International Journal of Computer Information Systems and Industrial Management Applications.
ISSN 2150-7988 Volume 6 (2014) pp. 66 -76

An FPGA Implementation of 1553 Protocol Controller

JEMTI JOSE
Dept. of Electronics and Communication Engineering
St. Joseph's College of Engineering and Technology
Palai, Kerala, India
jemtijose@gmail.com

***Abstract*: In a modern military avionics system all the devices need to communicate as efficiently as possible with a minimum amount of hardware. 1553 is a dual- redundant, bi-directional, Manchester encoded, digital time division command/ response data bus which eliminates the use of point-to-point wiring. It uses a shielded twisted pair wire for data transfer. This bus can allow communication between any devices (maximum of 31) connected to it. Even though 1553 is an old standard (developed in early 1970s), it is an inevitable part of almost all aircrafts of today. Compared to other avionics data bus standards 1553 is known for its reliability and flexibility. With the presented method, the protocol controller is modeled as state machine in HDL. This paper describes implementation of the Military data bus standard MIL-STD-1553 onto a Xilinx based FPGA platform.**

***Keywords*:** 1553 data bus, Bus Controller (BC), Remote Terminal (RT), Command Word (CW), Data Word (DW), Status Word (SW), FPGA, HDL**.**

I. Introduction

MIL-STD-1553 is a popular multiplex data bus standard which is originally developed by the U.S Air force for data integration in their military aircrafts. Since its inception in 1973 [2] it has undergone a number of revisions, first a tri-service version (means for Army, Navy and Air force) [3] is released and after further changes and improvements it has been opened to commercial applications as well [1]. Now it has developed as one of the internationally accepted networking standard and an inevitable component for ships, satellites, missiles, and the International Space Station Program, as well as advanced commercial avionic applications [2].

As we know the military and commercial avionic environments are full of interferers that may affect the signal integrity of data transmitted over the data bus. 1553 data bus uses a twisted shielded pair wire for information transfer. The twisted pair provides noise cancelling and the shielding limits interference from external sources thereby providing better EMI performance [4]. Also this bus system allows the use of redundant pair of buses, i.e. an alternative path in the case of damage / failure [5]. The dual redundant signal paths of 1553 make it suitable for flight-critical systems [1].

Almost all the data bus standards require point-to-point wiring; one such standard is ARINC 429, which uses a point-to-point unidirectional multi-drop topology [6]. Since size and weight are the critical factors for aircrafts and payloads, the single wire, highly flexible 1553 standard dominated the commercial avionics industry. Also, 1553 has a track record of more than 30 years of in service history in avionics. Recently Airbus has decided to adopt 1553 for its new aircraft A350 and Boeing is going to use it in its next generation aircrafts [6]; both indicate the reliability of the standard [1].

This paper presents the design of the military data bus standard MIL-STD-1553 protocol controller using HDL. This paper is organized as follows: Section 2 presents a brief overview of the 1553 data bus protocol; Section 3 describes the design and implementation of the protocol controller in detail with reference to the system architecture, system design and its implementation onto an FPGA platform. In section 4 simulation and test results obtained are presented. Conclusions are outlined in Section 5 followed by a set of references.

II. System overview

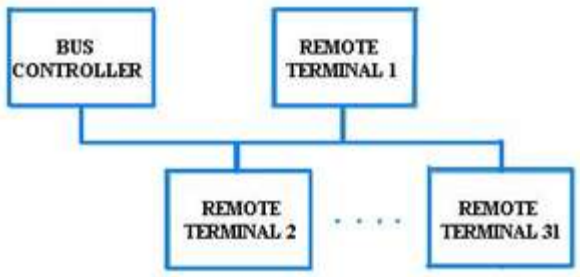

Figure 1. Sample of architecture of 1553 data bus [1]

A. Bus Structure

1553 data bus standard defines the characteristics of a serial, multiplex, time-division data bus [5]. The bus operates asynchronously, employing a command/response protocol [5]. The main components of MIL-STD-1553 bus system are the bus controller, the remote terminal and the twisted

shielded pair wire data bus. The 1553 bus can allow data transmission between a number of devices, among these devices one should be designed to work as a Bus Controller, which controls the entire transmission operations on the bus, and the other devices act as remote terminals. A maximum of 31 remote terminals can be connected to a single twisted pair cable in 1553 bus [7].

B. Word Formats

The information transfer through the bus is done as messages, each of which contains a set of words. There are three distinct word types defined by the protocol. These are Command Words (CW), Data Words (DW) and Status Words (SW) [7]. Each word type has a unique format, but all three maintain a common structure. Each word is twenty bits in length. The data code transmitted on the bus shall be Manchester II bi-phase level [8]. Manchester coding has the advantage of transitions in each bit of data. So error detection will be easy [1].

All message transfers are initiated by a CW from BC. Command words are transmitted by the bus controller to remote terminals to instruct them to receive data, transmit data or perform some other operations [5]. Data words contain the actual information to be transmitted. Both BC and RT can send a data word. MIL-STD-1553 allows a maximum of 32 data words in each message. Status words are transmitted by a remote terminal in response to an error free message transfer, after a time period called RT response time. Status words give information about the condition of RT, errors in the received words or a request by RT [8]. Status words are only transmitted by RTs after receiving a command from a BC [8]. The word formats for the command, data, and status words are shown in the figure 3[5].

C. Message Formats

Each message transfer over 1553 data bus includes a CW, DWs and a SW. There are different kinds of data transfer in 1553, each of which has a defined format in this protocol. A single transfer may contain a maximum of 32 messages. After each message there will be a time gap called Inter Message Gap (IMG) [8]. There are various modes of information transfer defined in this standard. Here we consider three important data transfers, which are BC to RT transfer, RT to BC transfer, RT to RT transfers. Each of these follows a pre-defined format in 1553 protocol, which is shown in figure 2[5].

