Design of Controller Area Network for Sensor Network Application using Verilog-HDL

Rugved D. Katyarmal
PG Student, Electronics Engineering Department,
G. H. Raisoni College Of Engineering, Nagpur

Prof. P. Daigavane
Professor, Electrical Engineering Department,
G. H. Raisoni College Of Engineering, Nagpur

***Abstract*— Communication modules are required for sensors interface with the sensor network. CAN provide a high decreasing in wiring complexity and, additionally, make it easy to connect with several devices using a single pair of wires, allowing the data exchange between them at the same time. This paper first studies the Controller Area Network (CAN) protocol. The CAN controller has been designed in Verilog. The design is implemented on FPGA. The designed CAN controller is interfaced with a LM35 temperature sensor.**

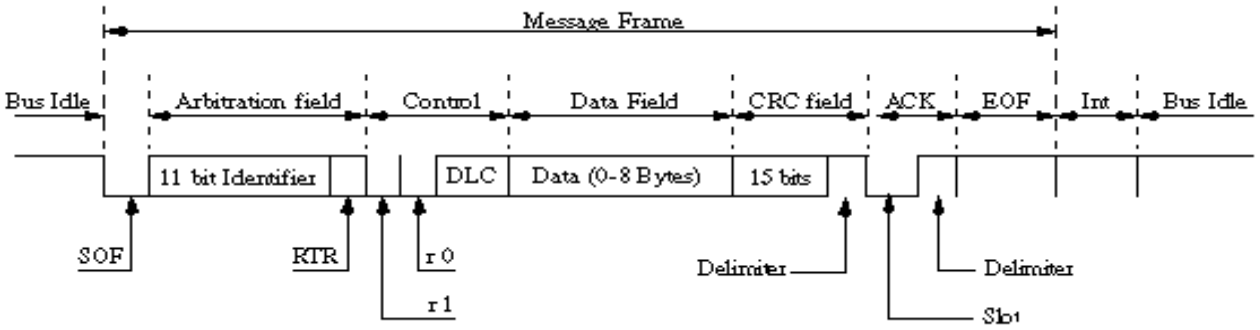
I. INTRODUCTION

Controller Area Network (CAN) is a broadcast and differential serial bus standard which was originally developed for automotive applications[1]. This protocol efficiently supports distributed real-time control with a very high level of security. Currently, CAN is also used in other applications, such as: home systems, medical devices, industrial control, etc. The interest in CAN is increasing rapidly due to the different applications that are foreseen and the availability of devices integrating CAN in the market. Moreover, a higher demand could be addressed with the new emerging technologies related to totally networked environments in factories, at home, etc. where control network protocols will be required, rather than data networks.

CAN has the following properties: prioritization of messages; guarantee of latency times; configuration flexibility; multicast reception with time synchronization; system wide data consistency; multi-master; error detection and signaling; automatic retransmission of corrupted messages as soon as the bus is idle again; distinction between temporary errors and permanent failures of nodes and autonomous switching off of defective nodes [2]. Such characteristics are very attractive to make CAN a suitable communication protocol for sensors networks wired and wireless.

Microcontrollers with integrated CAN interface suffer a performance penalty, as the microcontroller is responsible for message transmission and reception, besides reading inputs and driving outputs. This is a critical factor in industrial networks, where latency is an issue. If a standalone CAN controller is used, there is a cost penalty as an extra IC is required, increasing the cost of the system. Finally, IP cores developed by FPGA manufacturers and independent designers are generally not free. All these factors evidence the necessity of development of a CAN module for smart sensors. Using the ISO/OSI reference model, the CAN protocol is subdivided into different layers: the Data Link Layer (DLL) and the Physical Layer. The DLL is subdivided into two sub layers: the Logical Link Control (LLC) sub layer and the Medium Access Control (MAC) sub layer [2].

