Implementation of CAN Controller With FPGA Structures

Blagomir Donchev, Marin Hristov

Abstract – **In this article is present a stand-alone controller for the controller area network (CAN) protocol. The design contains all necessary features required to implement a high performance communication protocol. The CAN controller with a simple bus line connection performs all the functions of the physical and data-link layers. The application layer is provided by a microcontroller, to which the CAN controler connects through general purpose non-multiplexed parallel 8-bits bus.**

Keywords — **VHDL, FPGA, CAN, Microcontrollers, MBCAN.**

I. Introduction

The CAN communications protocol describes the method by which information is passed between devices. It conforms to the Open Systems Interconnection model, which is defined in terms of layers. Each layer in a device apparently communicates with the same layer in another device. Actual communication is between adjacent layers in each device and the devices are only connected by the physical medium via the physical layer of the model. The CAN architecture defines the lowest two layers of the model: the data link and physical layers. The application levels are linked to the physical medium by the layers of various emerging protocols, dedicated to particular industry areas plus any number of propriety schemes defined by individual CAN users.

The physical medium consists of a twisted-pair with appropriate termination. In the basic CAN specification, it has a transmission rate of up to 250 KBaud whilst full CAN runs at 1 MBaud.

The physical and data link layers will normally be transparent to the system designer and are included in any component that implements the CAN protocols. There are some microcontrollers with integral CAN interfaces, for example, the 8051-compatible Siemens C505C processor and the 16-bit SAB-C167CR. The 81C91 is a standalone CAN controller which directly interfaces to many microcontrollers.

The connection to the physical medium can be implemented with discrete components or with the 82C250 integrated circuit or similar.

II. Functional Description

The represented CAN controller (**MBCAN**) contains all the necessary hardware for controlling the serial communication flow through the area network using the CAN-protocol. The MBCAN conforms to the CAN specification 2.0 part B. The MBCAN fully supports the Standard Format frames and it acts as a passive node for the frames in Extended Format. That means the MBCAN can be connected in a network that makes use of Extended Format as well as Standard Format. Messages in Extended Format are only received - the MBCAN doesn't corrupt them but sends acknowledge if they are correct. Messages with extended identifier cannot be accepted (i.e. the host CPU is not notified) nor can the MBCAN send a frame in Extended Format.

The MBCAN must be linked to the physical transfer medium through an appropriate bus transceiver according to ISO/DIS 11898 or modified RS-485 transceivers. The MBCAN connects to this transceiver using one logic-input and one logic-output line. These are RX and TX correspondingly.

The MBCAN can be partitioned in two blocks – the *Protocol Core* accomplishing all the CAN rules, and the *Interface* block that provides the ability to control the *Protocol Core* using a general microprocessor bus (see Fig.1).

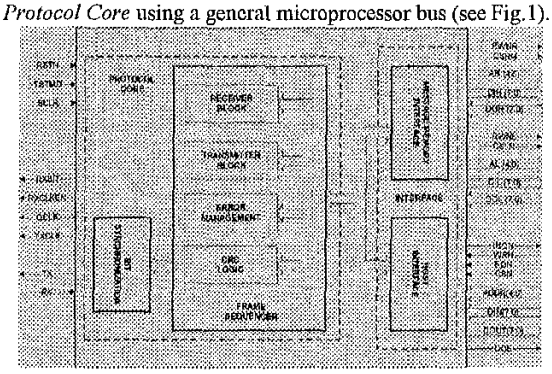

Fig.1 Block Diagram of CAN Controller

The *Frame Sequencer* is responsible for realizing the frame serialization, coding, decoding, error detection and signaling, and bus monitoring. The *Transmitter* and the *Receiver* incorporate the corresponding state machines needed for the transmission and for the reception of correct frames.

The *Error Management* unit performs the error confinement according to the CAN-protocol. It comprises of two error counters and additional logic, in order to define the bus/error status of the MBCAN.

The *CRC Logic* takes care of CRC generation during transmission of a message, and of a CRC check at the end of a received message.

The *Bit Synchronization* block controls the bit duration and bus sampling point, and derives information for synchronizing with the serial data stream. The internal logic of this block is able to lengthen/shorten the bit timing so that a correct sampling of the CAN-bus may be performed.

Blagomir Donchev - PhD Student, Technical University-Sofia, Kliment Ohridski Str. 8, 1977, Sofia, Bulgaria (telephone: +35929653115, e-mail: donchev@ecad.vmei.acad.bg).

Marin Hristov - PhD, Professor, Technical University-Sofia, Kliment Ohridski Str. 8, 1977, Sofia, Bulgaria (telephone: +35929653115, e-mail: mhristov@ecad.vmei.acad.bg).

