IOSR Journal of Engineering (IOSRJEN)
e-ISSN: 2250-3021, p-ISSN: 2278-8719, www.iosrjen.org
Volume 2, Issue 10 (October 2012), PP 41-50

Study and Performance Evaluation of Xilinx HDLC Controller and FCS Calculator

Gaurav Chandil[1], Priyanka Mishra[2]
[1,2] *(Department of Electronics and Communication Engineering, United Group of Institutions, Allahabad, India)*

Abstract—*In this paper, design, simulation and implementation design of HDLC Controller provides a high performance. This design is then coded in a hardware description language (VHDL). The functioning of the coded design is to simulate on simulation software (e.g. ModelSim). After proper simulation, the design is synthesized and then translated to a structural architecture in terms of the components on the target FPGA device (Spartan 3) and the perform the post-translate simulation in order to ensure the proper functioning of the design after translation. After the successful simulation of the post-translate model the design is mapped to the existing slices of the FPGA and the post-map model simulated. The post-map model does not include the routing delays. After the successful completion of the post-map simulation, the design is then routed and a post-route simulation model with the appropriate routing delays is generated to be simulated on the HDL simulator. After this a programming file is generated to program the FPGA device. The objective of this paper is to run the programmed FPGA at a frequency as high as possible. HDLC Controller MEGACELL is a high performance module for the bit oriented, switched, non-switched packet transmission module. The controller fulfills the specifications according to ITU Q.921, X.25 Level 2 recommendation. It supports half duplex and full duplex communication lines, point-to-point and multipoint channels. Furthermore, the Controller is designed to permit synchronous, code transparent data transmission. The control information is always in the same position and specific bit patterns used for control differ dramatically from those representing data that reduces the chances of errors. The data stream and transmission rate is controlled by the network node. In this paper, we implemented the various HDLC Controller and bit stuffing and removal of error in HDLC.*

Keywords — *High level data link control (HDLC) Controller, FCS, cyclic redundancy check* (*CRC), SDLC.*

I. INTRODUCTION

HDLC (High-level Data Link Control) is a group of protocols for transmitting synchronous data packets between Point-to-Point nodes. In HDLC, data is organized into frames. HDLC protocol resides with Layer 2 of the OSI model, the data link layer. It is an efficient layer 2 protocol standardized by ISO for point-to-point and multipoint data links. It provides minimal overhead to ensure flow control, error control, detection and recovery for serial transmission. HDLC uses zero insertion/deletion process (bit stuffing) to ensure that the bit pattern of the delimiter flag does not occur in the fields between flags. The HDLC frame is synchronous and therefore relies on the physical layer to provide method of clocking and synchronizing the transmission and reception of frames.

The HDLC frame is synchronous and therefore relies on the physical layer to provide method of clocking and synchronizing the transmission and reception of frames. The frames are separated by HDLC flag sequences that are transmitted between each frame and whenever there is no data to be transmitted. To inform the receiving station that a new packet is arriving and synchronizes the receive clock with the transmitted clock a specific bit pattern is added at the front and the back of the packet. The header of the packet contains an HDLC address and an HDLC control field. The specific bit pattern is used to affix with the packet in the case of HDLC Controller is 01111110. The length of the address field is normally 0, 8 or 16 bits in length. In many cases the address field is typically just a single byte, but an Extended Address (EA) bit may be used allowing for multi-byte addresses. A one residing in the LSB bit indicates the end of the field that the length of the address field will be 8 bits long. A zero in this bit location (now the first byte of a multi-byte field) indicates the continuation of the field (adding 8 additional bits). The Control field is 8 or 16 bits and defines the frame type; Control or data .To guarantee that a flag does not appear inadvertently anywhere else in the frame, HDLC uses a process called bit stuffing. Every time the user wants to send a bit sequence having more than 5 consecutive 1s, it inserts stuffs one redundant 0after the fifth 1. The trailer is found at the end of the frame, and contains a Cyclic Redundancy Check (CRC), which detects any errors that may occur during transmission. A CRC value is generated by a calculation that is performed at the source device. The destination device compares this value to its own calculation to determine whether errors occurred during transmission. First, the source device performs a

