CAN Bus Analyser with Utilisation of FPGAs

Radek Holota[1]

Abstract

This article deals with a design of CAN bus analyser. For this purpose the design with utilisation of Field Programmable Gate Arrays (FPGAs) is used. A system structure consisted of basic blocks is introduced and the blocks are described. For the structure description is used VHDL (VHSIC Hardware Description Language).

Introduction

Present conception of control system is based on the principle of decentralised subsystems with a communication channel. Thanks to this arrangement the systems can be easily expanded or modified. There are many standardised 'Fieldbus' protocols based on ISO/OSI model for the communication in open systems. One of them is a CAN bus.

CAN (Controller Area Network) [1], [2] is a serial communications protocol which efficiently supports distributed realtime control with a very high level of security. The CAN protocol was developed in Europe for the use in passenger cars. Through the successful use of CAN in automotive and industrial applications, CAN found its way to the US and other parts of the world. Internationally standardized under ISO 11898. The massive utilisation of this bus is supported by mostly producers of microcomputers which incorporate the CAN bus controllers to their chips.

The bus analysers, which makes possible to watch the signals and messages on bus, are necessary for the effective design of new systems communicated by CAN bus. At present mostly CAN bus analysers are based on CAN controllers [3]. This work engages in the design of analyser based on programmable logic arrays. The main goals of this project were to create program segments in VHDL [4], [5] which describe the structures in FPGA circuits, to design a special CAN bus receiver by using the segments and finally to realise the CAN bus analyser by the interconnection of the receiver and a personal computer.

Background

CAN bus is exactly specified in [1]. Only basic properties, which are necessary for understanding of analyzer's settings and the result interpretation, are outlined in next paragraphs.

Communication protocol

Four types of messages (frames) are defined in CAN communication protocols – Data frame, Remote frame, Error frame and Overflow frame.

A Data frame serves for data transfer from a transmitter to the other nodes. At present two formats of data frames are mainly supported – CAN 2.0A and CAN2.0B. The differences between these formats are mainly in the length of identifier. The formats of messages are shown in Fig. 1.

[1] Ing. Radek HOLOTA
University of West Bohemia
Department of Applied Electronics and Telecommunication
Sedláčkova 15, 306 14 Plzeň, Czech Republic
phone: +420 377 634 231
fax: +420 377 634 202
e-mail: radek.holota@centrum.cz

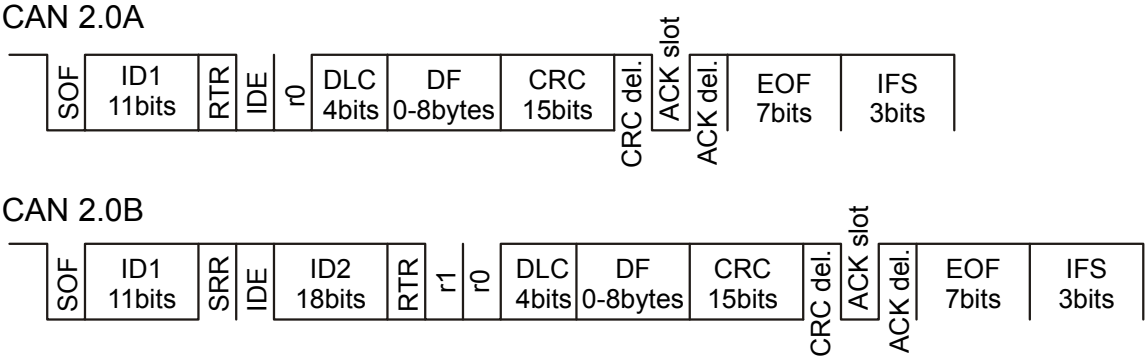

Fig. 1: Structure of data frames – 2.0A and 2.0B

SOF (Start Of Frame) – marks the beginning of data frames or remote frames.
ID1 (IDentifier 1) – is 11 bits message identifier.
SRR (Substitute Remote Request) – is a recesive bit because the standard frame (2.0A) has higher priority than extended (2.0B).
IDE (IDentifier Extension) – identifies the format of data frame (standard or extended).
ID2 (IDentifier 2) – is 18 bits message identifier.
RTR (Remote Transmission Request) – identifies the type of message (data or remote frame).
r1, **r0** – are reserved bits for future utilisation.
DLC (Data Length) – indicates the number of bytes in the data field (4 bits wide).
DF (Data Field) – consists of the data to be transferred within a data frame (0-8 bytes).
CRC (Cyclic Redundancy Code) – is 15 bits cyclic redundancy code for frame checking.
CRC del. –is a CRC delimiter.
ACK (ACKnowledge slot)– is an acknowledgement that data were received.
ACK del. – is an acknowledge delimiter.
EOF (End Of Frame) – is 7 bits field that completes the message.
IFS (Inter Frame Spacing) – is a field between two frames.