For BC-RT transfer, the bus controller issues a receive Command Word followed by the specified number of Data Words, the number of DWs is specified in the Data Word Count of the command word. The RT after checking the received word, transmit a status word back to the controller, if there is no error. RT will take a time of RT_response time to generate a SW [5]. The will be no gaps between CW and DWs [1].

For RT-BC transfer the bus controller issues a transmit command to the RT [5]. It is received by all the RTs connected to the bus wire. After decoding the CW, the particular RT whose address is specified in the CW will transmit a status word back to the bus controller, followed by the specified number of data words, whose number is specified in Data Word Count of the command word. The status and data words shall be transmitted with no gaps between them [1].

Consider the RT-RT transfer between two RTs, say RT A and RT B. The bus controller shall issue a receive command to RT A followed by a transmit command to RT B without any gap between them [5]. After decoding the CW, RT B transmits a status word followed by the specified number of data words, the number of DWs is specified in the Data Word Count of the command word. There will be no gaps between status and data words transmitted by the RT B. After all the data transmission by RT B, RT A shall transmit a status word after a time, say RT_response time, which indicates the status of the previous message transfer, status of RT and also errors in the last transmission if any [1].

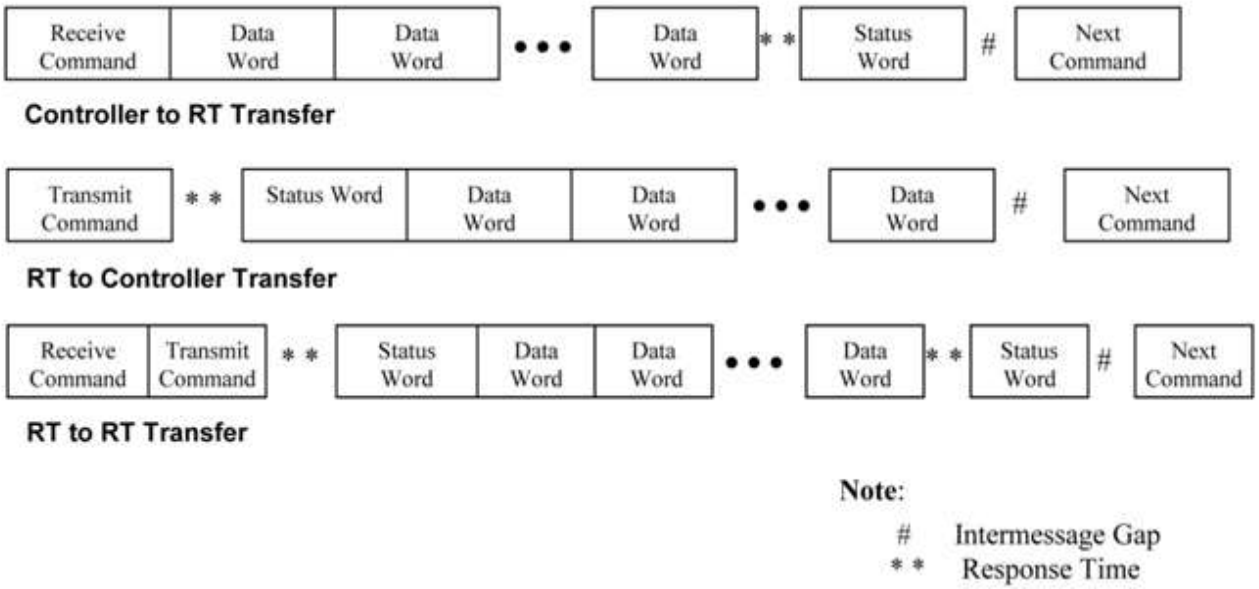

Figure 2. Message formats [5]

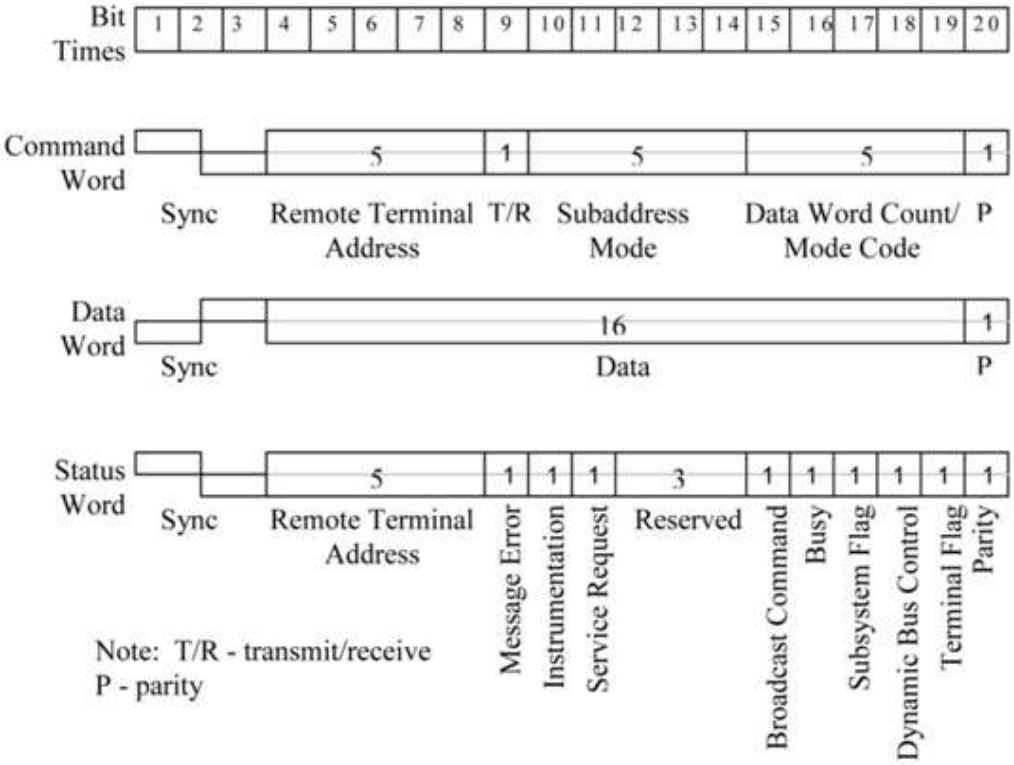

Figure 3. Word formats [5]