predetermined set of calculations over the contents of the packet to be sent. Then, the source places the calculated value in the packet and sends the packet to the destination. The destination performs the same predetermined set of calculations over the contents of the packet and then compares its computed value with that contained in the packet. If the values are equal, the packet is considered valid. If the values are unequal, the packet contains errors and is discarded. The receiver can be configured into transparent mode, effectively disabling the HDLC protocol functions. In normal HDLC protocol made, all received frames are presented to the host on the output register. A status register is provided which can be used to monitor the status of the receiver channel, and indicates if the packet currently being received includes any errors.HDLC has three operational modes. These modes are Normal Response Mode (NRM), Asynchronous Response Mode (ARM) and Asynchronous Balanced Mode (ABM).Normal Response Mode refers to the standard primary-secondary relationship. In this mode, a secondary device must have permission from the primary device before transmitting. Once permission from the secondary has been granted, the secondary may initiate a response transmission of one or more frames containing data. Asynchronous Response Mode (ARM) is a secondary may initiate the transmission without permission from the secondary whenever the channel is idle. ARM does not alter the primary-secondary relationship in any other way. All transmissions from a secondary must still be made to the primary for relay to final destination. In Asynchronous Balanced Mode (ABM) all stations are equal and therefore only combined stations connected in point-to-point are used. Either combined station may initiate transmission with the order-combined station without permission.

In OSI 7 Layers Reference Model the CDAC HDLC controller operates at the data link layer of the OSI Model. Hence, the main focus of the survey is to understand the data link layer and develop a protocol which can offer its services to the layer above it i.e. is the network layer and the layer below it i.e. the physical layer.

The main function of this protocol controller is to perform a number of separate activities like physical addressing, to check for errors, flow control etc.The layered concept of networking was developed to accommodate changes in technology. Each layer of a specific network model may be responsible for a different function of the network. Each layer will pass information up and down to the next subsequent layer as data is processed [1].

Open Systems Interconnection (OSI) model is a reference model developed by ISO (International Organization for Standardization) in 1984, as a conceptual framework of standards for communication in the network across different equipment and applications by different vendors. It is now considered the primary architectural model for inter-computing and internetworking communications. Most of the network communication protocols used today have a structure based on the OSI model. The OSI model defines the communications process into 7 layers, which divides the tasks involved with moving information between networked computers into seven smaller, more manageable task groups. A task or group of tasks is then assigned to each of the seven OSI layers. Each layer is reasonably self-contained so that the tasks assigned to each layer can be implemented independently. This enables the solutions offered by one layer to be updated without adversely affecting the other layers. [1]

The seven OSI layers use various forms of control information to communicate with their peer layers in other computer systems. This control information consists of specific requests and instructions that are exchanged between peer OSI layers. Control information typically takes one of two forms: headers and trailers. Headers are pretended to data that has been passed down from upper layers. Trailers are appended to data that has been passed down from upper layers. An OSI layer is not required to attach a header or a trailer to data from upper layers. Headers, trailers, and data are relative concepts, depending on the layer that analyzes the information unit. At the network layer, for example, an information unit consists of a Layer 3 header and data. At the data link layer, however, all the information passed down by the network layer (the Layer 3 header and the data) is treated as data. In other words, the data portion of an information unit at a given OSI layer potentially can contain headers, trailers, and data from all the higher layers. This is known as encapsulation. How the header and data from one layer are encapsulated into the header of the next lowest layer [1].

II. GENERAL HDLC FRAME FORMAT

In figure 1, user data which contains 7E is resolved using an escape sequence which converts 7E to 7D-5E (with 7D being the escape character). If 7D is used in the data stream it again is converted into 7D-5D. Address 11111111 is known as all stations, 00000000 is this station. Frames may be aborted by sending an abort sequence [01111111] instead of the normal flag sequence [01111110]. An abort sequence will cause the frame to be discarded. During idle times when no frames are being transmitted idle flags [11111111] may be sent to fill the area between frames. A continuous series of flags [01111110] may be sent to fill the area between frames instead of sending idle flags [11111111].

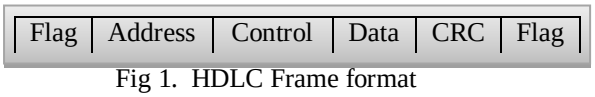

Fig 1. HDLC Frame format

Opening Flag, 8 bits [01111110], [7E Hex]
Address, 8 bits/16 bits
Control, 8 bits, or 16 bits
Data [Payload], Variable, not used in some frames, or may be padded to complete the fill
CRC, 16 bits, or 32 bits
Closing Flag, 8 bits [01111110], [7E hex]

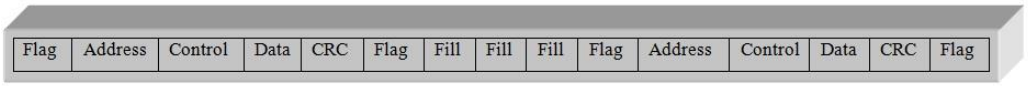

Fig 2. Fill between Frames

2.1 BIT STUFFING IN HDLC

To guarantee that a flag does not appear inadvertently anywhere else in the frame, HDLC uses a process called bit stuffing. Every time the user wants to send a bit sequence having more than 5 consecutive 1s, it inserts (stuffs) one redundant 0after the fifth 1.

