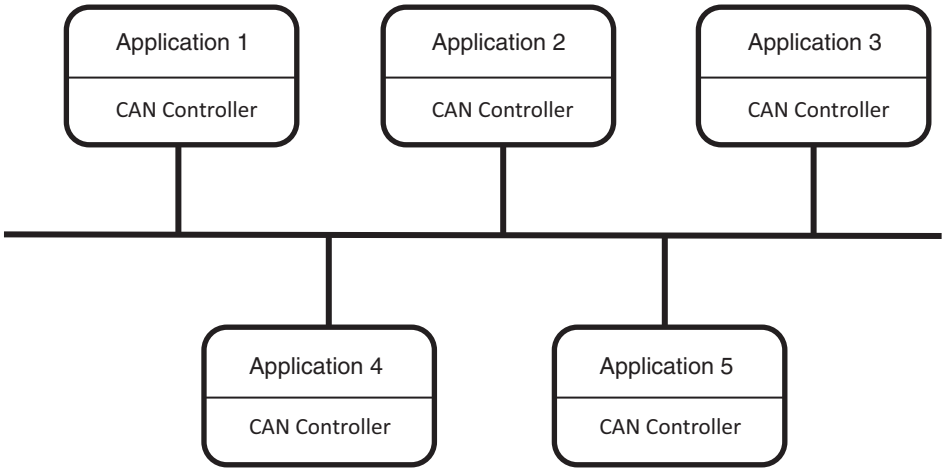
CAN Controller

April 2013 Reference Design RD1170

Introduction

The Controller Area Network (CAN) is a serial, asynchronous, multi-master communication protocol for connecting electronic Control modules, sensors and actuators in automotive and industrial applications. The CAN Controller is an interface between the Application and the CAN bus. The function of the CAN Controller is to convert the data provided by the application into a CAN message frame fit to be transmitted across the bus.

Figure 1. CAN Protocol



A CAN system sends message using a serial bus network. With every node connected to every other node in the network, no need of central controller for the entire network. Since CAN is a multi-master communication protocol, every node in the system is equal to every other node. Any processor can send a message to any other processor. Even if some processor fails, the other systems in the machine will continue to work properly and communicate with each other. Any node on the network that wants to transmit a message waits until the bus is free. Every message has an identifier, and every message is available to every other node in the network. The node selects those messages that are relevant and ignores the rest.

This document provides a brief description of CAN Controller and its implementation.

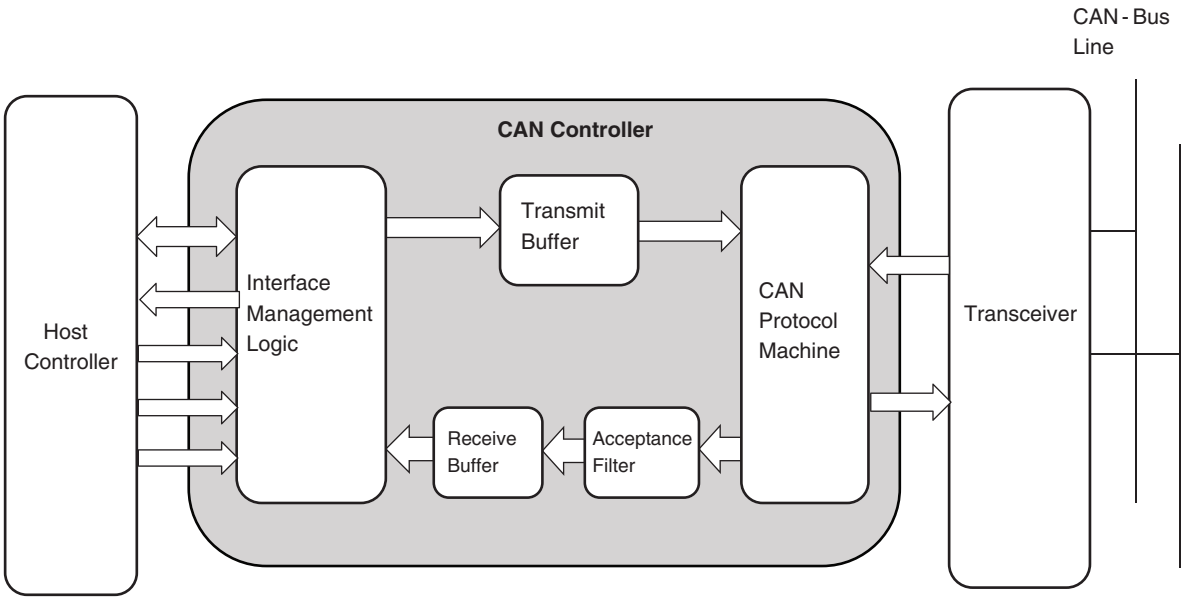
The design is implemented in VHDL. The Lattice iCEcube2™ Place and Route tool integrated with the Synopsys Synplify Pro® synthesis tool is used for the implementation of the design. The design can be targeted to other iCE40™ FPGA product family devices.