A Remote frame has the same structure as the data frame but it hasn't DF (data field). This frame is used for requesting the data from another station (node).

An Error frame is transmitted by any unit on detecting a bus error. It is composed of two fields (Error Flag and Error Delimiter). There are two forms of error flags – active (6 dominant bits) and passive (6 recessive bits). The error delimiter is consisted of 8 recessive bits.

An Overload frame is used to provide for an extra delay between the preceding and the succeeding data or remote frames.

Design

Partial blocks in VHDL

The first goal was to create program segments in VHDL which realises basic structures. In this project following segments were designed: BRP (Baud Rate Prescaler), BTL (Bit Timing Logic), STUFF (stuffing), CRC (CRC checking), ST_MACH (a state machine for message decoding), STACK (a block for storing data to a FIFO memory) and COMM (a state machine for communication with a personal computer).

BRP

A system frequency f_{sys} is obtained from an oscillator frequency f_{osc} by BRP. Their value is programmable in the range <0,31>. A relation between the system frequency and the oscillator frequency is given by equation (1).

$$f_{SYS} = \frac{f_{OSC}}{2*(BRP+1)} \tag{1}$$

BTL

This block performs a bit synchronisation. The ranges of programmable values are: PROP_SEG ∈ ⟨1,8⟩, PHASE_SEG1 ∈ ⟨1,8⟩, PHASE_SEG2 ∈ ⟨1,8⟩ a SJW ∈ ⟨1,4⟩. More information about bit timing configuration is in [6].

STUFF

The bit synchronisation is used in CAN bus communication. For better synchronisation the method of bit stuffing is utilised. This method is based on inserting inverse bit after five consecutive same bits. The STUFF block ignores these inserted bits.

CRC

The message transfer is checked by the CRC block which implements CRC code. The CRC code is generated by polynomial function $x^{15}+x^{14}+x^{10}+x^{8}+x^{7}+x^{4}+x^{3}+1$.

ST_MACH

This block is the state machine which decodes received messages. The outputs are a message code, a data length, the identifiers ID1, ID2 and data.

STACK

The block STACK performs storing of received and decoded messages to the FIFO memory. For description in VHDL the LPM (Library of Parameterizable Modules) function is used and the embedded memory blocks EAB or ESB create this memory.

COMM

The last block COMM ensures the communication between the receiver and PC (Personal Computer). In this case a parallel port is used. The block is designed as a state machine and makes possible to set the parameters of bit synchronisation in accordance with user requirements. The main goal of COMM is reading of stored messages and sending to PC.

Special receiver

After successfully simulation and testing of all designed blocks the special CAN bus receiver block was designed with the utilisation of described blocks. The block diagram is shown in Fig. 2.