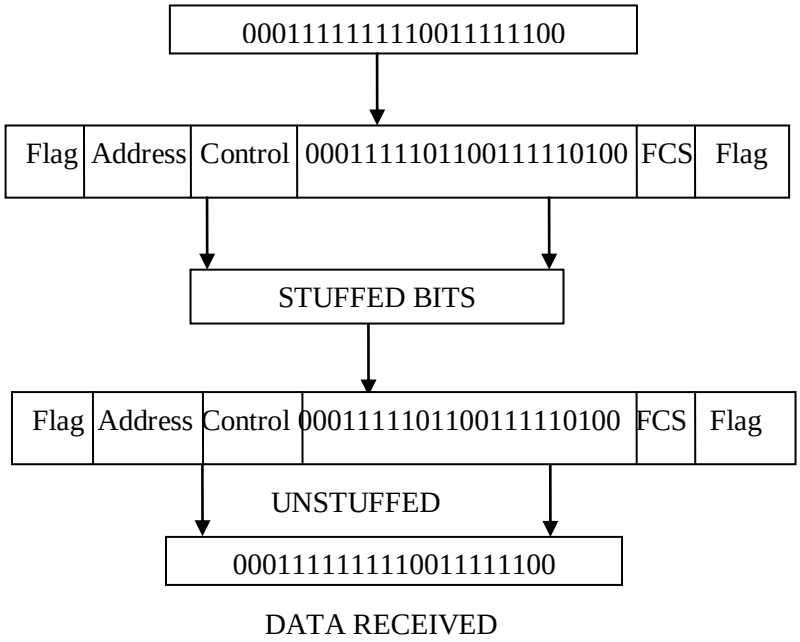

Fig 3. Bit stuffing and removal

For example the sequence 01111111111000 becomes 011111101111000. This extra zero is inserted regardless of whether the sixth bit is another one or not. Its presence tells the receiver that the current sequence is not a flag. Once the receiver has seen the stuffed 0, it is dropped from the data and the original stream is retorted Fig 3. shows the bit stuffing at the sender's end and bit removal at the receiver.

When it finds five consecutive 1s after a zero, it checks the seventh bit. If the seventh bit is a 0, the receiver recognizes it as a stuffed bit and discards it, and resets the counter. If the seventh bit is a 1, the receiver checks the eighth bit. If the eighth bit is another 1, the receiver continues counting. A total of 7 to 14 consecutive 1s indicates an abort. A total of 15 0r more 1s indicates an idle channel. The flowchart of the above discussion is illustrated below.

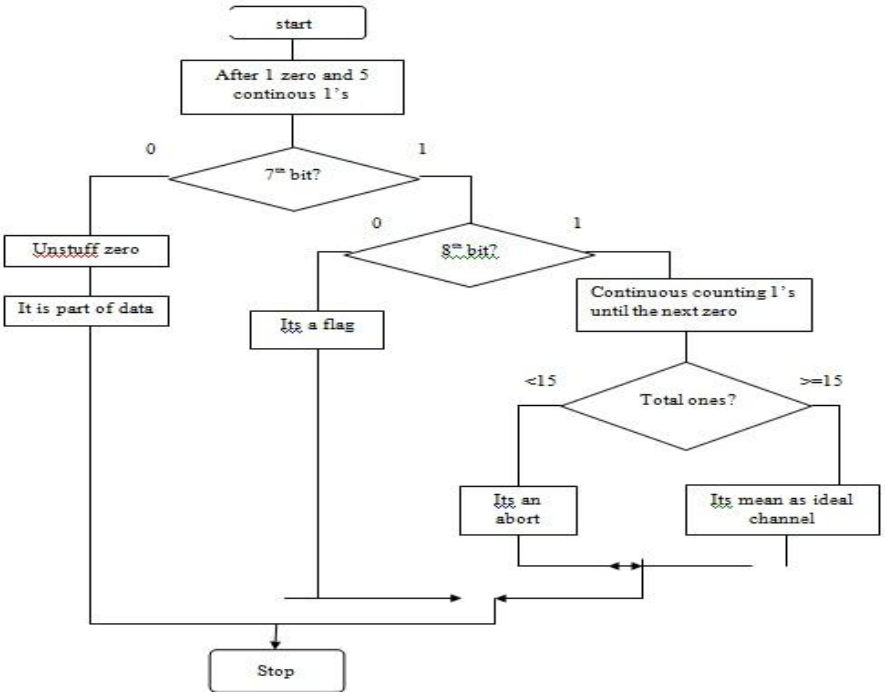

Fig 4.Flowchart of bit stuffing and removal

III. IMPLEMENTATION OF HDLC CONTROLLER IN XILINX

The whole design is organized as a collection of 2 sections that work together to efficiently perform the operation as shown in fig 1. These units are –

