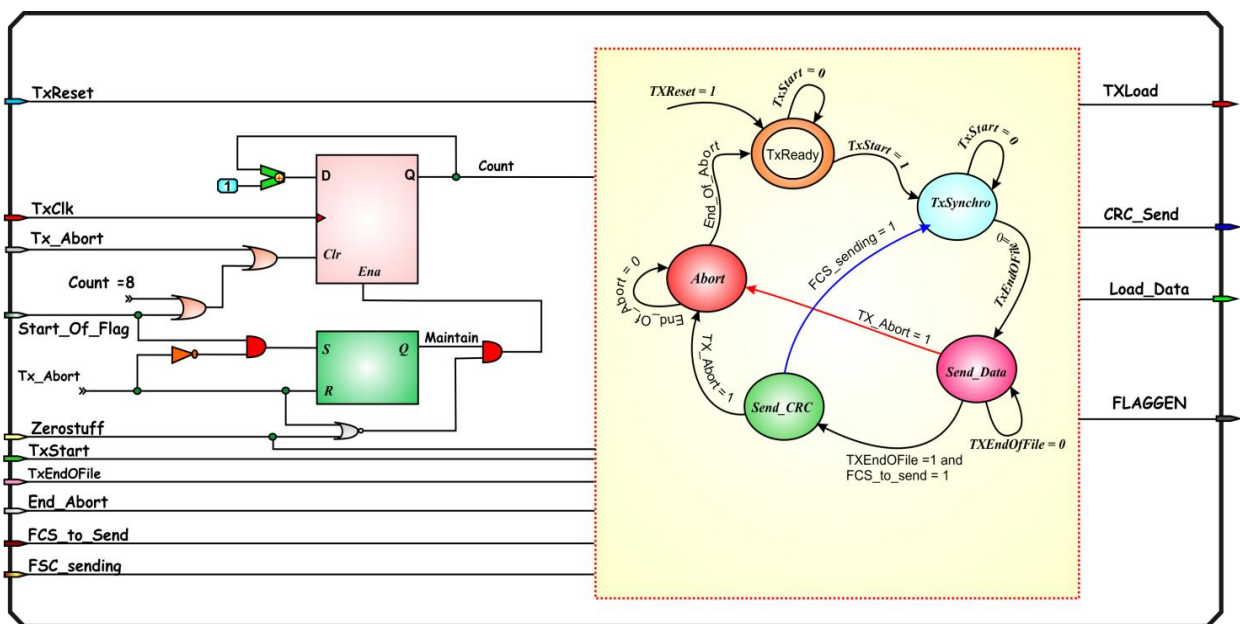


Figure 2: Tx_Control (Finite State Machine (FSM)) Model

Data to be transmitted is loaded into the buffer on the rising edge of Txclk when the Txload signal is asserted for one Txclk clock. To latch the data bus from the buffer (Reg_in) into the Latch_Buffer as indicated in Figure 4, the Tx_control module uses the Load_Data signal until the data is transmitted. At the same time, the CRC generator calculates

the CRC codes for the latched data in parallel for one TxCLKclock. Before the first byte is completely shifted out through the output line, the second Tx_load will be asserted to get the second byte and so on. Figure 4 clearly shows the different registers and counters used for parallel-to-serial conversion and CRC computing.