The MBCAN operates with three message buffers. They are realized in a dedicated external Dual Port RAM, called message memory. The *TX Message Buffer* takes 10 bytes of that memory into which the host writes the message that is to be transmitted. There are two *RX Message Buffers. Each* of them occupies 10-bytes of the message memory for storing the incoming message. In this way the host is able to process one message while another is being received. When both of the RX buffers are full, overload frames are generated until the host clears at least one of them. The interface signals to the DPRAM are elaborated inside the *Message Memory Interface*.

The *Host Interface* accomplishes a general purpose, non-multiplexed, parallel interface through which any control device (implementing the application layer of the protocol) can access control and status information as well as read and write the message buffers.

III. Host Interface

There are two types of accesses that might be performed by the host:

- Read the contents of the internal registers.
- Write into the internal registers.

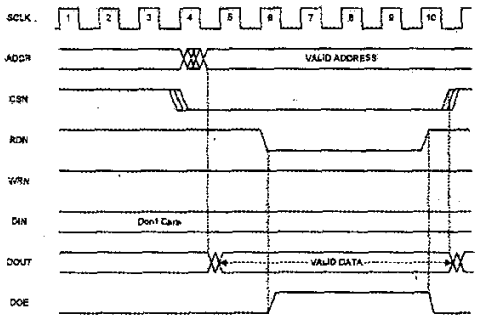

Fig.2 Read-Access

The host initiates *read access* by first issuing valid **ADDR** and **CSN**. Internal address decoding is done when **CSN** is low. The output bus **DOUT** becomes valid after the internal address decoding. When the host asserts **RDN** LOW the **DOUT** bus is already stable. The **DOE** signal is set to HIGH when both the **RDN** and **CSN** are LOW. The **DOE** should enable the three-state buffers if any at the system level. Therefore the CPU can sample the **DOUT** at any time while **RDN** is LOW.

The host initiates *write access* by first issuing valid **ADDR** and **CSN**. When these become stable, the host asserts **WRN** LOW. While the **WRN** signal is LOW, the MBCAN updates the accessed register at every rising edge of the **SCLK** (cycles 7,8,9 and 10). The host CPU must place valid (stable) data over the **DIN** bus prior to set the **WRN** HIGH. The internal register being target of the current access is updated more than once, but the last update is always performed at the end of a valid write operation (**SCLK** cycle 10). Thus the register effectively stores meaningful information. In the case of a write access to the *Command Register* the commands are executed once only, after the host has finished the operation (**WRN** is set to HIGH) and the valid data have already been latched.

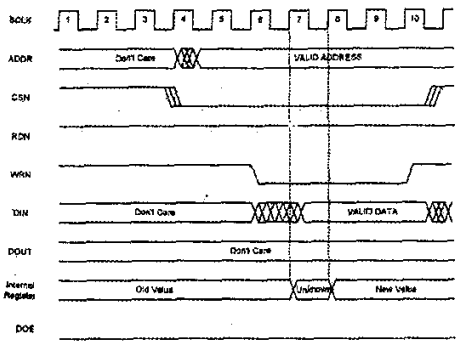

Fig.3 Write access

IV. Message Memory Interface

The data buffers for the received and the transmitted messages are not implemented into the MBCAN. The MBCAN is intended to use an external random access memory for the purpose of data storage. The MBCAN directly controls dual port asynchronous memory (DPRAM) with a capacity of 32 bytes. That means the address, data and access-control signals are issued directly from the MBCAN and the host CPU is not occupied in any data transfer to or from this memory. Consequently this memory is invisible to the host CPU – only the registers described in the register map of the MBCAN are mapped into the address space of the host CPU.

Dual port memory is needed in order to access the data both from the side of the CAN protocol core and from the side of the host CPU.

V. CAN Bit Timing

A bit period is built up from a number of time quanta as defined by the CAN protocol. The time quantum is a fixed unit of time derived from the system clock (oscillator) by the means of a prescaler. The nominal bit period is the result of the addition of the programmable segments TS1 and TS2 and the fixed length duration segment SYNCSEG (shown at Fig.5). TS1 and TS2 determine the number of clock cycles per bit period and the location of the sample point (Eq.1 and Eq.2).

$$T_{TS1} = T_{QCLK}(8TS1.3 + 4TS1.2 + 2TS1.1 + TS1.0 + 1) \quad (1)$$

$$T_{TS2} = T_{QCLK}(4TS2.2 + 2TS2.1 + TS2.0 + 1) \quad (2)$$

Therefore the contents of the **BT0** and **BT1** registers define the nominal bit-rate of the transmitted and received messages.

