Hardware implementation of the CloudBus protocol using FPGA

Kazimierz Krzywicki
Faculty of Electrical Engineering, Computer Science and Telecommunications, University of Zielona Gora
Zielona Gora, Poland
k.krzywicki@weit.uz.zgora.pl

Grzegorz Andrzejewski
Faculty of Electrical Engineering, Computer Science and Telecommunications, University of Zielona Gora
Zielona Gora, Poland
g.andrzejewski@iie.uz.zgora.pl

***Abstract*— In this paper, a CloudBus protocol for distributed embedded systems is presented. It provides a control mechanism for a number of processing units distributed in a network. The paper demonstrates a hardware implementation of a CloudBus protocol using FPGA. Furthermore, it considers the resource usage depending on the FPGA platform.**

***Index Terms*—Embedded Systems, Distributed Systems, FPGA, CloudBus protocol**

I. INTRODUCTION

Large and complex distributed embedded systems are difficult to design and implementation [1, 3, 6]. Moreover process synchronization in such systems are also complicated which often results in high load of the communication interfaces. CloudBus protocol [2, 7] proposed by authors allows to significant saving in the amount of the transmitted data between end modules of the distributed embedded system [4, 5] (especially when compared with the Modbus RTU or Profibus-DP protocol [7]). So far, the CloudBus protocol was implemented and tested on the distributed microcontroller platforms. This paper presents implementation of the CloudBus protocol on the FPGA platform. The implementation and the synthesis [3] was made using Hardware Description Language (HDL) – Verilog, under two different development tools: Xilinx ISE Design Suite 14.7 – synthesis to the following devices: Kintex-7 (XC7K70T), Spartan 3E (XC3S1600E), Virtex-6 (XC6VLX75T); Quartus II 13.1 – synthesis to the following devices: Arria II GX (EP2AGX45DF29C4), Cyclone IV E (EP4CE 115F23I8L), Cyclone V (5CGXFC7D7F27C8).

Comparison of the resource usage was made for the different destination devices and development tools.

In Section II CloudBus protocol is presented, Section III presents CloudBus protocol implementation with internal modules description. Section IV discuss research results for different FPGA devices. Section V, concludes the paper.

II. CLOUDBUS PROTOCOL

Presented CloudBus [2, 7] protocol (Fig.1) is one of the methods of the data exchange and concurrent process synchronization in the distributed systems. It realizes decentralized (distributed) control method, where all of the devices (modules), in designed system, are equal to each other. CloudBus is based on the dependence, that, data are only transferred between modules, when one of modules needs information (shared resource) from the outside of its own resource variables. Module sends (broadcasts) question to other modules about the state of the specified variable, e.g. (*if y == 1*?). The module which is responsible for this variable, sends the answer to the system, if and only if quested variable get previously quested state. This model of the communication, allows to significant savings in the amount of the transmitted data.

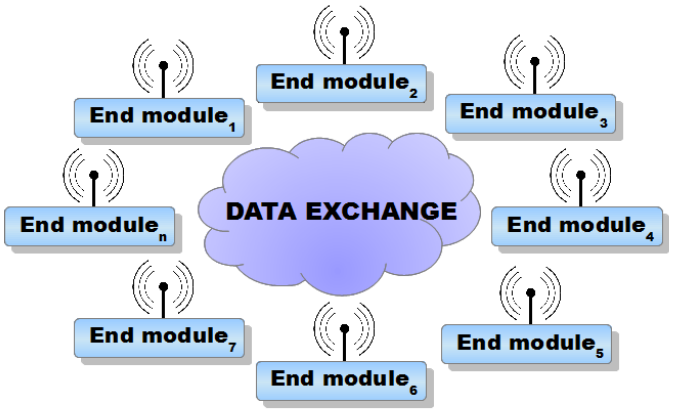

Fig. 1. Schematic diagram of the CloudBus protocol.

Basic data frame of the CloudBus protocol is shown in Table I.

TABLE I. CLOUDBUS PROTOCOL FRAME

CNT	FUNC	VARS	DATA	CRC

Fields of the protocol corresponds to: *CNT* – 1 byte for entire frame length; *FUNC* – command code (e.g. question about condition or simple answer to other of the modules); *VARS* and *DATA* represents binary array of the variables and their states (values); CRC – 1 byte of the CRC error checksum.

III. IMPLEMENTATION

The implementation of the CloudBus protocol in the FPGA devices required dividing system functionality into dedicated modules. Designed modules are shown in Fig. 2.