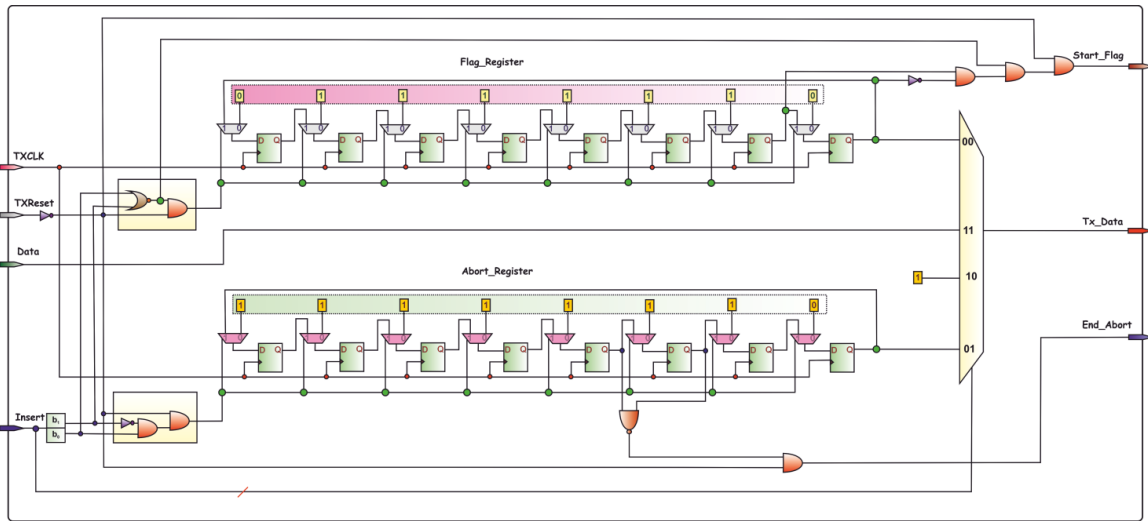


Figure 3: Idle, Flag and Abort sequences generator Module

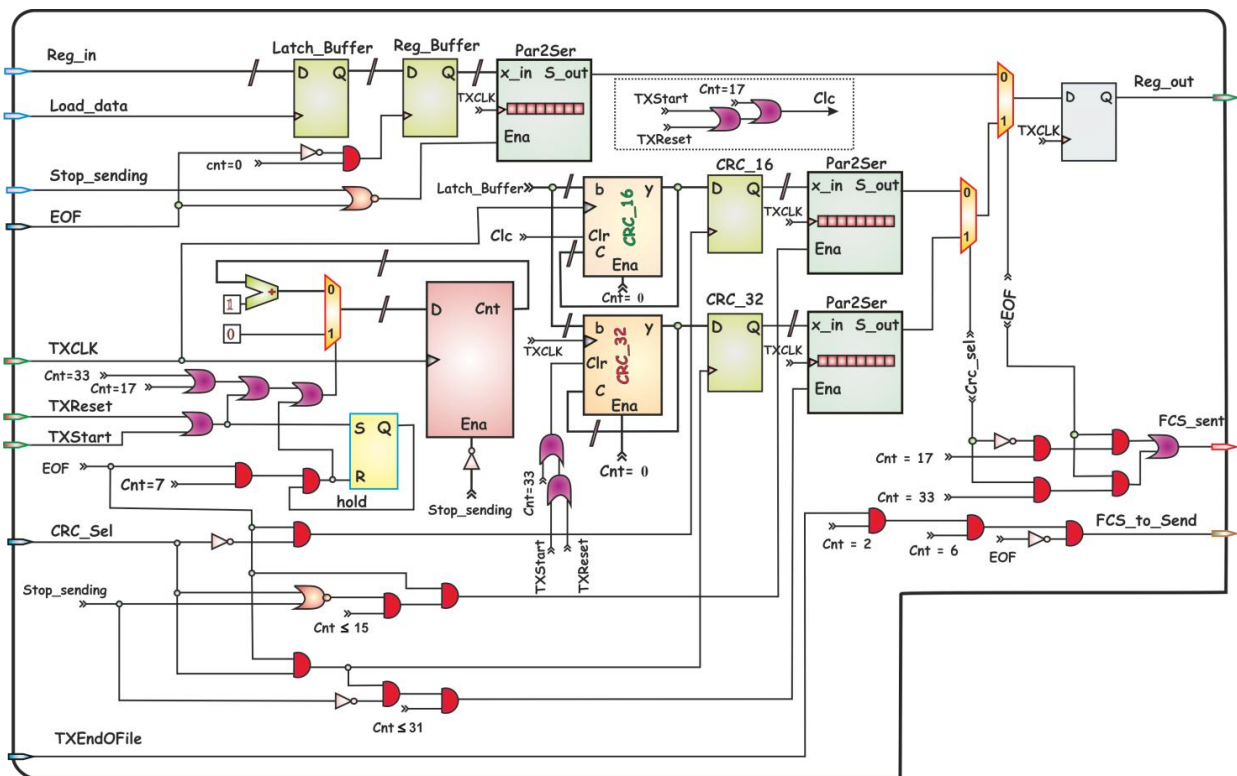


Figure4: Parallel-to-serial conversion and FCS generation module

The different logic circuits used to ensure a sequential binary lossless transmission are also shown. As long as the FIFO is not empty the CPU keeps the signal TxEndOfFile not asserted. The latched data will be shifted bit by bit through the Zero insertion module and the FLAG Generation module to the TxData output. The Zero insertion module is shown in Figure 5. The aim is to introduce a 0 after five consecutive 1 detected. Moreover the module generates a stop_reading

signal to stop the treatments in all the other modules in order to avoid losing data during the insertion of a zero. This task is successfully completed using simple shift registers.

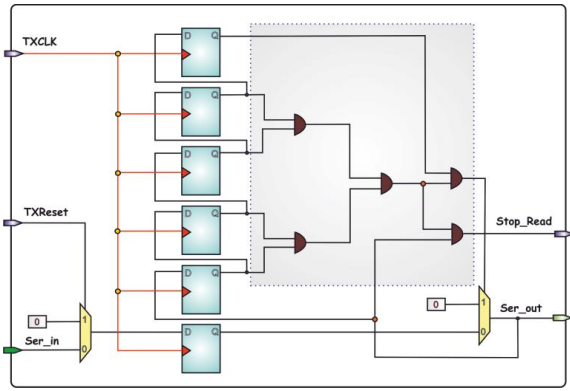


Figure 5: Zero insertion module

When loading the Data bus, the signal TxEndOfFile will be examined as well. If it is asserted with the signal fcs_to_send (see Figure 4) which is used to ensure the transmission of the last bit of data, the state machine switches to the Send_CRC state and the byte being loaded into the buffer will be considered as the last one of the current transmission frame.

After this last byte has been loaded into the latch_buffer and shifted out, the signal EOF(Crc_send signal from the