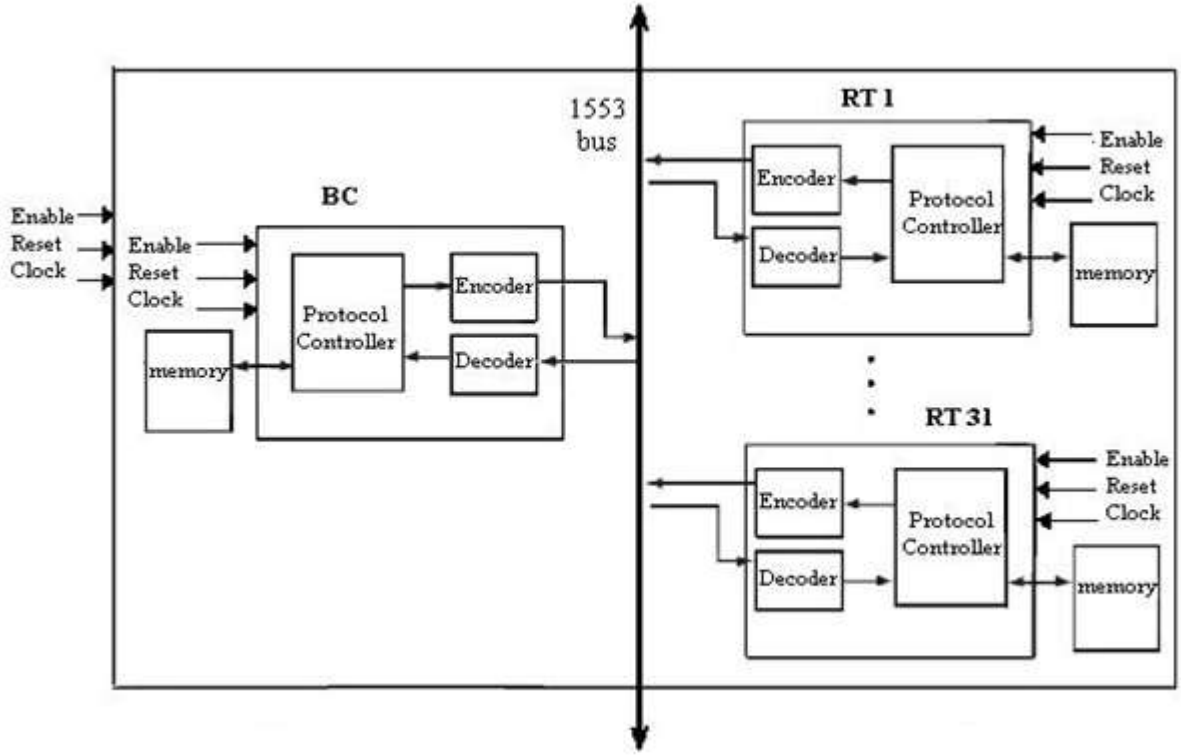

Figure 4. Block diagram of 1553 bus system

III. Design& Implementation

Figure 4 shows a generalized block schematic representation of the 1553 system in a Bus Controller or in a Remote Terminal. In the bus system there will be a BC and a number of RTs. All the systems which are connected to the bus wire for sharing information will act as a Remote terminal other than it is specially designed as the Bus Controller. Only one system connected to the 1553 bus wire can be programmed as BC and it controls the entire operations. All the internal blocks are same for BC and RT except the protocol controller.

The BC protocol controller is associated with entire control of bus system and the protocol checks, whereas an RT controller is associated with data transfers and the Status words. Both use same encoder/ decoder system. The function of encoder is to convert the serial data output from the controller to Manchester format before transmitting it to the bus. Decoder takes each bit from the 1553 bus and decodes it from the Manchester format to normal bits. The design of each of the modules used is explained below:

A. Design of Manchester Encoder- Decoder

MIL-STD-1553 uses Manchester II Bi-phase encoding for all the information transmitted through the bus wire. Manchester encoding has got a lot of advantages. It provides a self-clocking waveform, i.e., the clock is embedded in the data [9]. So recovering the clock at the receiver side will be easier. In Manchester encoding there are two conventions used for representing encoded data. The one we are following is described here: a low-to-high transition is used to encode a "zero" and a high-to-low transition is used to encode a "one" [10]. So a bit in Manchester encoded data has a transition in it, either from low to high or vice versa.

As explained in section II B, each word in 1553, whether it is DW, CW or SW, follows a particular format, i.e., a 3 bit sync pattern, followed by 16 bits of information and 1 parity bit. The Manchester II bi-phase encoder and decoder used for the data bus should consider the sync pattern, data bits and parity bit separately.

The sync field is a 1.5 bit periods high pulse followed by a 1.5 bit periods low pulse for CW and SW, and for DW it is a 1.5 bit periods low pulse followed by a 1.5 bit periods high pulse. The sync field has to be considered as an invalid Manchester encoded form because there is no transition within a bit period [9]. While decoding, decoder understands the type of data from the sync field. And the decoded information does not have this field since this is removed by decoder.