The paper is divided into five sections. This introduction is the first, next Controller Area Network (CAN) is described along with its message frame format. In the third section design methodology for designing the Controller Area Network using Verilog-HDL is described. The forth section goes through the simulation results followed by the final section conclusion and references.



SOF : Start Of Frame
r0, rl : two dominant bits
CRC : Cyclic Redundancy Code (15 bits)
EOF : End Of Frame (7 recessive bits)

RTR : Remote Transmission Request (1 bit)
DLC : Data Length Code (4 bits)
ACK : Acknowledge (2 bits)
INT : Intermission period (3 bits)

Fig. 1. CAN message format

THE CAN CONTROLLER

CAN networks have several features that make them well suited for control applications. Among these features, the main ones are the following:

- Serial bus communication for real time applications
- Multi-master node hierarchy, in a way that if a node fails the whole system does not collapse
- Typical network size could be between 32 to 64 nodes, running up to 100.
- Cost-efficient both in design and implementation.
- NRZ (Non-Return to Zero) coding with bit stuffing.
- To determine the priority of messages a CSMAICD mechanism is used (Carrier Sense telegram sections (start of frame, arbitration field, Multiple Access with Collision Detect)
- Message name (identifier) designates the information, but not the address of the node.

Fig.1 shows the format of data message frames for the CAN protocol. The message frames can have different lengths, depending on the type of identifier used (1 I-bit for CAN2.OA or 29-bit for CANZ.OB), and on the length of the transmitted received data, specified in the DLC field [2]. The number of data bytes in a transmission may vary from 0 to 8 bytes. Another important advantage of CAN is the large number of available components in the market, normally integrated as a peripheral port of a standard microcontroller. However, there is also a need of having such a peripheral block in an ASIC (Application Specific Integrated Circuit) or a FPGA (Field Programmable Gate Array)

II.A BIT TIMING LOGIC (BTL)

The bit timing logic monitors the serial CAN-bus line. It is synchronized to the bit stream on the CAN-bus on a 'recessive-to-dominant' bus line transition at the beginning of a message (hard synchronization) and re-synchronized on further transitions during the reception of a message (soft synchronization). It also provides programmable time segments and phase shifts (e.g. due to oscillator drifts) and to define the sample point and the number of samples to be taken within a bit time[3].

- **Synchronization Segment (Sync_Seg):** This part of the bit time is used to synchronize the various nodes on the bus.
- **Propagation Time Segment (Prop_Seg):** It is used to compensate the physical delay times within the network.
- **Phase Buffer Segment1 (Phase_Seg1) and Phase buffer segment2 (Phase_Seg2):** These segments are used to compensate for edge phase errors.

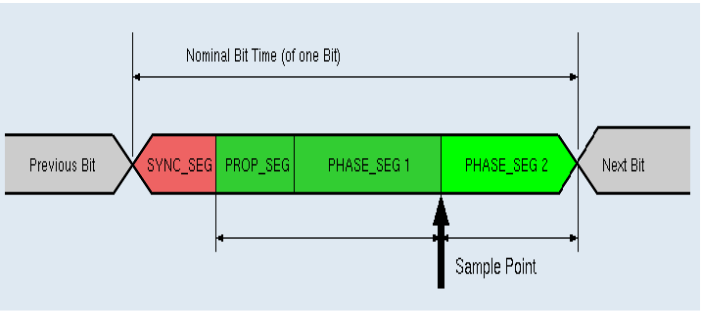


Fig. 2. CAN bit timing

Ii.B Transmit And Receive Buffer

The transmit buffer is an interface between the CPU and the Bit Stream Processor (BSP) that is able to store a complete message for transmission over the CAN network[3]. The buffer is ten bytes long, written to by the CPU and read out by the BSP.

The receive buffer is an interface between the acceptance filter and the CPU that stores the received and accepted messages from the CAN-bus line. The Receive Buffer represents a CPU-accessible ten-byte window. There are two ten bytes receive buffer. With the help of this the CPU is able to process one message while other messages are being received[3].