Tx_module) is examined. If it is asserted, the FCS will be shifted out bit by bit through the Zero insertion to the TxDataoutputTheTx_control module is informed by the end of the transmission of the FCSv by activating the signal fcs_sent to jump into the TxSychro state to generate a flag and the cycle repeats for a new frame. The assertion of the signal Fcs_sent depends on the choice of the size of CRC by examination of the value of crc_sel which is 0 for CRC-16 or 1 for CRC-32.

If a Tx_Abort command is received, the state machine jumps into the Abort state and cancels the current frame sending by asserting an Abort sequence 0xFE. Once the abort sequence has been transmitted, the FLAG Generation module asserts the signal End_Abort to switch to the TxReady state. In this case the Transmitter module keeps sending an idle sequence until the assertion of the signal TxStart.

The simulation results using ModelSim Altera 6.1g of the functional timing waveforms of the transmitter module are shown in Figure 6. The transmission simulation has been carried out using an arbitrary sequence: "FE02". It is clear in this figure that the output of this module is a binary sequence delimited by two FLAGS.

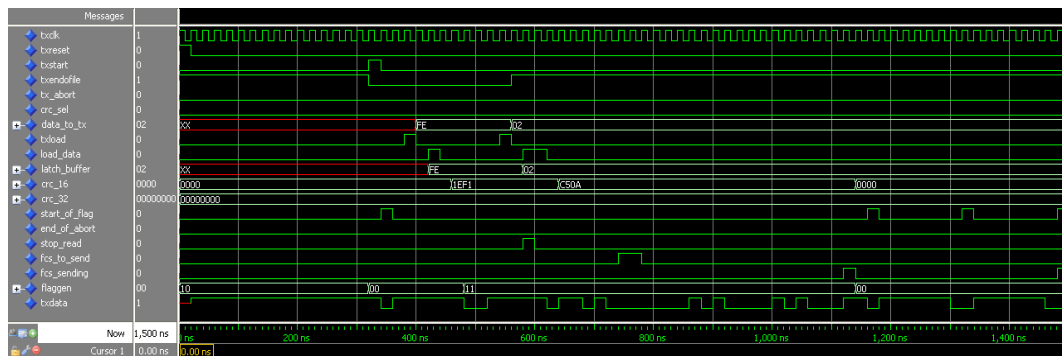


Figure6: The simulation waveforms of the transmitter module

It is clear in this figure that the output of this module is a binary sequence delimited by two FLAGS.