Data bits come after the sync field, each of which occupies a single bit period. The data bits are encoded as half bit period high and other half low. In Manchester encoding a low-to-high transition is used to encode a "zero" and a high-to-low transition is used to encode a "one". The encoded data bit portion has same frequency as the original signal. The encoding of sync field and data field is shown in figure 5 and 6 respectively [10].

The last bit in all the words transmitted via 1553 bus system is the parity bit. Odd parity is used here. This parity bit can detect an error in the transmitted word. The encoder will encode this parity bit too in the Manchester format, i.e., either with a half bit high followed by another half low or with a half

bit low followed by next half high for a parity bit of "one" and "zero" respectively. This bit is sent as the 20th bit of each word. At the decoder the reverse operation takes place, i.e., it checks whether the parity is correct or not. Decoder finds the parity of all the coming bits in a word excluding the sync field. Then it compares the obtained parity bit with the coming parity bit. If there is any parity mismatch found then the decoder sends that information to controller and requests for retransmission of that entire word.

The Figure 5 shows how the sync field of a CW, SW and a DW are generated by a Manchester encoder in the 1553 bus system. Bit period is shown at the top. As we have explained earlier the entire sync field occupies three bit periods, half of which is high and next half low or vice versa. Figure 6 shows the Manchester encoding of normal bits other than sync. Both data bits and parity bits are encoded in the same format, i.e., each bit occupies a single bit period with a transition in it. It shows the Manchester encoding of both "one" and "zero" bits for a period of three bit times. The main advantage of using such an encoding scheme is that we are getting a transition in each of the bit times, except for sync field which is an invalid Manchester encoding, thereby increasing the easiness of detecting errors if any occurs in the information transmitted.

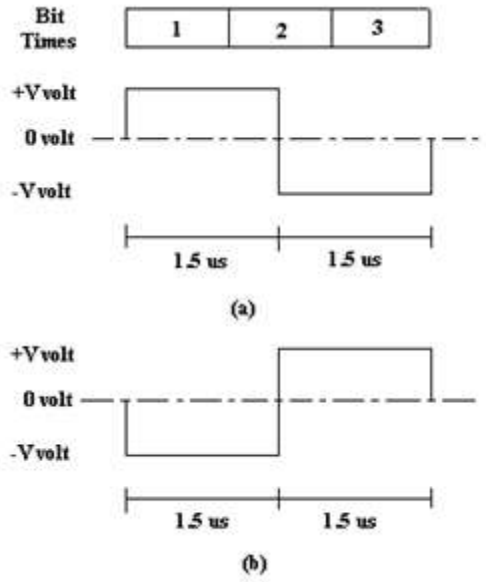

Figure 5. Manchester encoding of (a) Command & Status sync (b) Data sync

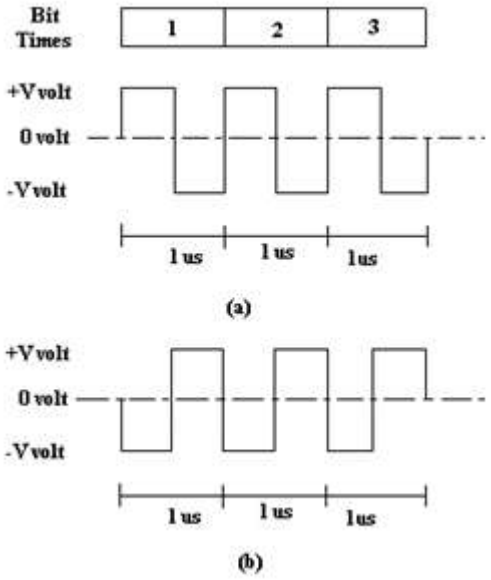

Figure 6. Manchester encoding of the sequence (a) "111" (b) "000"

B. Design of BC and RT

Detailed block schematic representation of BC and RT are shown in figure 7 and figure 8 respectively. First consider a BC block. The two blocks, named as descriptor and memory, indicates storage locations, a descriptor module for storing the control words and a memory module for storing data words. In addition to these two modules there is one encoder, one decoder and a protocol controller. The encoder used is a Manchester II bi-phase encoder, which encodes each bit of the information to be transmitted via 1553 data bus in to a Manchester format. The decoder has the opposite function of decoding the Manchester encoded data bits coming via 1553 data bus. So all the devices connected to the bus wire for sharing information should include a Manchester encoder and decoder since we are following the Manchester encoding scheme for 1553 data bus protocol. The encoder attaches a parity bit to each word which is to be transmitted via 1553 bus wire. The parity error check is performed by the decoder. And the most important component is a protocol controller, which performs all the operations related to the protocol. The main function of BC protocol controller is to issue CWs to RTs, only a CW can initiate any of the message transfers in a 1553 protocol. Also it performs all the protocol checks, monitors the entire bus system, checks the status of all RTs and all message transfers and a lot more. In short the BC protocol controller is the heart of 1553 bus system which controls the entire operations.