II.C Cyclic Redundancy Code Generator

Each message is provided with a 15-bit-long CRC code. This code is generated from the various fields from the frame format. When receiving a message frame, the code is generated analogously from the received data and compared with the CRC field in the message. This implies an error protection for the messages through the network[2].

Ii.D Acceptance Filter

The acceptance filter compares the received identifier with the acceptance filter register contents and decides whether this message should be accepted or not. In the event of a positive acceptance test, the complete message is stored in the receive buffer.

II. DESIGN METHODOLOGY

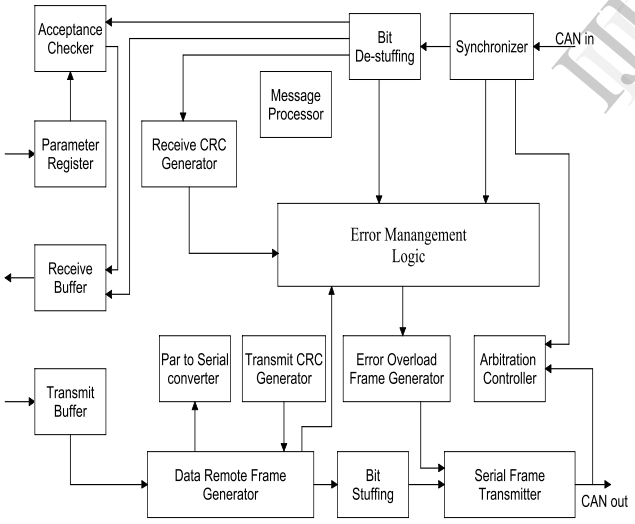


Fig. 3. Block diagram of CAN controller

• Parameter Registers: The code parameter, mask parameter and the Re-Synchronous Jump Width (SJW) specified for the CAN node are stored in this register.

• Transmit Buffers: There are ten transmit buffers, each of which can hold one byte of data. The controller then receives the data to be transmitted from the host CPU and stores the message in the buffer before further processing takes place.

• Data / Remote Frame Generator: Data / Remote Frame Generator is responsible for generating the message frame as specified by the CAN protocol.

• Par-Ser Converter: This unit serializes the message to facilitate the CRC computation.

• Transmit CRC Generator: Before transmission, this unit computes the CRC for the message to be transmitted. The generated CRC frame is appended to the message being transmitted before bit-stuffing is performed.

• Bit Stuff Unit: This unit performs bit-stuffing as specified by the CAN protocol, making the message suitable for transmission across the CAN network.

• Overload / Error Frame Generator: Generates Error or Overload frame whenever error or overload condition occurs. Error containment measures are also taken care of to ensure the accuracy of the controller's performance and its further participation in the CAN network.

• Serialized Frame Transmitter: This unit transmits the data/ remote frame or the error / overload frame or a dominant bit during the acknowledgment slot based on the prevalent conditions.

• Message Processor: This is the central unit which provides all the control and the status signals to the various other blocks in the controller. This unit routes the different signals generated in various blocks to the necessary target blocks.

• Error Management Logic: The error management logic consists of form checker, crc checker, acknowledgement checker etc. A form error is generated if any of the fixed-form fields in a received CAN message is violated. The fixed form fields include the CRC delimiter, ACK delimiter and the EOF field. During the transmission of the acknowledgement slot a transmitter transmits a recessive bit and expects to receive a dominant bit. If the node receives a recessive bit in the acknowledgement slot an ACK error is signaled.

• Bit De-stuffing unit: This unit performs the de-stuffing of the messages received from the CAN network. This unit also extracts the relevant information from the received message.