Using QUARTUS II software package provided by Altera, VHDL modules were written for each entity. Figure 7 shows the top-level schematic design for the implemented transmitter on Cyclone II EP2C35F672C6 FPGA after successful compilation. The resources used by the units of the transmitter module are listed in table1.

Table 1. Resources used by the units of the transmitter module.

Module	LogicCells	Dedicated LogicRegisters	Pins	LUT - Only LCs
Tx_control: U1	90	37	0	53
Tx_buffer_and_CRC_computation: U2	241	85	0	154
Zero_insert:U3	3	6	0	3
Flag_gen: U4	10	19	0	2
Transmitter Module (Total)	344	147	16	212
Utilization	1%(/33216)	< 1%(/33216)	3%(/475)	-
Total thermal power Dissipation(mw)	111.73			

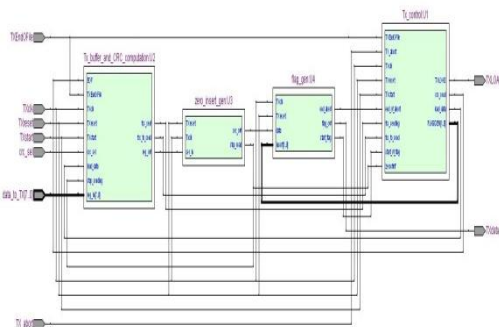


Figure 7: Top-level design of the implemented transmitter

The single channel HDLC transmitter is found to be running at a frequency of 41.61MHz (24.03ns) with a total equivalent cell count of 344.

3.2 TheReceive Module

The receivermodules complete the frame synchronization byte "7E" identification and Abort detection, bits unstuffing, serial-parallel conversion, and CRC checking and comparing. In the receiving module, as shown in Figure 8, the received data RX_Data is shifted at the clock edge of receiving RXCLK and placed in an 8-bit register (Reg_buffer) to check the presence of an eventual flag or an abort sequence.

The detection of the special sequences and the identification of data are controlled by a finite state machine as illustrated in figure 9.

The Receiver module is in RX_Idle state when the system is reset, and moves into RX_synchro state when the frame synchronization byte "7E" is identified. In this state, if a byte not equal to "7E" has been detected, then the last flag is recognized as an opening flag and the receive state machine

enters the RX_receiving_Data state. The receiver begins to collect the incoming frame, but continues to monitor for a closing flag or an eventual aborting and proceeds to the following tasks:

- Assert the length_packet signal to enable the '0' removing, the CRC computation and the bit counter to check for if the frame length is not equal to a multiple of bytes. The Bit counting and the CRC checking are used for error detection in the received HDLC frame.
- Once the length_packet is enabled, the zero removing unit will keep track of the incoming bit stream and removes the 0 bit after five consecutive 1 bits (remove redundant "0"). Figure 8 clearly shows the mechanism used to remove the inserted zero. This mechanism is based on a modulo 5 counter and a comparator.