Transmitter section- The Transmit Data Interface provides a byte wide interface between the transmission host and the HDLC Protocol core. Transmit data is loaded into the core on the rising edge of clk when the write strobe input asserted. The start and end bytes of a transmitted HDLC frame are indicated by asserting the appropriate signals with the same timing as the data bytes. The HDLC core will, on receipt of the first byte of a new packet, issue the appropriate flag sequence and transmit the frame data calculating the FCS. When the last byte of the frame is seen, the FCS is transmitted along the closing flag. Extra zeroes are inserted into the Bit stream to avoid transmission of the control flag sequence within the frame data. The transmit data is available on the TxD pin with appropriate setup to be sampled be clk. If TxEN is reasserted, the transmit pipeline is stalled, and the TxD pin is tristated. A transmit control register is provided which can enable or disable the channel, select transparent mode where the HDLC protocol is disabled, and specify the HDLC core action on transmit FIFO under runs. In addition, it is possible to force the transmission of the HDLC Abort sequence. This will cause the currently transmitted frame to be discarded. The transmit core can be configured to automatically restart after an abort, with the next frame, or to remain stalled until the host microprocessor cleared the abort or transmit FIFO under run conditions shown in figure 5.

Receiver section- Receiver accepts a bit stream on port RxD. The data is latched on the rising edge of clk under the control of the enable input RxEN. The flag detection block stream for the flag sequence in order to determine the frame boundaries. Any stuffed zeroes are detected and removed and the FCS is calculated and checked. Frame data is placed on the receiver data interface and made available to the host. In addition, flag information is passed over indicating the start and end bytes of the HDLC frame as well as showing any error condition which may have been detected during receipt of the frame. The receiver can be configured into transparent mode, effectively disabling the HDLC protocol functions. In normal HDLC protocol made, all received frames are presented to the host on the output register. A status register is provided which can be used to monitor the status of the receiver channel, and indicates if the packet currently being received includes any errors. The HDLC protocol core receiver accepts a bit stream. The flag detection block searches the bit stream for the flag sequence in order to determine the frame boundaries. Any stuffed zeroes are detected and removed by the zero deletion blocks.

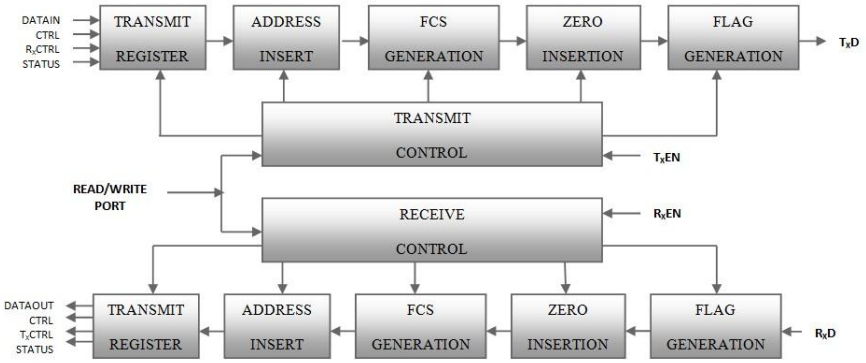

Fig 5. Basic block diagram of HDLC Controller

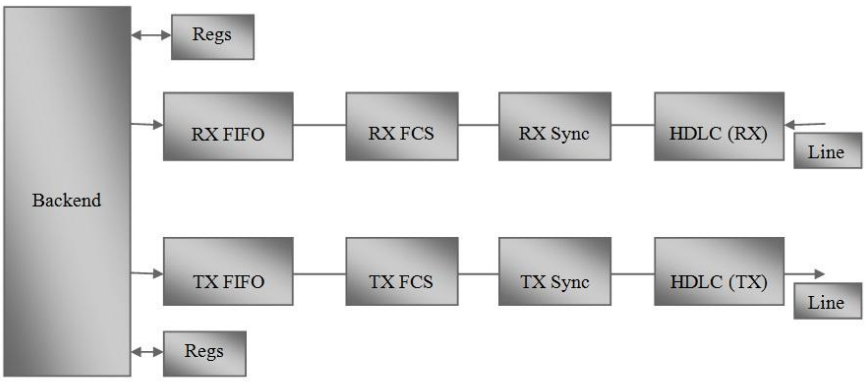

Fig 6.Transmit and receive the signal through backend

IV. FCS CALCULATOR

A powerful method for detecting errors in the received data is by grouping the bytes of data into a block and calculating a Cyclic Redundancy Check (CRC). This is usually done by the data link protocol and calculated CRC is appended to the end of the data link layer frame.