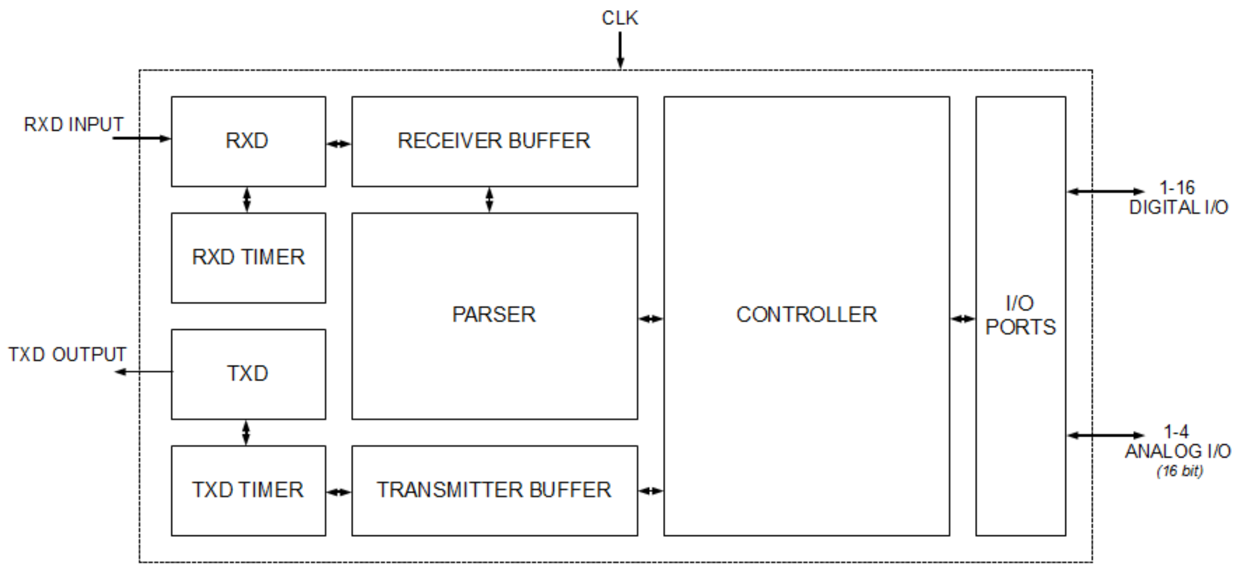

Fig. 2. Schematic diagram of the end module unit.

End module features:

- 16 digital inputs/outputs
- 4 analog inputs/outputs (16 bit)
- serial communication interface
- external clock input

A. *RXD TIMER module*

RXD TIMER module (Fig. 3) is double timer/counter. First counter (*clkHi*) with higher frequency is used for the sampling RXD INPUT line, second (slower) is used for the bit read from RXD INPUT line. *clkMain* input is external CLK clock input. *hiRst* and *loRst* are timer reset inputs.

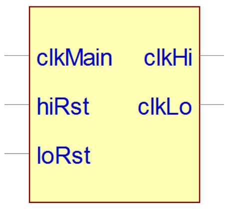

Fig. 3. Schematic diagram of the RXD TIMER module.

B. *RXD module*

RXD module (Fig. 4) is responsible for incoming communication from the outside of the end module. It retrieves data from RXD INPUT by input *RxD* line. Input line is sampled with *clkHi* clock frequency to check for incoming transmission. When it detects incoming data, *clkLo* clock counter is started for data read from *RxD* line and data bits are saved to *data* register.

Furthermore, RXD module can reset timer by loRst output or inform by received line about completed successful packet receiving. Error line, delivers information about corruption of the received data (parity/CRC checksum error).

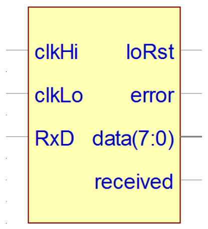

Fig. 4. Schematic diagram of the RXD module.

C. *RECEIVER BUFFER module*

RECEIVER BUFFER module (Fig. 5) is 128-bit data buffer for received data by RXD module. It merges all single bytes to entire frame of the CloudBus protocol. Module is synchronized by *clk* clock. Data incoming by 8-bit *in* input and outgoing to PARSER module by 128-bit *out* output. Additional *reset* input for resetting module and *ready* output which gives information when buffer is full.

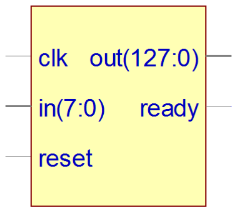

Fig. 5. Schematic diagram of the RECEIVER BUFFER module.

D. *PARSER module*

PARSER module (Fig. 6) is synchronized by *clk* input. Data are received from RECEIVER BUFFER by 128-bit *in* input. PARSRER module is responsible for parsing received frame of the CloudBus protocol from RECEIVER BUFFER. When parsing is done, without any errors (frame length check and CRC checksum check) valid CloudBus data are set to outputs: *func*, *vars*, *digitalIO*, *anagalogIO* else *error* output is driven high, which means that received CloudBus frame is corrupted.