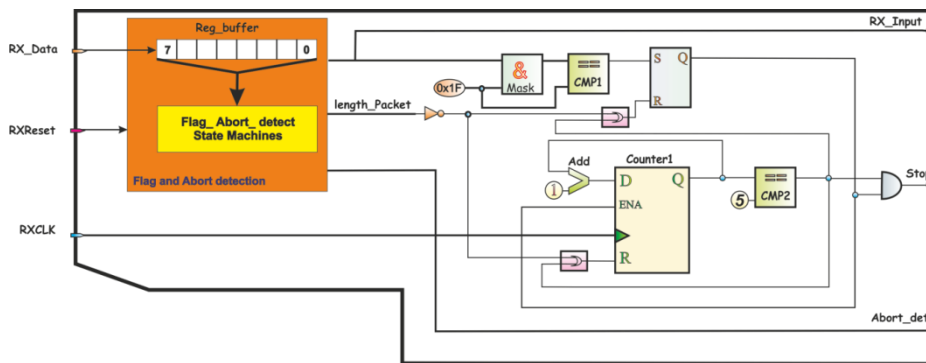


Figure 8: Idle, Flag and Abort sequences detector and bits unstuffing Module

- The serial data received is shifted into the R_buffer to remove the padded zero as shown in figure 10. Once the shift register is full, the CRC code is generated and the available 8-bit data is placed on the RX_data_out bus.
- Then the receiver will inform the host processor to store the 8-bit data in an external receive buffer by asserting the RX_Data_valid signal for one RXCLK clock. If this 8-bit

data is the first byte captured the receiver informs the user that the current received data is the first byte (start of frame) by enabling the SOPacket signal pulse for one Rxclk clock. To assure the assertion of the SOPacket signal in the heading of the frame, a driver signal named Start has been used as illustrated in figure 10.