Features

- Supports Standard Data Frames and Remote Frames
- Uses two Receiver Buffers to store the messages
- Transmit buffers are used
- Acceptance Filter is used
- Data rates up to 1Mbit/s
- Error detection for data using 16-bit CRC

Functional Description

Figure 2. System Block Diagram



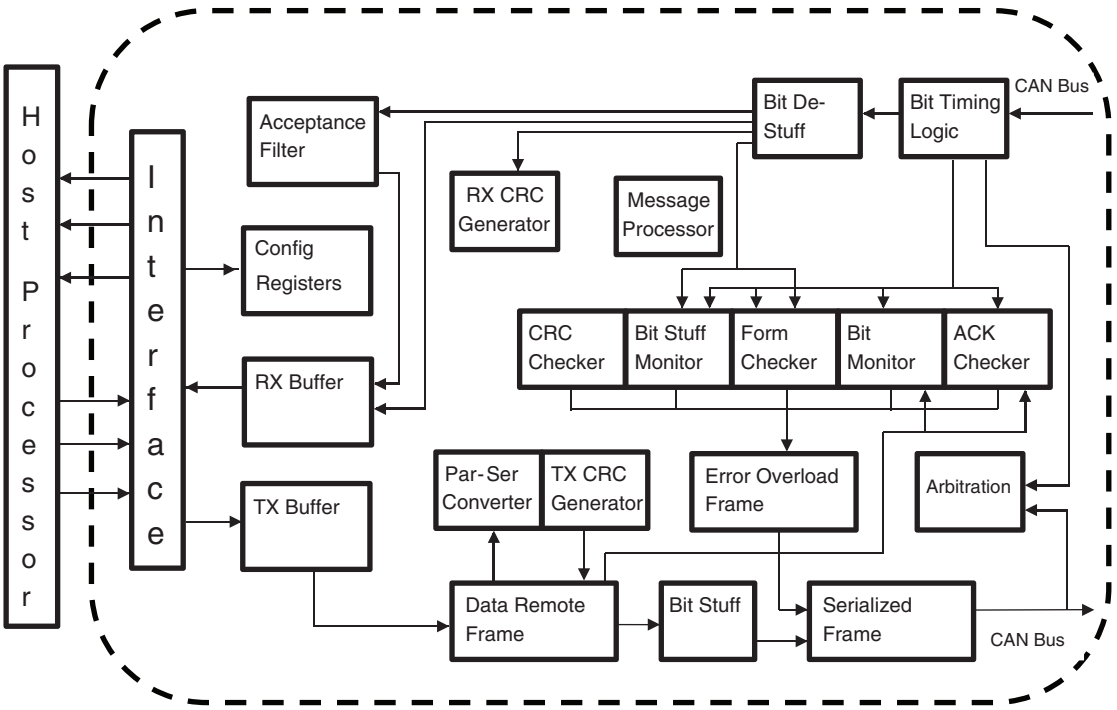
Signal Description

Table 1. Signal Description

Signal	Width	Type	Description
i_osc_clk	1	Input	System Clock
i_rst	1	Input	Asynchronous Active Low system Reset
i_param_ld_en	1	Input	Signal to load parameter to CAN Controller
i_data_in	8	Input	8-bit data to be loaded into Controller
i_tx_buff_ld_en	1	Input	Load Enable Signal to load data into transmitter buffer
i_rd_en	1	Input	Signal to read the RX buffer
i_init_err_st	1	Input	Signal to reset error counters
o_tx_buff_busy	1	Output	Status signal to indicate that buffer is loaded
o_buff_rdy	1	Output	Signal to indicate that the receiver buffer is loaded
o_data_out	8	Output	Data output bus