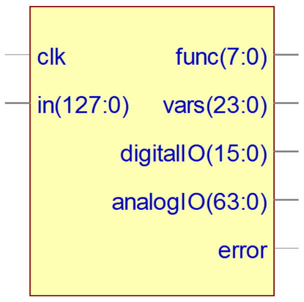

Fig. 6. Schematic diagram of the PARSER module.

E. *CONTROLLER* with *I/O PORT module*

CONTROLLER module (Fig. 7) is the most important module. It is responsible for implementing previously designed control algorithm and for the communication with other end modules via CloudBus protocol. CONTROLLER module is synchronized by *clk* input and it gets data (*func*, *vars*, *digitalIO*, *analogIO*) from PARSER module. *Error* input delivers information about correctness of the incoming data. Furthermore CONTROLLER module controls the 16 digital inputs/output and 4 analog inputs/outputs (16-bit each of I/O). Outside data transmission is made by *dataOut*, *sendData* outputs. *DataOut* carries data to transmit by TXD module. *SendData* triggers when data are ready to send.

For the research comparison CONTROLLER module does not perform any control algorithm. It is important, because of the different FPGAs architectures.

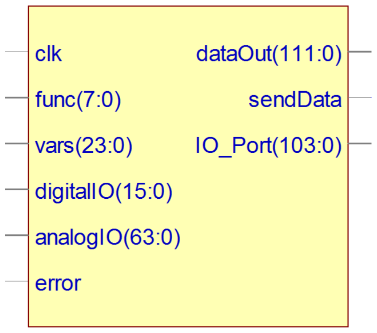

Fig. 7. Schematic diagram of the CONTROLLER module.

F. *TRANSMITTER BUFFER module*

TRANSMITTER BUFFER module (Fig. 8) preparing data and encoding entire frame for TXD module. Module also counts frame length and CRC checksum of CloudBus protocol frame. TRANSMITTER BUFFER is synchronized by *clk* clock, *data* input corresponds to data received from CONTROLLER module (data contains CloudBus protocol commands). *Reset* input clears TRANSMITTER BUFFER. Two outputs connected with TXD module transfers byte to send (*byteForTransmit*) and ready to send signal (*readyToSend*).

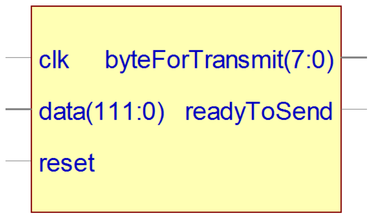

Fig. 8. Schematic diagram of the TRANSMITTER BUFFER module.

G. *TXD TIMER module*

TXD TIMER module (Fig. 9) generates (*clkLO*) sending clock for TXD module. *clkMain* input is external CLK input. *loRst* is timer reset input.

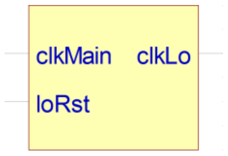

Fig. 9. Schematic diagram of the TXD TIMER module.

H. *TXD module*

TXD module (Fig. 10) is responsible for data transmission on *TxD* output line – outside of the end module. It takes 3 inputs: *clkLo* – sending clock, *send* – signal which starts byte transmission and *data* – byte to send. Outputs are: *TxD* which is connected with TXD OUTPUT as external transmission line and *loRST* for counter/timer reset.

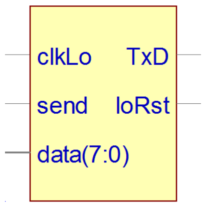

Fig. 10. Schematic diagram of the TXD module.

IV. RESEARCH RESULTS

The implementation and the synthesis was made using two different development tools: Xilinx ISE Design Suite 14.7 – synthesis to the following devices: Kintex-7 (XC7K70T), Spartan 3E (XC3S1600E), Virtex-6 (XC6VLX75T); Quartus II 13.1 – synthesis to the following devices: Arria II GX (EP2AGX45DF29C4), Cyclone IV E (EP4CE 115F23I8L), Cyclone V (5CGXFC7D7F27C8). Source code for both development platforms and for all devices was customized to be universal (all devices used same source code).

Results of the synthesis for Xilinx devices of each module is shown in Table II. Maximum percent of total register (*Reg.*) resource usage was noted for Spartan 3E (XC3S1600E), percent of the maximum occupied slices (*Slices*) and LUTs (*LUTs*) for the Kintex-7 (XC7K70T). Furthermore, minimum percent of resource usage was noted for Virtex-6 (XC6VLX75T) device.