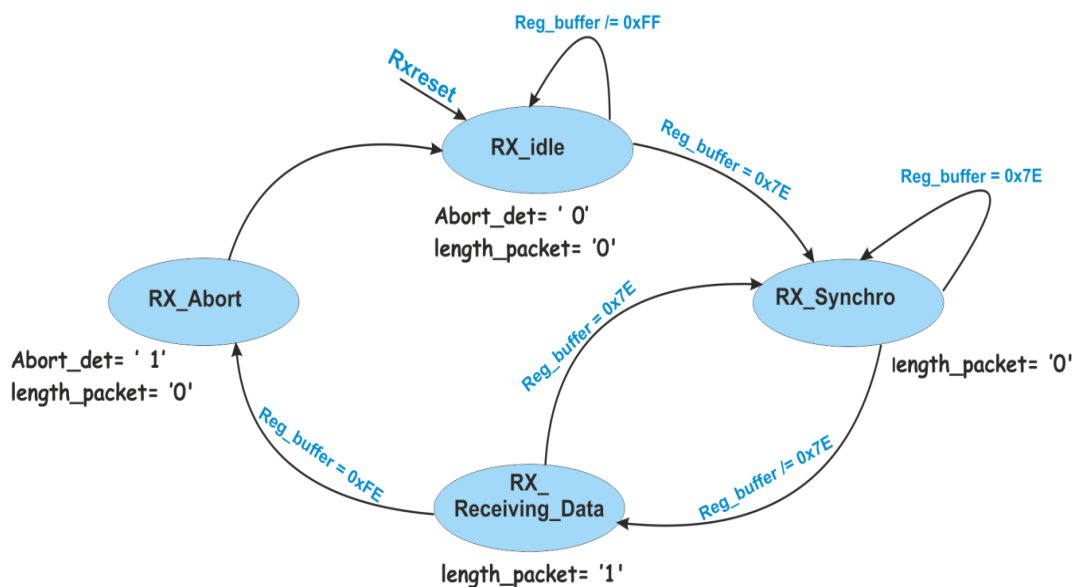


Figure 9: Flag and Abort detection FSM

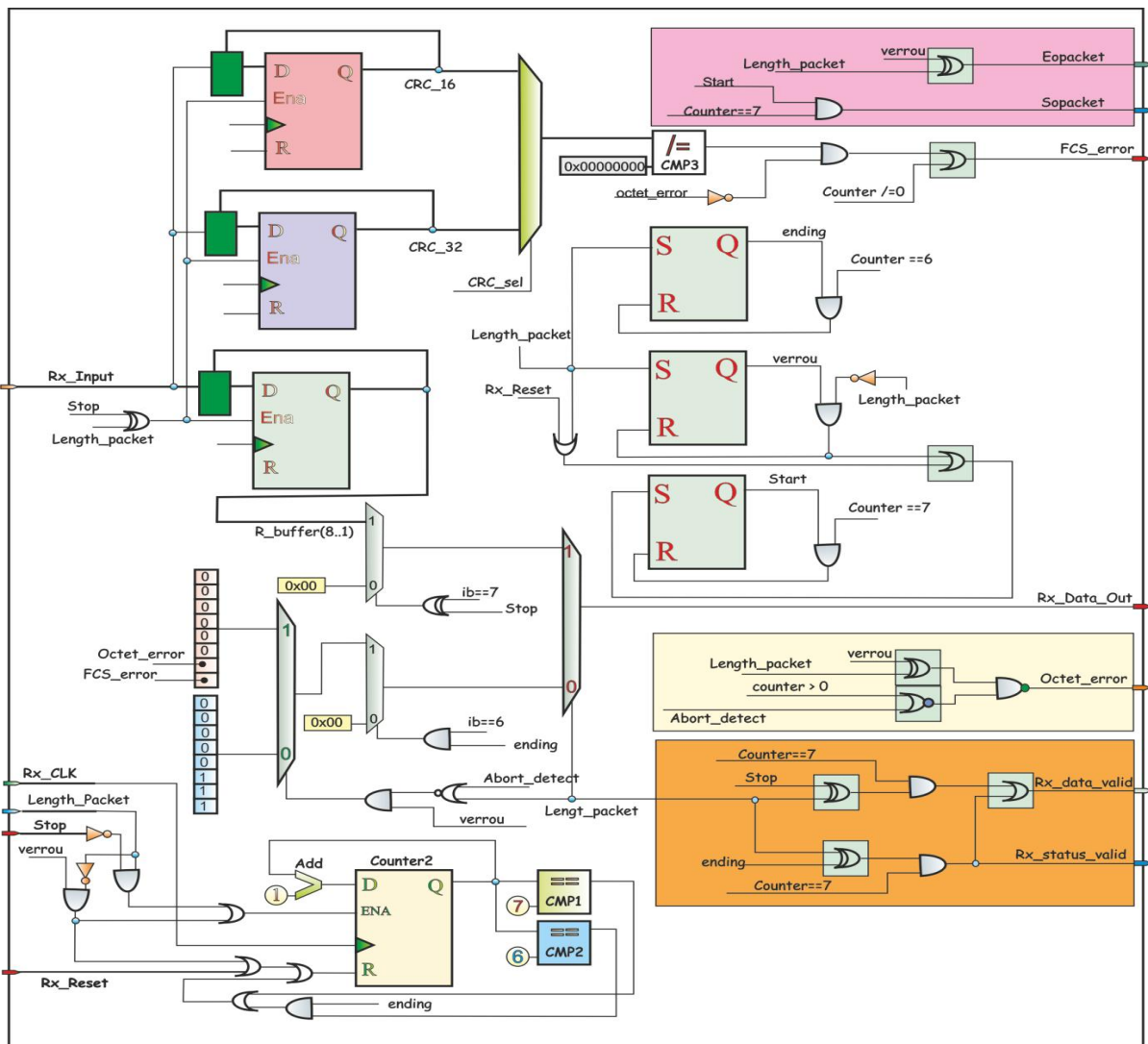


Figure 10: CRC checking and error frame detection Module

Once the closing Flag has been captured the RX_receiving_Data state is changed to the RX_synchro state and another operation for finding the flag is initiated. In this state the receiver performs the following actions:

- The length_packet signal is disabled to stop the CRC computation and bit counting.
- The receiver generates an active high pulse denoted by EOPacket to inform the user that this byte is the last one. Like the SOPacket signal the EOPacket is controlled by the verrou signal.
- After eight RXCLK clock, the receiver will present the status byte to the RX_data_out bus. The RX_status_valid will be asserted to inform the host processor that the value of the RX_data_out bus is the status and not the data. Note that the assertion of RX_status_valid coincides with that of RX_Data_valid signal. To ensure its validation after 8 Rxclk clock at the end of the frame an ending signal is used. This status byte will be reported after the entire HDLC frame has been received or the abort has been detected and is presented as follows [14]:

Bit7-Bit3	Bit2	Bit1	Bit0
-----------	------	------	------

Bit7-Bit3: Reserved

Bit2: Abort Detected

Bit1: Octet Error

Bit0: CRC Error

Note that during the detection of the abort sequence RX_receiving_Data state directly switches to RX_Abort state. The receiver will then inform the user that an error has occurred by asserting the status byte and move to the RX_idle state to look for another valid frame.