The CAN bus bit stream is sampled by the Synchronizer of the CAN controller. This sampled bit stream is then de-stuffed before the relevant information is extracted from the received message. Due to the bit stuffing process of the CAN protocol a stuff bit of opposite polarity follows a sequence of 5 consecutive bits of the same polarity. The function of the de-stuffing unit is to remove the stuffed bits from the received message.

III. SIMULATION RESULTS

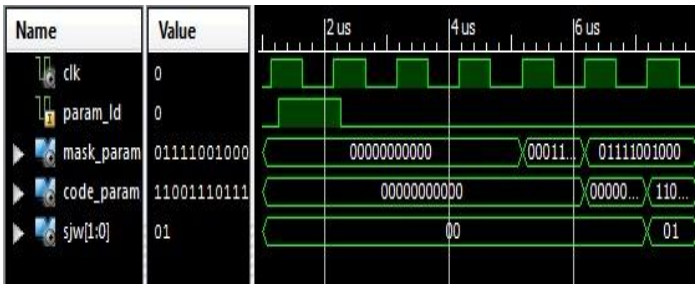


Fig. 4. Load Parameters

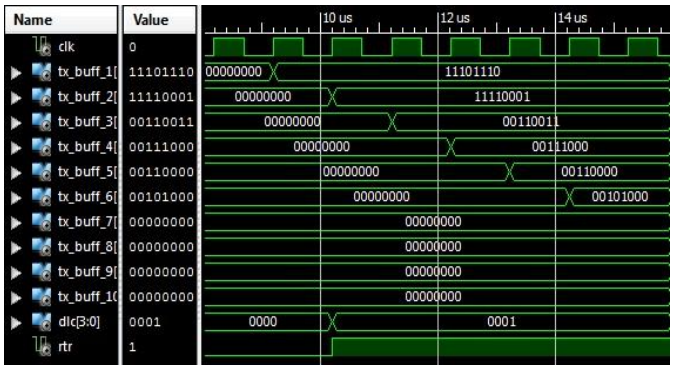


Fig. 5. Load Message

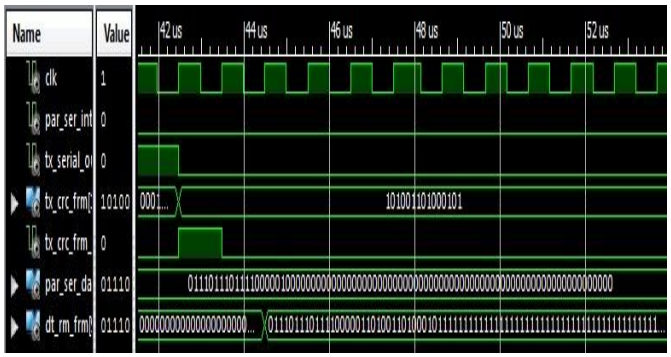


Fig. 6. Data Remote Frame Generation

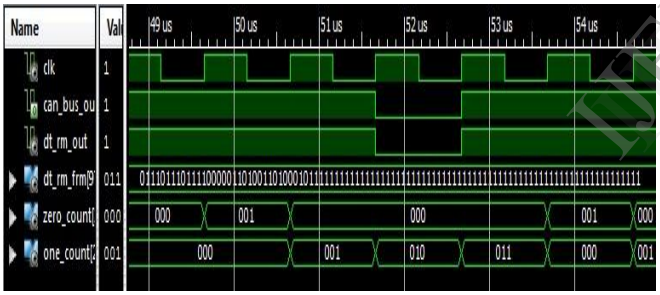


Fig. 7. Bit Stuffing Mechanism

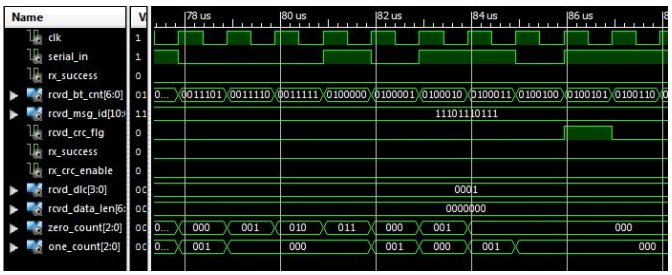


Fig. 8. Bit De-stuffing

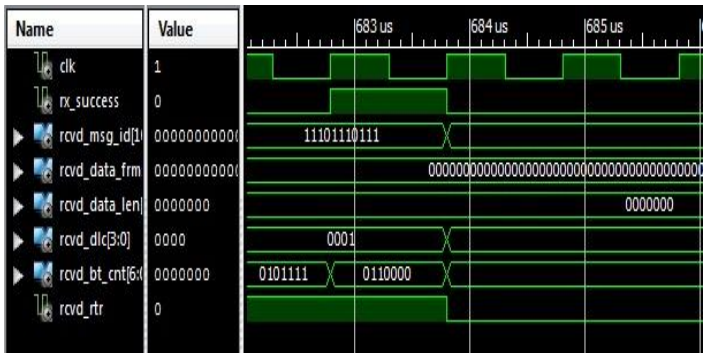


Fig. 9. Successful Reception

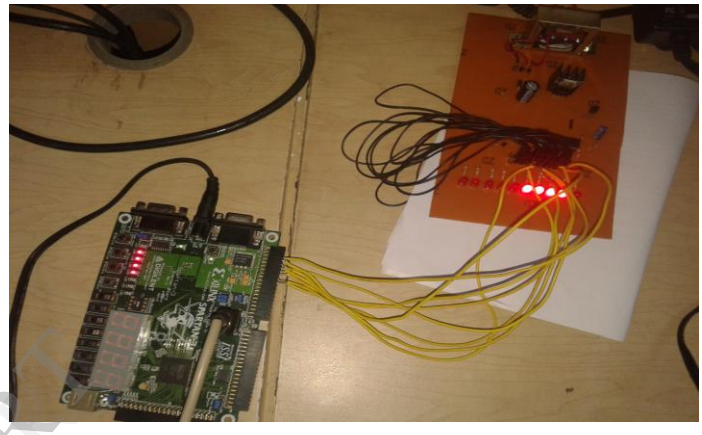


Fig. 10. Implementation and Interfacing of Controller Area Network with Temperature Sensor on FPGA

IV. CONCLUSION

The design of a CAN controller has been reported in this paper. The various simulation results of CAN controller are been reported. The designed CAN controller has been implemented on an FPGA and interfaced with a LM35 temperature sensor and the results are shown.