The CRC is calculated by performing a modulo 2 division of the data by a generator polynomial and recording the remainder after division.

1. A string of 0s is appended to the data unit. The no. n is less than the no. of data of bits in the predetermined divisor, which is n+ 1 bit.
2. The newly elongated is divided is divided by the divisor. The remainder is the CRC
3. CRC replaces n 0 bits derived in step 2 at the end of data unit.
4. Data unit arrives at the receiver data first, followed by CRC. The receiver treats the whole string as a data unit and divides by the same divisor.
5. If remainder comes out to be 0 the string is error free.

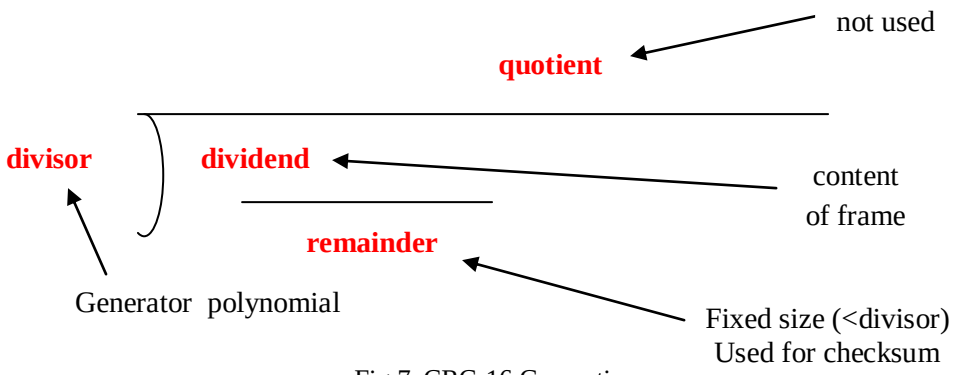

Fig 7. CRC-16 Generation

Three polynomials are in common use they are:
CRC-16 = x16 + x15 + x2+ 1 (used in HDLC)

CRC-CCITT = x16 + x12 + x5 + 1
CRC-32 = x32 + x26 + x23 + x22 + x16 + x12 + x11 + x10 + x8 + x7 + x5 + x4 + x2 + x + 1 (used in Ethernet)
Although this division may be performed in software, it usually performed using a shift register and X-OR gates. The hardware solution for implementing a CRC is much simpler than a software approach. For a CRC-16. A practical implementation of a decoder also requires a method to initialise the encoder prior to transmission of the first bit of data in a frame, and to flush the encoder after sending the last byte. In the example below (which uses a different representation of the schematics for X-OR gates and shift register elements), the process starts by initialising the encoder with zero bits, by setting the switch to B. Some CRC's initialise the register to a non-zero value, which can give added detection capability when the first set of bits in a frame may themselves be zero. Then the switch is moved to position A and one data bit enter the encoder for each clock cycle. The data bits are immediately available at the output. After the last bit has been sent, the switch is returned to position B and the contents of the encoder are sent to the output. This is often called flushing the encoder and requires one clock cycle per bit held in the shift register as shown in figure 8.

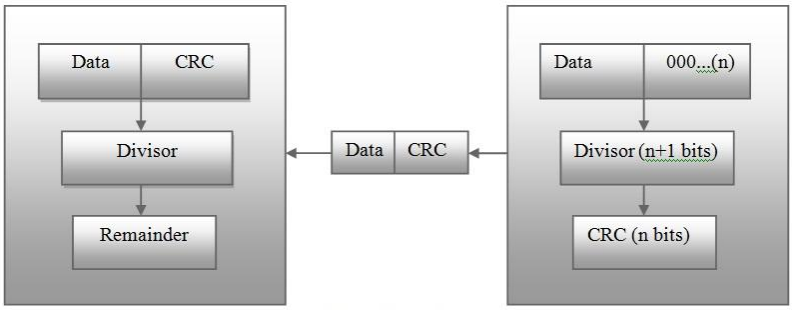

Fig 8. FCS calculation

A practical implementation of a decoder also requires a method to initialise the encoder prior to transmission of the first bit of data in a frame, and to flush the encoder after sending the last byte. In the example below (which uses a different representation of the schematics for X-OR gates and shift register elements), the process starts by initialising the encoder with zero bits, by setting the switch to B. Some CRC's initialise the register to a non-zero value, which can give added detection capability when the first set of bits in a frame may themselves be zero. Then the switch is moved to position A and one data bit enter the encoder for each clock cycle. The data bits are immediately available at the output. After the last bit has been sent, the switch is returned to position B and the contents of the encoder are sent to the output. This is often called flushing the encoder and requires one clock cycle per bit held in the shift register in Fig 9.