Figure 11 shows the simulation waveforms of the Receiver Module component for an example of transmitted data. For depiction purposes, the transmitted data TXdata output of the Transmitter Module is connected to the RXdata input and both the TXCLK and the RXCLK are running at the same clock. It is clear in this figure that the output data bus of this entity is similar to the data transmitted.

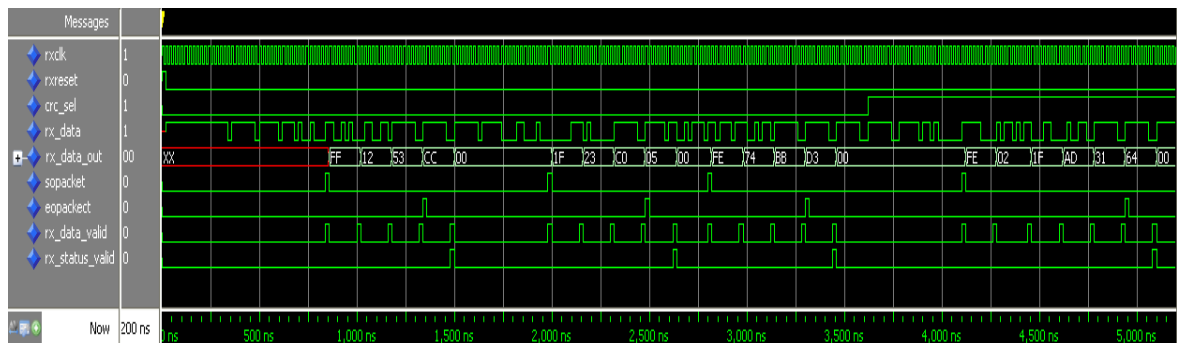


Figure 11: The simulation waveforms of the Receive Module

The three last bytes represent CRC 2_bytes and the status Octet. The status byte " 0x00" indicates an intact transmission (no errors). Other detailed waveform results, like the presence of the abort sequence, using CRC 32 bits, CRC error and octet error are beyond the scope of this paper and can be requested from the authors.

As with the transmitter module, figure 12 shows the top-level schematic design for the implemented receiver on Cyclone II FPGA board EP2C35F672C6 after successful compilation. TABLE 2 shows the resources taken by the receiver module units.

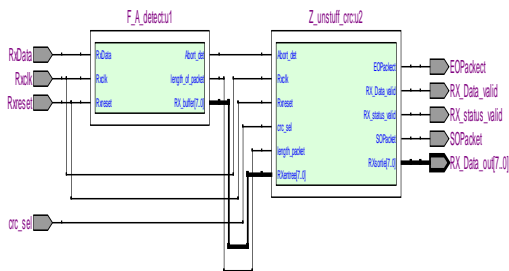


Figure 12. Top-level design of the implemented Receiver

Table 2. Resources taken by the units of the receiver module

Module	LogicCells	Dedicated LogicRegisters	Pins	LUT-Only LCs
F_A_detect: U1	95	49	0	46
Z_unstuff_crc: U2	223	135	0	88
Receiver Module (Total)	318	184	16	134
Utilization	<1% (/33216)	< 1% (/33216)	3% (/475)	-
Total thermal power Dissipation(mw)	111.79			

The single channel HDLC receiver is found to be running at a frequency of 199.12 MHz (5.022ns) with a total equivalent cell count of 318.

4. CONCLUSION

A Single-Channel HDLC Controller was successfully designed using VHDL codes, tested and verified functionally using Modelsim and Quartus II softwares. The whole system

is based on a modular design, and can therefore be easily extended and upgraded. The HDLC controller has the following features: bit stuffing/unstuffing, Flag sending/detection, CRC computation/verification, octet error/abort detection and converting serial/parallel bits into parallel/serial data. Along the paper, the focus is to clearly show the operation of the modules using quite detailed diagrams. The simulation results show that the module is reliable. The design has been synthesized using ALTERA FPGA Technology.