VI. REFERENCE

1. "CAN in Automation e. V. CiA 301 V4.2.0 - CANopen application layer and communication profile, Feb. 2011.
2. "Design of a CAN interface for custom circuits" J. de Lucas, M. Quintana, T. Riesgo, Y. Torroja, J. Uceda. - IEEE Industrial Electronics Society
3. CAN Specification Version 2.0, Robert Bosch GmbH, Stuttgart, Germany.
4. "A VHDL CAN MODULE FOR SMART SENSORS", Jose E. 0. Reges, Edval J P. Santos, Laboratory for Devices and Nanostructures, Recife, PE, Brasil, 4th Southern Conference on Programmable Logic 2008
5. "The application of controller area network on vehicle" Wang Xing, Huiyan Chen, Huarong Ding – IVEC International Vehicle Electronics Conference.
6. "Research On CAN controller conformance test system" Feng Luo, Jie Chen ICCSIT 2009. 2nd IEEE International Conference on Computer Science and Information Technology.
7. "A modular Controller Area Network vision system using programmable interface controller" Mohd Sharil b. Salleh, Herdawatie bt.Abd. Kadir, and Mohd Helmy bin Abd Wahab - ICED 2008. International Conference on Electronic Design
8. "Introduction to the Controller Area Network (CAN)" S. Corrigan, Texas Instrument, Application Report, July 2008".