Design Module Description

Figure 3. Functional Diagram



Configurable Parameters

None

Design Details

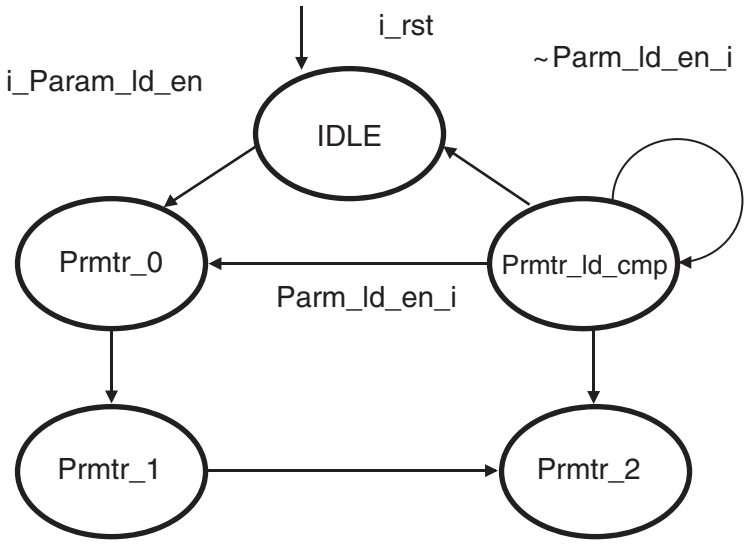
Configuration Registers

The controller receives the code parameter, the mask parameter and the Resynchronization Jump Width (SJW) specified for the CAN node from the Host controller and stores them in the configuration register. The content of Mask parameter, code parameter registers is used to determine the acceptance of message using acceptance filtering. The synchronous jump width register content is used for bit synchronization. The state diagram for the Configuration Registers is shown below.

Initially the mask parameter and the code parameter need to be loaded into the CAN Controller. The host controller asserts a signal called i_param_ld when it needs to load new values in the mask and code registers. The host controller accomplishes this by asserting the i_param_ld signal high. This indicates to the CAN controller that the host controller wishes to load the parameters into the corresponding registers. Following the high assertion of i_param_ld signal the Host proceeds to send the 8 bit data in the data bus in bytes. This signal is asserted only for one clock cycle.

Once the transmission is completed the registers retain the values stored in them. A new value is loaded only when the host controller initializes another parameter load by asserting the i_param_ld signal high. A reset to the system removes the parameters stored in theses registers.