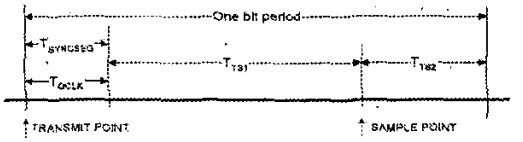

Fig.5 Bit Period as implemented in the MBCAN

A typical case is 1Mb/s with 8 quanta per bit with the sample point at the end of the 5th quantum and maximum of 2 quanta per resynchronization jump. Thus the SCLK should be 16 MHz, the contents of the BT0 register must equal hex (80) and BT1 register must be hex (23).

VI. CAN Communication Protocol

The MBCAN bus controller supports the four different CAN-protocol frame types for communication: Data Frame, to transfer data; Remote Frame, to request data; Error Frame, globally signal a (locally) detected error condition; Overload Frame, to extend delay time of subsequent frames.

There are two logical bit representations used in the CAN-protocol: A recessive bit on the bus-line appears only if all connected MBCAN send a recessive bit at that moment. Dominant bits always overwrite recessive bits i.e. the resulting bit level on the bus-line is dominant.

- Data Frame

A Data Frame carries data from a transmitting MBCAN to one or more receiving MBCANs. A Data Frame is composed of seven different bit-fields: Start-Of-Frame, Arbitration Field, Control Field, Data Field, CRC Field, Acknowledge Field, End-Of-Frame, Start-Of-Frame Bit.

Signals the start of a Data Frame. It consists of a single dominant bit used for hard synchronization of a MBCAN in receive mode.

Arbitration Field - There are two formats for this message field – extended and standard format. The MBCAN is capable of transmitting, receiving and accepting messages with standard identifier. The MBCAN can only receive a frame with extended identifier, i.e. it will reject the message after sending acknowledge. Thus the MBCAN may be connected to a network using both types of identifiers.

Following is the description of the standard arbitration field.

Standard Arbitration Field consists of the message identifier and the RTR bit.

In the event of simultaneous message transmissions by two or more MBCANs the bus access conflict is solved by bit-wise arbitration, which is active during the transmission of the Arbitration Field.

Identifier – This 11-bit field is used to provide information about the message, as well as the bus access priority. It is transmitted in the order ID.10 to ID.0 (LSB). The situation that the seven most significant bits are all recessive must not occur. An Identifier does not define which particular MBCAN will receive the frame, because a CAN based communication network does not discriminate between a point-to-point, multicast or broadcast communication.

'Remote Transmission Request Bit (RTR) – An MBCAN, acting as a receiver for certain information may initiate the transmission of the respective data by transmitting a Remote Frame to the network, addressing the data source via the Identifier and setting the RTR bit HIGH (remote; recessive bus level). If the data source simultaneously transmits a Data Frame containing the requested data, it uses the same Identifier. No bus access conflict occurs due to the RTR bit being set LOW (data; dominant bus level) in the Data Frame.

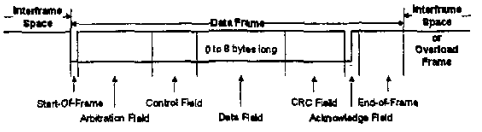

Fig.6 Data Frame Format

Control Field - This field consists of six bits. It includes two reserved bits, transmitted with a dominant bus level, and is followed by the Data Length Code (4 bits). The number of bytes in the Data Field is indicated by the Data Length Code. Admissible values of the Data Length Code and hence the number of bytes in the Data Field, are 0 to 8. A logic 0 (logic 1) in the Data Length Code is transmitted as a dominant (recessive) bus level, respectively.

Data Field - The data, stored within the Data Field of the Transmit Buffer, is transmitted according to the Data Length Code. Conversely, data of a received Data Frame will be stored in the Data Field of a Receive Buffer. Data is stored byte-wise both for transmission and reception by the microcontroller and on the first data byte (lowest address) is transmitted/received the first byte.

Cyclic Redundancy Code Field (CRC) - The CRC field consists of the CRC Sequence (15 bits) and the CRC Delimiter (1 recessive bit). The CRC encloses the destuffed bit stream of the Start-Of-Frame, Arbitration Field, Control Field, Data Field and CRC Sequence. The most significant bit of the CRC Sequence is transmitted/received first. This frame check sequence, implemented in the MBCAN, is derived from a cyclic redundancy code best suited for frames with a total bit count of less than 127 bits.

Acknowledge Field (ACK) - The Acknowledge Field consists of two bits, the ACK Slot and the ACK Delimiter, which are transmitted with a recessive level by the transmitter of the Data Frame. All MBCAN having received the matching CRC Sequence, report this by overwriting the transmitter's recessive bit in the ACK Slot with a dominant bit. Thereby a transmitter, still monitoring the bus level recognizes that at least one receiver within the network has received a complete and correct message. The ACK Delimiter (recessive bit) is the second bit of the ACK Field. As a result, two recessive bits surround the ACK Slot.