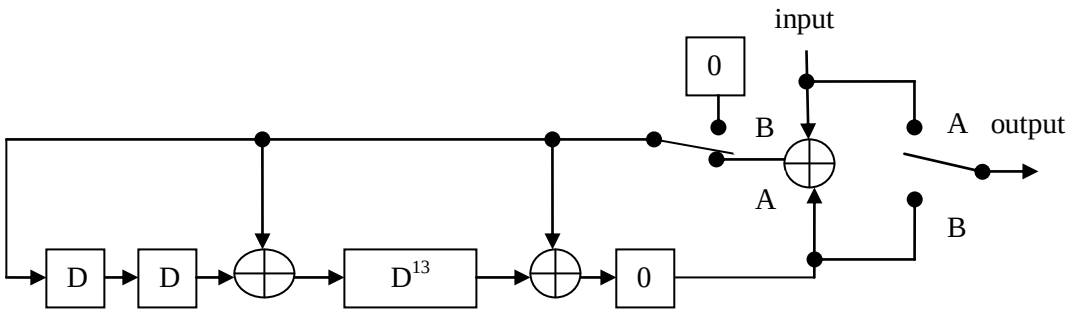

Fig 9. Diagram of suggested implementation of an Encoder/Decoder

On reception, the process is reversed. The CRC register is first set to zero (or the initial value on transmission, if non-zero). The bits (this time including the CRC) are fed into the register on each clock cycle. If the CRC contains the value zero (assuming initialisation was zero), the CRC is valid, if not it has detected an error. The CRC-16 is able to detect all single errors, all double errors, all odd numbers of errors and all errors with burst less than 16 bits in length. In addition 99.9984 % of other error patterns will be detected. Protocols at the network layer and higher (e.g. IP, UDP, TCP) usually use a simpler checksum to verify that the data being transported has not been corrupted by the processing performed by the nodes in the network.

V. SYNTHESIS AND SIMULATION RESULT

After the design and implementation of the HDLC Controller, the results obtained are as follows:

```
Title      :  HDLC components package for HDLC controller
File       : hdlc_components_pkg.vhd
Simulators  : Modelsim 5.3XE/Windows98
Dependency  : ieee.std_logic_1164

library ieee;
use ieee.std_logic_1164.all;
package hdlc_components_pkg is
  component rxcont_ent
    port (RxClk        : in  std_logic;
      rst          : in  std_logic;
      RxEn         : in  std_logic;
      AbortedFrame : out std_logic;
      Abort        : in  std_logic;
      FlagDetect   : in  std_logic;
      ValidFrame   : out std_logic;
      FrameError   : out std_logic;
      aval         : in  std_logic;
      initzero     : out std_logic;
      enable       : out std_logic);
  end component;

  component ZeroDetect_ent
    port (Readbyte     : in  std_logic;
      aval         : out std_logic;
      enable       : in  std_logic;
      StartOfFrame : in  std_logic;
      rdy          : out std_logic;
      rst          : in  std_logic;
      RxClk        : in  std_logic;
      RxD          : in  std_logic;
      RxData       : out std_logic_vector(7 downto 0));
  end component;

  component FlagDetect_ent
    port (Rxclk      : in  std_logic;
      rst        : in  std_logic;
      FlagDetect : out std_logic;
      Abort      : out std_logic;
      RXD        : out std_logic;
      RX         : in  std_logic);
  end component;

  component RxChannel_ent
    port (Rxclk       : in  std_logic;
      rst         : in  std_logic;
      Rx          : in  std_logic;
      RxData      : out std_logic_vector(7 downto 0);
      ValidFrame  : out std_logic;
      AbortSignal : out std_logic;
      FrameError  : out std_logic;
      Readbyte    : in  std_logic;
      rdy         : out std_logic;
      RxEn        : in  std_logic);
  end component;
end hdlc_components_pkg;
```

```
------------------------CODE FOR CRC GENERATION-------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

Uncomment the following library declaration if instantiating any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity hdlc is
port(clk,reset,wrtxaddrhi,wrtxaddrlo,wrtxctrl:in std_logic;
datain:in std_logic_vector(7 downto 0);
txaddressout:out std_logic_vector(15 downto 0);
txd:out std_logic;
txaddressin1,txaddressin2,txctrlreg:in std_logic_vector( 7 downto 0));
end hdlc;
.
.
-----crc 16 generation------------

a2:process(clk,done,crcappreg)
variable count:std_logic_vector(5 downto 0):="000000";
begin    ---crc16 for 16 bit address
.
.
end process a2;

--------------crc32 generation------------

a3:process(clk,done)
variable count:std_logic_vector(5 downto 0):="000000";
begin
.
.
end process a3;

---------Code for zero insertion at the transmitter-----------
zero insertion for 16 bit address and crc32

a4:process(ready,ready1,ready2,ready3,clk)
variable count1, count2,count3,count4:std_logic_vector(5 downto 0):="000000";
begin
.
.
end process a4;

end Behavioral;
---------Code for CRC Check and ZERO deletion----------
Uncomment the following library declaration if instantiating any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;
entity hdlcrx is
.
.
end process a2;
```

1.1. Simulation result for 8-bit data, 8 bit address and crc-16: For the data<=00001110and8 bit address<=11110000, we make clock=1, reset=0, wrtaddresshi=0 and wrtaddresslo=1. After the address and the

data are attached together, we divide them with a constant polynomial and append the remainder of the division along the data and address. The simulation result for the generation of crc1 is given in figure 10.