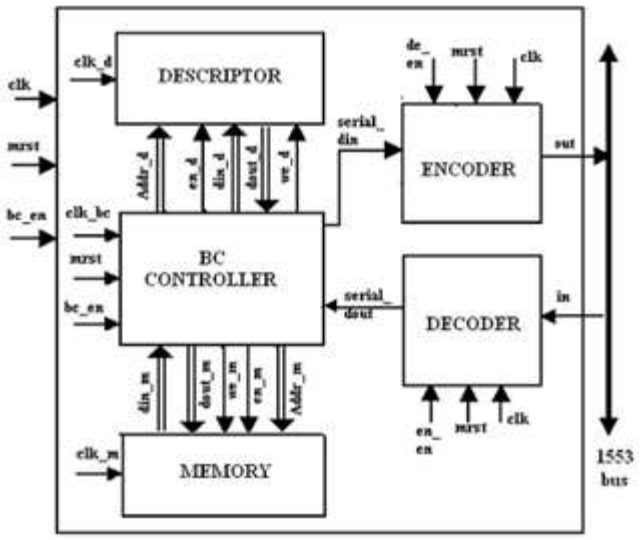

Figure 7. Design of 1553 BC

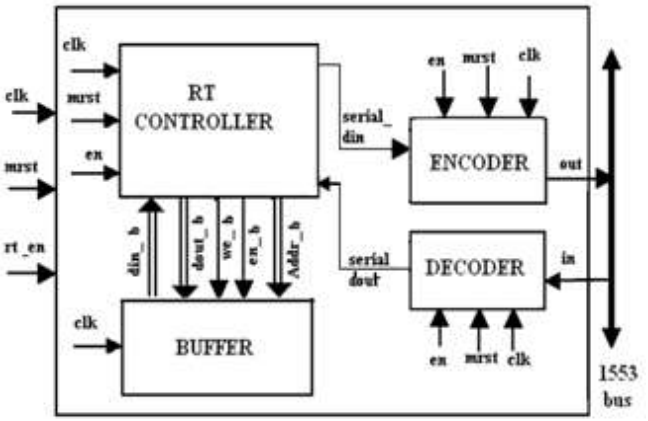

Figure 8. Design of 1553 RT

In the case of an RT it has a single memory module, named as Buffer in figure 8, since it needs to send only DWs. This memory block stores data words to be transmitted by the RT and the DWs received by the RT. Along with the memory unit an RT block ha an encoder, a decoder and a protocol controller. The encoder and decoder are same as we have explained in the case of BC block diagram. The encoder attaches a parity bit to each word which is to be transmitted via 1553 bus wire. The parity error check is performed by the decoder. The RT protocol controller performs all the functions that an RT must handle, i.e., it decodes the CWs sent by the BC, either receives DWs from BC or transmits DWs to the BC according to BC commands, and acknowledges each message transfer with a SW which indicates the status of RT, whether any error present in the previous message transfer, etc.

All the blocks are designed using state machine modeling in VHDL. State diagrams of BC controller and RT controller are given in figure 9 and figure 10 respectively. MSGCMD and Dataptr [11] in the figure 9 indicate the control words for BC system. As we have explained first BC sends a CW, which is common for all the three types of message transfers. The next states of the diagram are different for each type of the transfer. And this state machine strictly follows the protocol rules of 1553. As we can send a maximum of 32 DWs only in a single message, there is a counter for counting data words which checks whether the count exceeds 32. Similarly for messages, there will be a message counter which checks the number of messages, which should not be more than 32 in a single transfer.

State diagram of RT is given in figure 10. It performs the functions that an RT must handle in a 1553 protocol. An RT should receive the CW first, should be able to decode it, then either receive or transmit DWs and finally should sent back a SW acknowledgement back to the BC. The state diagram in figure 10 follows all these functions of an RT. As we explained for BC, we are using a data counter and message counter in RTs also.

C. Timing diagram of BC

Bus controller is associated with three types of transfers: BC-RT transfer, RT-BC transfer and RT-RT transfer. Each of these message transfers is initiated by a CW from BC. Each transfer may contain a maximum of 32 Data Words and at least one Status Word indicating the status of that message transfer. And a maximum of 32 such message transfers are allowed in each communication. There are pre-defined formats for each of the above said transfers. BC controller's state diagram follows the same formats.