End-Of-Frame - Each Data Frame or Remote Frame is delimited by the End-Of-Frame bit sequence, which consists of seven recessive bits.

- Remote Frame

An MBCAN, acting as a receiver for certain information

may initiate the transmission of the respective data by transmitting a Remote Frame to the network, addressing the data source via the Identifier and setting the RTR bit HIGH (remote; recessive bus level). The Remote Frame is similar to the Data Frame with the following exceptions: RTR bit is set HIGH, Data Length Code is ignored, No Data Field Contained.

- Error Frame

The Error Frame consists of two different fields. The first field is accomplished by the superimposing of Error Flags contributed from different MBCANs. The second field is the Error Delimiter. There are two forms of an Error Flag: Active Error Flag, Passive Error Flag.

An error-active MBCAN detecting an error condition signals this by transmission of an Active Error Flag. This Error Flag's form violates the bit-stuffing law applied to all fields, fixed form of the fields Acknowledge Field or End-Of-Frame. Consequently, all other MBCANs detect an error condition and start transmission of an Error Flag. Therefore the sequence of dominant bits, which can be monitored on the bus results from a superposition of different Error Flags transmitted by individual MBCANs. The total length of this sequence varies between six and twelve bits.

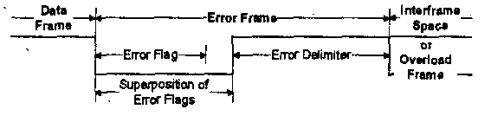

Fig.7 Error Frame Format

An error-passive MBCAN detecting an error condition tries to signal this by transmission of Passive Error Flag. The error-passive MBCAN waits for six consecutive bits with identical polarity, beginning at the start of the Passive Error Flag. The Passive Error Flag is complete when these six identical bits have been detected.

The Error Delimiter consists of eight recessive bits and has the same format as the Overload Delimiter. After Transmission of an Error Flag, each MBCAN monitors the bus-line until it detects a transmission from a dominant-to-recessive bit level. At this point in time, every MBCAN has finished sending its Error Flag and all MBCANs start transmission of seven recessive bits. If a detected error is signaled during transmission of a Data Frame or Remote Frame, the current message is spoiled and a retransmission of the message is initiated.

- Overload Frame

The Overload Frame consists of two fields, the Overload Flag and the Overload Delimiter. There are two conditions in the CAN-protocol, which lead to the transmission of an Overload Flag: Receiver circuitry requires more time to process the current data before receiving next frame; Detection of a dominant bit during Intermission Field.

The transmission of an Overload Frame may only start: During the first bit period of an expected Intermission Field; One bit period after detecting the dominant bit during Intermission Field.

The Overload Flag consists of six dominant bits and has a similar format to the Error Flag.

The Overload Delimiter consists of eight recessive bits and takes the same form as the Error Delimiter. After transmission of an Overload Flag, each MBCAN monitors the bus-line until it detects a transition from a dominant-to-recessive bit level. At this point in time, every MBCAN has finished sending its Overload Flag and all MBCANs start simultaneously transmitting seven more recessive bits.

VII. Conclusion

CAN is ideal for any situation where microcontrollers need to communicate either with each other or with remote peripherals. In its home environment, the car, CAN was originally used to allow mission-critical real time control systems such as engine management systems and gearbox controls to exchange information. Here, CAN's short and guaranteed message latency times allowed each end of the network is working with current data, even where this may be changing on a hundreds of microsecond time-scale. These systems all utilize full CAN in that the CAN controllers filter out unwanted messages to reduce the host CPU load. However, the appearance of low-cost standalone full CAN devices has allowed less time-critical tasks such as door systems to become part of the CAN network. Indeed, the entire conventional wiring harness has been replaced in some instances by two-wire CAN networks in which even mundane devices such brake lights and indicators are just additional nodes.

The MBCAN is a soft macro written in VHDL and tested on Altera® PLD with the MSOS™ emulator.

TABLE1 AREA ESTIMATION

Equivalent gate number estimation	Gates
CAN Protocol Core	2650
MBCAN	3100

The area of the external Dual Port RAM necessary for the storage of messages is not included.

References

[1] CAN Specification, *Robert BOSCH GmbH, 1991.*
[2] CAN Description, *Siemens, 1997.*
[3] C. Holmes and M. Willoughby, "VHDL Language Course", *Rutherford Appleton Laboratory, 1997.*
[4] S. Mazor and P. Langstraat, "A Guide to VHDL", Synopsys, Inc., 1992.
[5] D. Smith, "HDL Chip Design", Doone Publication, Madison , 1996