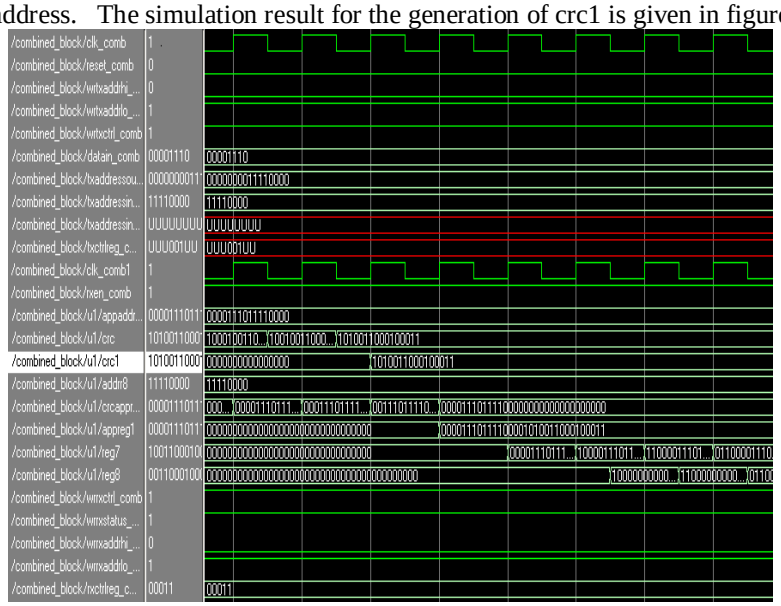

Fig 10. Simulation result of the final O/P at the receiver end for 8 bit and 16-bit crc

1.2. Simulation result of the final O/P at the receiver end for 8 bit data: In figure 11,8bit address and 16-bit crc. At the receiver the data input, address and the appendedcrc is again divided with the same constant polynomial and if the crc2 comes out be zero, it shows an error free reception of the packet. The receiver O/P i.e. rxdataout<=00001110 and rxaddressout<=0000000011110000 which is same as that of transmitter.

VI. CONCLUSION

In this paper study and investigate of simulation of xilinx HDLC Controller It can automatically check frame sequence generation using cyclic redundancy check CRC-16. The result of Post Synthesis is an Optimized Gate Level net list form which a net list code is extracted and it is simulated using Simulator and it is verified that design is working properly. In this paper study and investigate of Removal of bit stuffing in HDLC. It can automatically check frame sequence by every time the user wants to send a bit sequence having more than 5 consecutive 1s, it inserts (stuffs) one redundant 0after the fifth 1.In this paper study and investigate of HDLC and CRC Calculation. HDLC controller has the capability to operate in full duplex and half duplex mode. It can automatically check frame sequence generation using cyclic redundancy check CRC-16. And It is compatible with all the protocols present at the physical layer i.e.X.25 protocol and network layer i.e. Internet protocol (IP protocol).

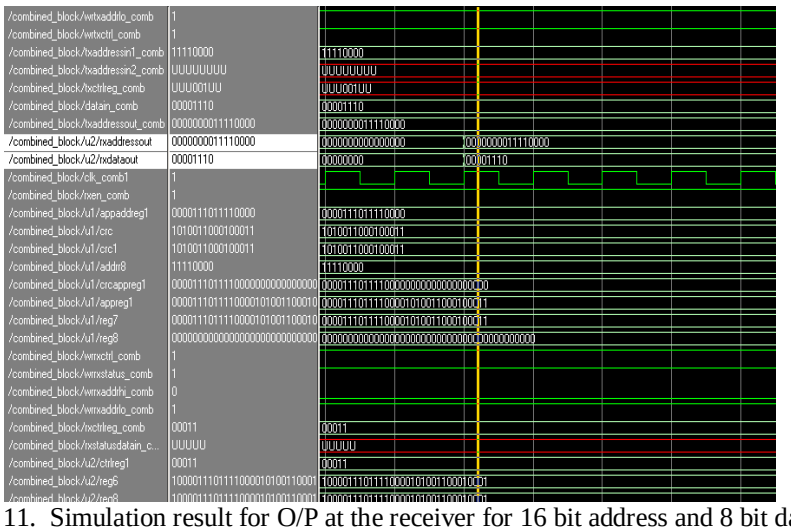

Fig 11. Simulation result for O/P at the receiver for 16 bit address and 8 bit data