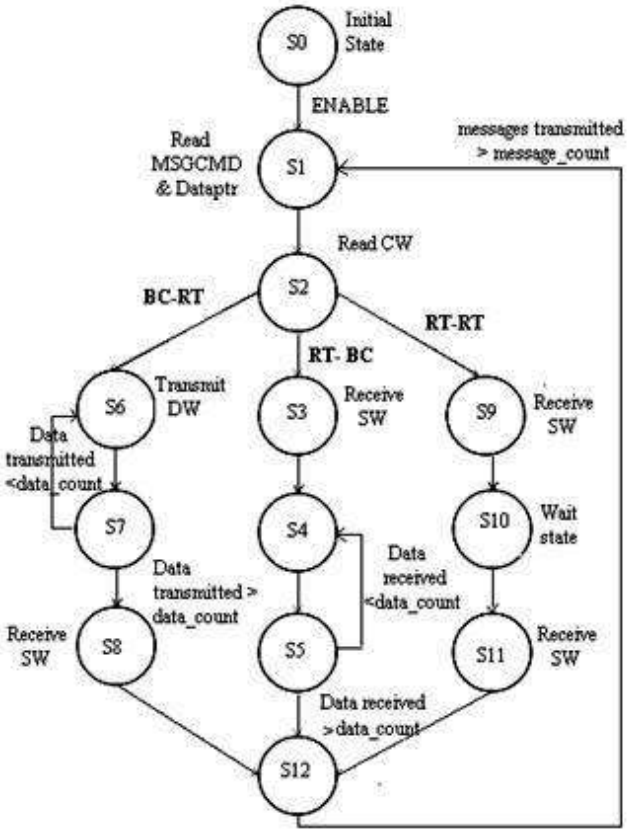

Figure 9. State diagram of 1553 BC

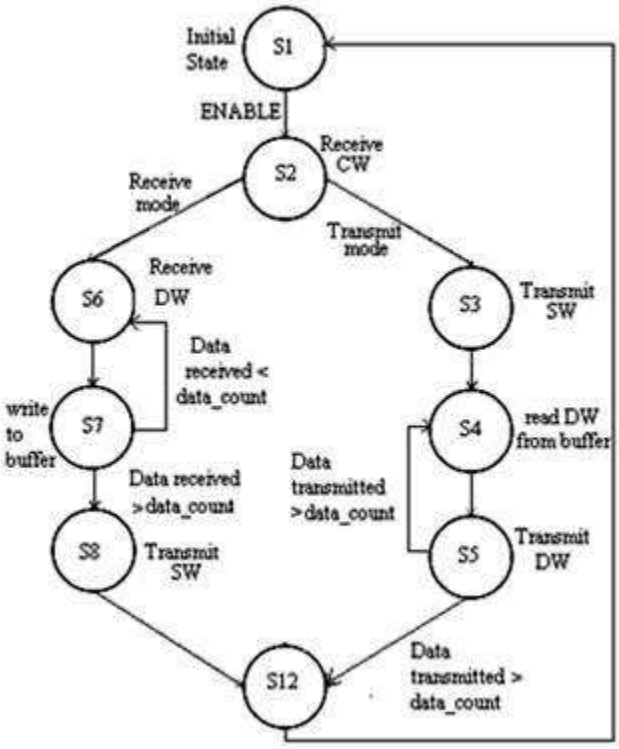

Figure 10. State diagram of 1553 RT

Similarly RT's function is to act according to the commands of Bus Controller, i.e., to receive or transmit the data words according to the command word sent by BC. After receiving DWs, the RT prepares a SW indicating the status of the last message transfer and sends it to the BC.

For all message transfers, first the BC sends a CW; which initiates the message transfer in 1553 data bus. It may be either transmit CW or receive CW according to the type of transfer. First we will discuss the BC-RT transfer. Here the controller sends a CW to the bus via BC_out (shown in figure 11), which is followed by continuous DWs (a maximum of 32 DWs are possible). The initial CW carries the address of the RT to which BC wish to communicate. All RTs decode the CW and only the particular RT whose address is present in the CW receives the following DWs. After receiving the CWs and DWs the RT sends the SW as acknowledgement, which is received in BC_in signal, which is also shown in figure 11.

The RT-BC transfer is also initiated with a BC Command Word via BC_out. The RT whose address is specified in the CW, after receiving it, acknowledges by sending a SW. This is followed by continuous DWs to the BC, which are received serially by BC via BC_in as shown in figure 12.

For RT-RT transfer, two CWs and two SWs are to be transmitted. First BC sends a receive CW to the RT which should receive data followed by a transmit CW to the RT which should transmit data as shown in BC_out in figure 13. In response the transmitting RT sends back a SW to BC and after transmission the receiving RT also sends a SW, which are received by BC via BC_in as shown in figure 13.

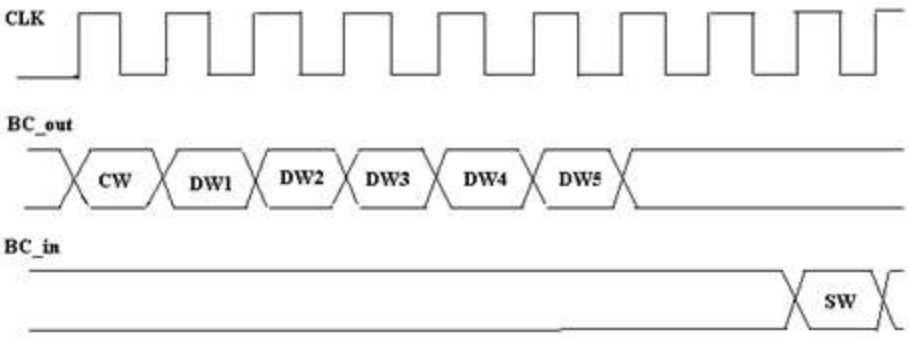

Figure 11. Timing diagram of Bus controller (BC-RT transfer)

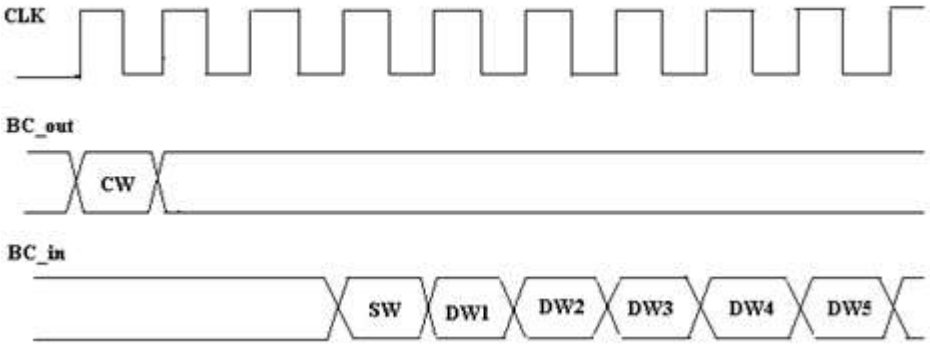

Figure 12. Timing diagram of Bus controller (RT-BC transfer)