5. REFERENCES

- [1] Information technology -- Telecommunications and information exchange between systems -- High-level data link control (HDLC) procedures, ISO/IEC 13239:2002, www.iso.org.
- [2] Yuanlin Lu; Zhigong Wang; LufengQiao; Bin Huang, "Design and implementation of multi-channel high speed HDLC data processor," in *Communications, Circuits and Systems and West Sino Expositions, IEEE 2002 International Conference on*, vol.2, no., pp.1471-1475 vol.2, 29 June-1 July 2002
- [3] S. M. Qasim, and S. A. Abbasi, "FPGA implementation of a single-channel HDLC layer-2 protocol transmitter using VHDL," Proc. 15th Int. Conf. on Microelectronics ICM, Cairo, Egypt, December 2003, pp. 265–268
- [4] S. M. Qasim, and S. A. Abbasi, "Hardware Realization of Single-Channel HDLC Protocol Transmitter using FPGA," Global Signal Processing Expo & Conference'2004, Santa Clara, California USA Sept.27-30
- [5] Gao Zhen-Bin; Liu Jian-fei, "FPGA implementation of a multi-channel HDLC protocol transceiver," in *Communications, Circuits and Systems, 2005.Proceedings. 2005 International Conference on*, vol.2, no., pp.1302, 27-30 May 2005
- [6] Arshak, K.; Jafer, E.; McDonagh, D.; Ibala, C.S., "Modelling and simulation of wireless sensor system for health monitoring using HDL and Simulink mixed environment," in *Computers & Digital Techniques, IET*, vol.1, no.5, pp.508-518, Sept. 2007

- [7] J. Wang; W. Zhang; Y. Zhang; W. Wu; W. Chang, "Design and implementation of HDLC procedures based on FPGA," *Anti-counterfeiting, Security, and Identification in Communication*, 2009. ASID 2009. 3rd International Conference on , vol., no., pp. 336, 339, 20-22 Aug. 2009.
- [8] G. Li; N. Tan; "Design and Implementation of HDLC Protocol and Manchester Encoding Based on FPGA in Train Communication Network," *Information and Computing (ICIC)*, 2010 Third International Conference on. Vol.1, pp. 105-108, 2010.
- [9] Hong Luo; Cheng Chang; Yan Sun, "Advanced sensor gateway based on FPGA for wireless multimedia sensor networks," in *Electric Information and Control Engineering (ICEICE)*, 2011 International Conference on , vol., no., pp.1141-1146, 15-17 April 2011
- [10] Chen Zhifeng, Chen He, "Implementation and Application of HDLC Protocol Based on FPGA in Radar Processing System", School of information and electronics, Electronics, Communications and Control (ICECC), 2011 International Conference; Sept. 2011.
- [11] W. Lie; Y. Ming, "Design of HDLC controller based on Xilinx FPGA," *Computer Science and Network Technology (ICCSNT)*, 2011 International Conference on , vol.3, no., pp.1362,1366, 24-26 Dec. 2011.
- [12] M.Sridevi, P.S. Reddy " Design And Implementation Of Hdlc Protocol On Fpga," *International Journal of Engineering Research and Applications (IJERA)*, Vol. 2, Issue 5, September- October 2012, pp.2217-2219
- [13] G. Chandil, P. Mishra " Design and Implementation of HDLC Controller by Using Crc-16," *International Journal of Modern Engineering Research (IJMER)*, Vol. 3, Issue. 1, Jan.-Feb. 2013 pp-12-18.
- [14] "HDLC Controller Implemented in ispMACH 4000ZE and CPLD Families," Reference Design RD1009, July 2009 .www.latticesemi.com
- [15] A.Aftab , « Data Communication Principles: For Fixed and Wireless Networks », Kluwer Academic Publishers, 2003.
- [16] "Parallel Cyclic Redundancy Check (CRC) for HOTLink ," Document No. 001-27960 Rev. *A, March 11,1999, <http://www.cypress.com/?docID=31573>