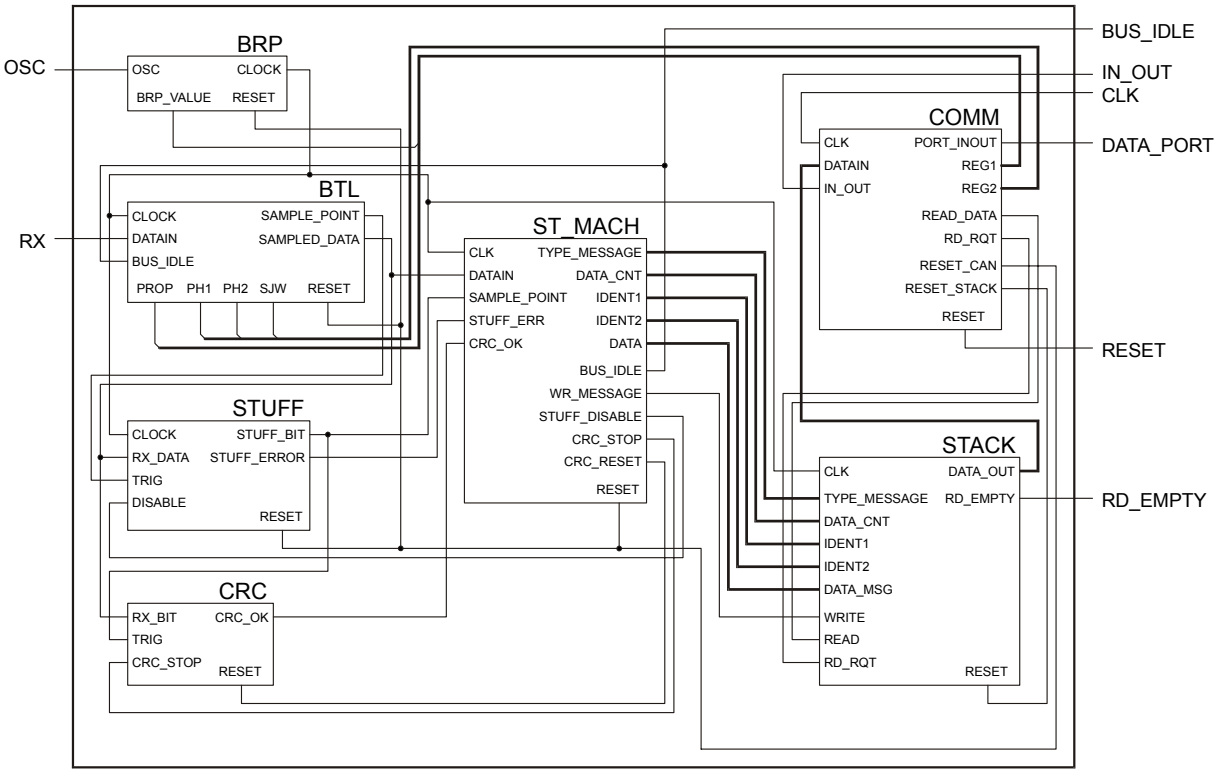

Fig. 2: Special receiver – block diagram

Software - CAN analyser

Finally a program in Microsoft Visual C++ was written. This program forms a powerful CAN bus analyser in cooperation with the special receiver. It makes possible to set the parameters of receiver. During process the decoded messages are read from the receiver and displayed in a program window. A screenshot of the program is shown in Fig. 3.

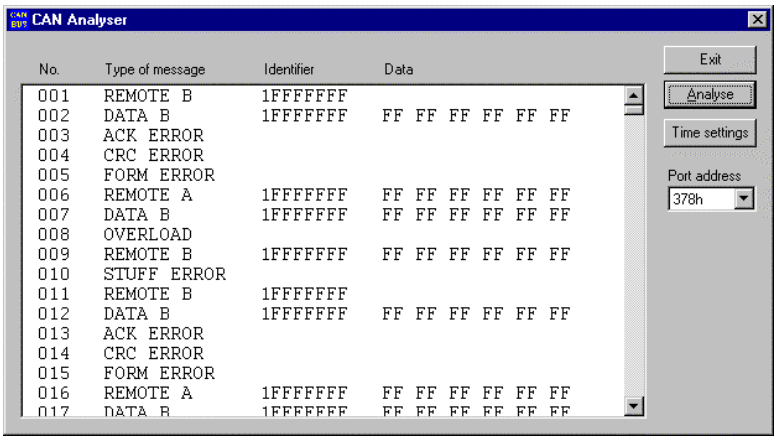

Fig. 3: CAN Analyser – software screenshot

Hardware – CAN Analyser

A general structure of planned hardware system is shown in Fig. 4. The hardware is not fully designed yet. The design of CAN Analyser hardware is a subject of future work.

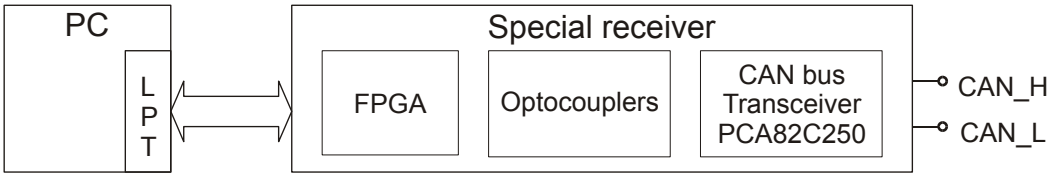

Fig. 4: Structure of planned hardware system

Conclusion

In this project the universal blocks in VHDL were designed and successfully simulated. These blocks can be easily used in next projects. Then the special CAN bus receiver was created with the utilisation of these blocks. The receiver and PC form the CAN bus analyser which can be used for watching and analysing the messages on the bus. This analyser was designed for educational purposes but can be used in industry too.

Acknowledgement

This paper is based upon work sponsored by FRVŠ under project G1/1616/2002.

References

[1] CAN Specification Version 2.0, Robert Bosch GmbH, 1991.

[2] Bagschik P.: An Introduction to CAN, I+ME ACTIA, 1998.

[3] Kostrurik K.: Prostředky ke sledování činnosti sběrnice typu CAN bus, Pilsen, University of West Bohemia, 2000.

[4] Cohen B.: VHDL Coding Styles and Methodologies, 2nd Edition, Kluwer Academic Publishers, USA, 1999.

[5] Salcic Z., Smailagic A.: Digital Systems Design and Prototyping, Second Edition, Kluwer Academic Publishers, USA, 2000.

[6] Hartwich F., Bassemir A.: The Configuration of the CAN Bit Timing, 6th International CAN Conference, Turin.