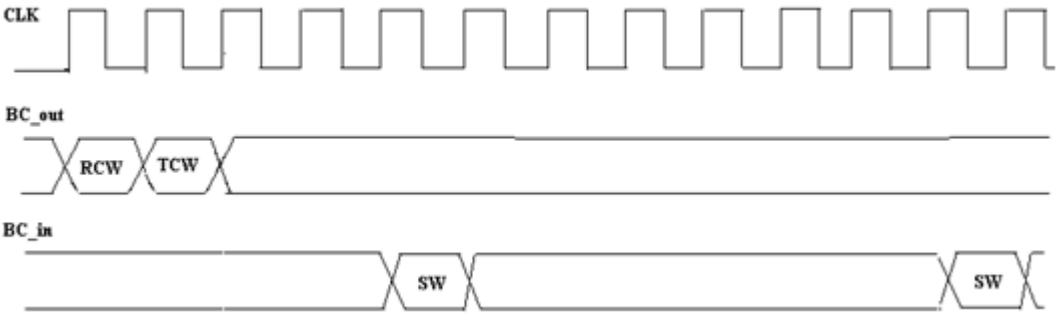

Figure 13. Timing diagram of Bus controller (RT-RT transfer)

D. Timing diagram of RT

The function of Remote terminal is to take part in the communication according to the commands of Bus Controller. An RT can either send Data Words, can receive Data Words and can send a Status Word. The Remote Terminal can also support all the three transfers we discussed in section B. We will discuss them one by one. All the transfers in 1553 bus system are initiated a BC Command Word, which is received by all the RTs connected in the bus system via RT_in.

First consider BC-RT transfer. RT receives the CW and the DWs, transmitted by BC, via RT_in. After receiving all words the RT sends the SW as acknowledgement to the bus system via RT_out as shown in figure 14. For RT-BC transfer, the RT, after receiving the CW via RT_in, acknowledges by sending a SW which is followed by continuous DWs to the BC via RT_out as shown in figure 15. For RT-RT transfer, two CWs and two SWs are involved. First RT receives a receive CW via RT1_in, second RT receives a transmit CW via RT2_in. In response the transmitting RT, i.e., RT2 sends back a SW followed by DWs via RT2_out to the bus system which will be received by RT1 via RT1_in. And after receiving entire words, the receiving RT, i.e., RT1 sends back a SW to the BC via RT1_out as shown in figure 16.

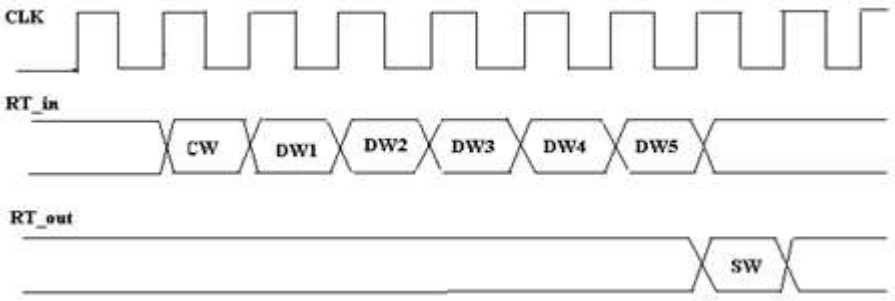

Figure 14. Timing diagram of Remote terminal (BC-RT transfer)

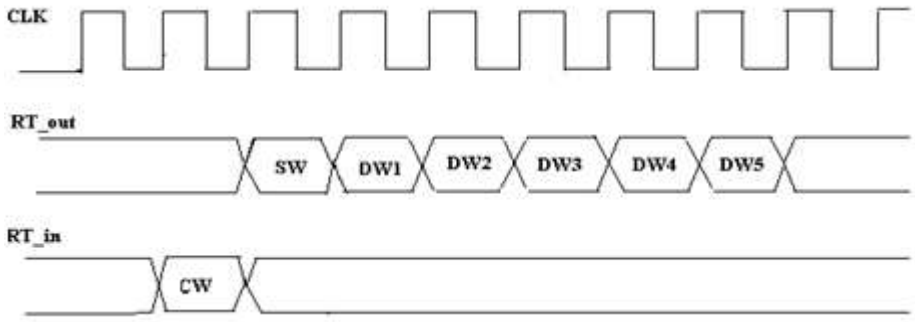

Figure 15. Timing diagram of Remote terminal (RT-BC transfer)

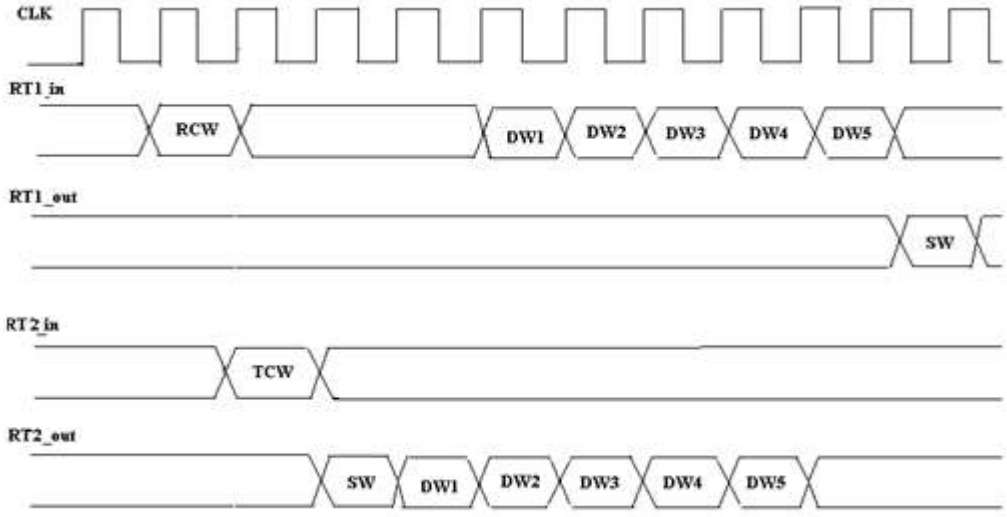

Figure 16. Timing diagram of Remote terminal (RT-RT transfer)

E. Implementation

The design of the system including the Manchester encoder, decoder, BC protocol controller and RT protocol controller are to be carried out using VHDL state machine modeling, and required memory modules for BC and RT are generated using Xilinx Core Generator tool for optimized operation in Xilinx FPGAs. The entire code is downloaded and implemented on a Xilinx Spartan-2 FPGA (xc2s200-5pq208) using Xilinx ISE tool. The target device utilization of the proposed design is shown in figure 17. It is clear from the same that the device utilization is below 25%. Also by using a platform like Spartan II, a cost effective way to implement the designed system can be provided. And FPGA implementation of 1553 BC and RT provides a flexible core, which when integrated to the systems that need to share information, provide a highly reliable data transfer method between critical systems in an aircraft.