ACKNOWLEDGEMENTS

Mr. Gaurav Chandil received his M.tech degree in VLSI Design from ABV-IIITM, Gwalior (M.P.) in 2011 and received B.tech in Electronics & Communication Engineering from ITM University, Rajiv Gandhi Proudyogiki Vishwavidyalaya,Bhopal in 2008.He worked as Lecturer in Electronics and Communication Engineering Department at Institute of Engineering Science & Management (IESM), Indore (M.P.), from July-2008 to June-2009,he also worked as an Assistant Professor in Electronics and Tele-communication Engineering Department at G L Bajaj Group of Institutions, Mathura (U.P) and at College of Engineering Roorkee (COER), Roorkee. Currently, he is working as an Asst. Professor in United Group of Institution, Allahabad. His area of interest are VLSI technology, Digital Signal Processing, linear integrated circuits, signal & system, Network Analysis and Control Engineering.

Ms. Priyanka Mishra received her M.Tech degree in Electronics and Communication Engineering degree from Jaypee University of Information Technology, Solan in 2012 and received her B.Tech degree from BBS College of engineering & technology in 2009. Currently,she is working as Assistant Professor in Electronics and Communication Engineering department in the United Group of Institutions, Allahabad. Her area of interest is Wireless communication, data communication, MIMO technologies and wireless 4G communication.

REFERENCES

[1] A Petri net simulation model of HDLC Marsan, M.A.; Barbetta, L.; Neri, F.; TENCON '89. Fourth IEEE Region 10 International Conference 22-24 Nov. 1989Page(s):240–247Digital Object Identifier 0.1109/TENCON.1989.176933.

[2] Modified byte insertion/deletion for HDLC in ISDN Davis, G.T.; Mandalia, B.D.; Southeastcon '89. Proceedings. 'Energy and Information Technologies in the Southeast'. IEEE 9-12 April 1989 Page(s):1207 - 1210 vol.3 Digital Object Identifier 10.1109/SECON.1989.132614.

[3] FPGA implementation of a single-channel HDLC Layer-2 protocol transmitter using VHDL Qasim, S.M.; Abbasi, S.A.; Microelectronics, 2003. ICM 2003. Proceedings of the 15th International Conference on 9-11.

[4] Lu, Y., Z. Wang, L. Qiao and B. Huanq, 2002. "Design and implementation of multi-channel high speed HDLC data processor," IEEE International Conference on Communications, Circuits and Systems, and West Sino Expositions, 2: 1471-1475.

[5] Khosla, P. and Volpe, R. (1988). Superquadric artificial potentials for obstacle avoidance and approach. In Proc. of the IEEE Conf. on Robotics and Automation

[6] Latombe, J. (1996). Robot Motion Planning. Kluwer Academic Publishers, UK.

[7] Khatib, O. (1985). Real-time obstacle avoidance for manipulators and mobile robots. In Proc. of the IEEE Intl.

[8] Conf. on Robotics and Automation.

[9] Manz, A., Liscano, R., and Green, D. (1991). A comparison of real-time obstacle avoidance methods for mobile robots.

[10] Borenstein, J. and Koren, Y., 1991, "The Vector Field Histogram - Fast Obstacle- Avoidance for Mobile Robots," IEEE Journal of Robotics and Automation

[11] Shoval, S., Borenstein, J., and Koren, Y., 1994, "Mobile Robot Obstacle Avoidance in a Computerized Travel Aid for the Blind," IEEE International Conference on Robotics and Automation

[12] Yehuda Sonnenblick, "An Indoor Navigation System For Blind Individuals," URL: http://www.dinf.ne.jp/doc/english/Us_Eu/conf/csun_98 /csun98_008.htm.

[13] Jun Wang; Wenhao Zhang; Yuxi Zhang; Wei Wu; Weiguang Chang; Sch. of Electron. & Inf. Eng., Beihang Univ. (BUAA), Beijing, China "Design and implementation of HDLC procedure based on FPGA" , Anti- ounterfeiting, Security, and Identification in Communication, 2009. ASID 2009. 3rd International Conference, 20-22 Aug.2009.

[14] Guozheng Li Nanlin Tan State Key Lab. of Rail Traffic Control & Safety, Beijing Jiaotong Univ.,Beijing, China "Design and Implementation of HDLC Protocol and Manchester Encoding Based on FPGA in Train Communication Network", Information and Computing (ICIC), 2010 Third International Conference.