TABLE II. RESOURCE USAGE FOR THE XILINX DEVICES

Implemented modules	Device								
	Kintex-7 (XC7K70T)			*Spartan 3E (XC3S1600E)*			*Virtex-6 (XC6VLX75T)*		
	Reg.	*Slices*	*LUTs*	*Reg.*	*Slices*	*LUTs*	*Reg.*	*Slices*	*LUTs*
RXD TIMER	21	14	41	21	17	29	21	14	47
RXD	31	14	28	32	33	41	31	13	28
RECEIVER BUFFER	266	93	289	266	199	264	265	127	289
PARSER	112	170	364	0	37	46	0	18	28
CONTROLLER	0	1	2	0	2	3	0	1	2
TRANSMITTER BUFFER	9	2	8	9	6	9	9	3	8
TXD TIMER	12	8	29	12	10	16	12	11	29
TXD	8	9	15	8	12	19	8	7	17
Total used	459	311	776	348	316	427	346	194	448
Total available	82000	10250	41000	29504	14752	29504	93120	11640	46560
Usage [%]	0,56	3,03	1,89	1,18	2,14	1,45	0,37	1,67	0,96

Results of the synthesis for Altera devices of each module is shown in Table III. Maximum percent of total register (*Reg.*) resource usage and percent of the maximum occupied LUTs (*LUTs*) was noted for Arria II GX (EP2AGX 45DF29C4). Minimum percent of total register (*Reg.*) resource usage was noted for Cyclone V (5CGXF C7D7F27C8). Minimum percent of LUTs (*LUTs*) for both Cyclone IV and Cyclone V devices.

TABLE III. RESOURCE USAGE FOR THE ALTERA DEVICES

Modules	Device					
	Arria II GX (EP2AGX 45DF29C4)		*Cyclone IV E (EP4CE 115F23I8L)*		*Cyclone V (5CGXF C7D7F27C8)*	
	Reg.	*LUTs*	*Reg.*	*LUTs*	*Reg.*	*ALMs*
RXD TIMER	21	28	21	28	21	18
RXD	36	35	32	47	36	23
RECEIVER BUFFER	262	152	262	275	263	140
PARSER	113	32	113	155	113	73
CONTROLLER	3	2	3	3	3	2
TRANSMITTER BUFFER	9	2	9	9	9	5
TXD TIMER	12	16	12	16	12	11
TXD	8	15	8	17	8	10
Total used	464	282	460	550	465	282
Total available	36100	36100	114480	114480	225920	56480
Usage [%]	1,29	0,78	0,40	0,48	0,21	0,50

All of the presented results, for all of the selected devices for comparison are oscillating around 1-2% of the total available resource usage. Xilinx and Altera devices, compared between each other or even within same manufacturer has got different hardware architecture, so it is impossible to make direct comparison of them. Cheap and small devices like Arria II GX (EP2AGX45DF29C4) and Spartan 3E (XC3S1600E) uses about 1% more of the available resources than other presented platforms.

Average register usage for the Xilinx devices is 0,7%, average for occupied slices is 2,28% and average for used LUTs is about 1,43%. Average register usage for Altera devices is 0,63% and average LUTs usage 0,59%. This comparison allows to make conclusion that in this specified implementation and synthesis, Altera device with Quartus II development tool, gives a little bit more efficient synthesis result that Xilinx ISE.

The most important research results is very low resource usage for all of the devices, after implementing CloudBus protocol. It allows to use approximately 98% of resource e.g. implementing control algorithm.

V. CONCLUSIONS

This paper presented the implementation and synthesis results of the CloudBus protocol for distributed embedded systems on different FPGA platforms. It considers the resource usage depending on the FPGA device and development platform.

Presented research results allows to make conclusion, that implementing CloudBus protocol on the FPGA platform gives negligibly small resource usage. CloudBus protocol implementation almost doesn't limit the implementation of the other control algorithms, on the same field-programmable gate array. This feature is especially important in large and complex embedded systems which needs lot of the resources to preform designed control algorithm.

REFERENCES

[1] M. Adamski, A. Karatkevich and M. Wegrzyn "Design of Embedded Control Systems", Springer, 2005

[2] K. Krzywicki and G. Andrzejewski, "Concurrent process synchronization in distributed systems", Proceedings of the XV International PHD Workshop – OWD 2013, 2013, pp.36-39

[3] J.P. Deschamps, G. Bioul and G. Sutter, "Synthesis of arithmetic circuits: FPGA, ASIC and embedded systems", Wiley, 2006

[4] H. Attiya and J. Welch, "Distributed Computing: Fundamentals, Simulations and Advanced Topics", J. Wiley Interscience, 2004

[5] V. K. Garg, "Elements of Distributed Computing", Wiley&Sons, 2002

[6] L. Shang and N.K. Jha, "Hardware-software co-synthesis of low power real-time distributed embedded systems with dynamically reconfigurable FPGAs", ASP-DAC '02 Proceedings of the 2002 Asia and South Pacific Design Automation Conference, 2002, pp. 345

[7] K. Krzywicki, G. Andrzejewski, "Interfejsy wymiany danych w systemach rozproszonych", in press