IV. Results

The design described in the previous section has been simulated using ModelSim. The proposed system has successfully worked as 1553 bus system, which provided information transfer between BC and RTs in the Manchester II bi-phase encoded format. So a test result that is matching the expected timing diagrams is obtained. Then the design is synthesized using Xilinx ISE. The target system selected was a Xilinx Spartan II FPGA (xc2s200-5pq208). Finally the resulting bit streams have been downloaded onto the FPGA platform in order to verify the design and the expected results were obtained. The target device utilization by the design is given in figure 17. It is clear from the figure that only 25% of the available LUTs in the target device are occupied by the proposed design. So this provides area efficient implementations for the 1553 bus system including protocol controller for BC and RT, Manchester encoder and decoder and the required memory modules. Compared to many other designs available in market the area utilization of the proposed design is comparatively much less, which can be considered as an advantage.

V. Conclusion & Future Works

This paper describes an approach to implement the military standard 1553 data bus protocol onto a Xilinx based FPGA platform. Being a widely used data bus standard in avionics, it will be a good idea if we could provide an inexpensive option for the same. Here in this paper this could be achieved using a relatively inexpensive ordinary FPGA. The system design is done using state machine modeling in Hardware Description Language (HDL). Tool used for testing and simulation is ModelSim and the design has been downloaded onto a Xilinx

Spartan FPGA kit using Xilinx ISE. The target system selected was a Xilinx Spartan II FPGA (xc2s200-5pq208).

After implementation we could say that the area utilization of the proposed design is much less. Also as the FPGA can be reconfigured easily, the flexibility of design will be more. Since this standard has only a limited data rate of 1 Mega bits per second, the future work for this design will be the speed improvements of the standard and integration of new technologies into this. Even with the recent developments of newer and higher-speed technologies, 1553 is used for data transfer between mission critical systems of an aircraft where the reliability is of more importance than speed. Also some of today's modern aircrafts use a mix of high-performance data buses and 1553. So it is clear that 1553 will continue its journey in the new applications and integration platforms for years to come.

Device Utilization Summary			
Logic Utilization	Used	Available	Utilization
Number of Slice Flip Flops	494	4,704	10%
Number of 4 input LUTs	1,054	4,704	22%
Logic Distribution			
Number of occupied Slices	599	2,352	25%
Number of Slices containing only related logic	599	599	100%
Number of Slices containing unrelated logic	0	599	0%
Total Number of 4 input LUTs	1,054	4,704	22%
Number of bonded IOBs	3	140	2%
Number of GCLKs	1	4	25%
Number of GCLKIOBs	1	4	25%
Total equivalent gate count for design	10,423		
Additional JTAG gate count for IOBs	192		

Figure 17. Target device utilization by designed system

References

[1] Jemti Jose and Sharone Varghese, "Design of 1553 protocol controller for reliable data transfer in aircrafts", In *Proceedings of the 12th International Conference on Intelligent Systems Design and Applications* (ISDA 2012), Cochin, India, pp 686-691.

[2] Carey B. R (2007, Dec), Avionics magazine, [online]. Available: http://www.AvionicsToday.com

[3] MIL-STD-1553A Multiplex Applications Handbook, 1988, Washington, D.C., Department of Defense.

[4] Kobus van Rooyen, "FPGA based MIL-STD- 1553B for new aircraft and mid-life upgrades", Data week electronics and communication technology magazine, May 2003.

[5] J. Furgerson, "MIL-STD-1553 Tutorial", AIM-Avionics Databus Solutions, U.K, November 2010.

[6] Military aerospace magazine, (2010, May 20), [online]. Available: http://www.militaryaerospace.com

[7] MIL-STD-1553B, Aircraft Internal Time- Divison Multiplexing Data Bus, Washington, D.C., Department of Defense, 1978.

[8] MIL-STD-1553 Designer's Guide, Data Device Corporation, Bohemia, NY, 2003.

[9] Mil-std-1553 tutorial [online]. Available: http://mil-std-1553.org/ManchesterTutorial.html

[10] Jemti Jose, "Design of Manchester II bi-phase Encoder for MIL-STD-1553 Protocol", In *Proceedings of the IEEE International Multi Conference on Automation, Computing, Control, Communication and Compressed Sensing* (iMac4s), Kerala, India, March 2013, in press (not indexed).

[11] Core1553BBC- MIL-STD-1553B Bus Controller, Actel Corporation, CA, USA, December 2005.

Author Biography

Jemti Jose was born in Kerala, India, in 1987. She received her Bachelor of Technology (B.Tech) degree in Electronics and Communication Engineering from Mahatma Gandhi University, Kottayam, India, in 2009 and her Master of Technology (M.Tech) in VLSI and Embedded Systems from the Cochin University of Science and Technology (CUSAT), Cochin, India in 2012. She is now Assistant Professor in St. Joseph's College of Engineering and Technology Palai, India. Her main areas of interest are Design of VLSI systems, VLSI Signal Processing and FPGA based Design.