Figure 4. Configuration State Register Diagram



Transmit Buffer

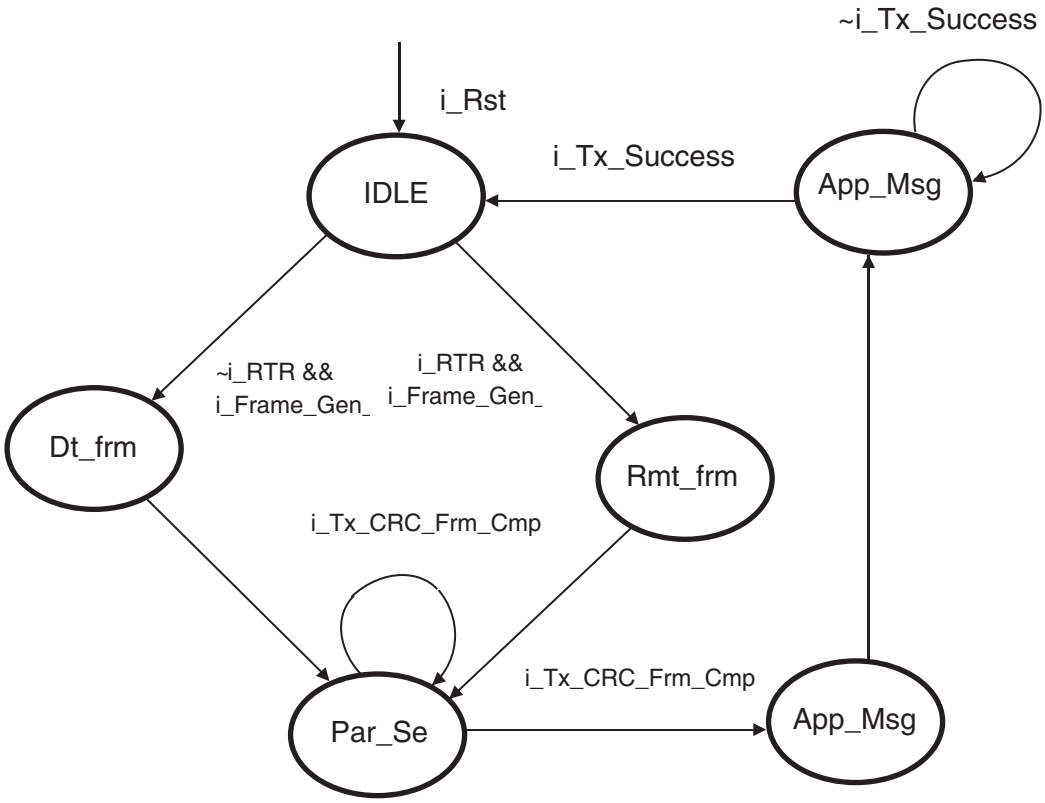
There are ten transmit buffers. Each buffer can hold one byte of data. The controller receives the message to be transmitted from the host controller and stores the message in the transmit buffer before further message processing takes place.

By asserting the i_tx_buff_ld signal high the host controller indicates to the CAN controller that it wishes to transmit a message over the CAN bus. Following high assertion of the i_tx_buff_ld signal, the host controller sends out the message on the 8 bit data bus in bytes. The controller loads the message on the data bus on to the transmit buffer at the positive edge of the clock.

The host controller sends the message in the order of the identifier first, followed by control bits, and then the data bytes. When controller completes the loading of the transmit buffer a busy signal goes high and stays high till the message has been transmitted successfully. After successful transmission the busy signal goes low and buffers are reset.

Data/Remote Frame Generation

Figure 5. Data/Remote Frame Generation State Diagram



This module is responsible for generating data and remote frame as specified by the CAN protocol. This data/remote frame generation is initialized after completion of loading of final transmit buffer. This is done by asserting frame generation initiative signal high. Based on the number of bits in the data field and status of Remote Transfer Request (RTR) bit a frame named o_par_ser_data is generated. If RTR bit is recessive, the message to be transmitted is a Remote. In this case the frame doesn't have any Data Field and will be formed by concatenation of the dominant start bit, the message Identifier and the Control Field. In other case if RTR bit is dominant then it is a Data Frame and contains dominant start bit, the Message Identifier, the Control Field and the Data Field. The Data Field is of variable length given by the DLC. The Data Field can contain zero to eight bytes of data.

The o_par_ser_data frame is serialized using parallel to serial converter module and fed as input to CRC generator module. After CRC frame generated from CRC generator module it will be appended to end of Data Field. The newly formed frame is named as o_dt_rm_frm. The State diagram for the Data/Remote frame generation is shown below.

Parallel to Serial Converter

This unit serializes the message to facilitate CRC computation. On computing data frame, a parallel to serial initializing signal is asserted high. It starts the parallel to serial conversion. Message is loaded into another temporary register, and at the same time CRC calculation is enabled by asserting a crc enable signal high. The content of the temporary register is left shifted out to the serial input of the CRC generation module. A count is incremented for every bit shifted out. As soon as the count reaches the frame length of data frame length a crc frame complete signal is asserted high and parallel to serial conversion is stopped.

CRC Generator

The CRC frame calculation commences with the high assertion of the CRC enable signal. The CRC frame is initialized to 15'h0 with the high assertion of the CRC initialization signal. The CRC frame for CAN is calculated by using a generator polynomial called CAN Polynomial and is represented in hexadecimal by 15'h4599. The CRC input message stream is divided by the generator polynomial given by

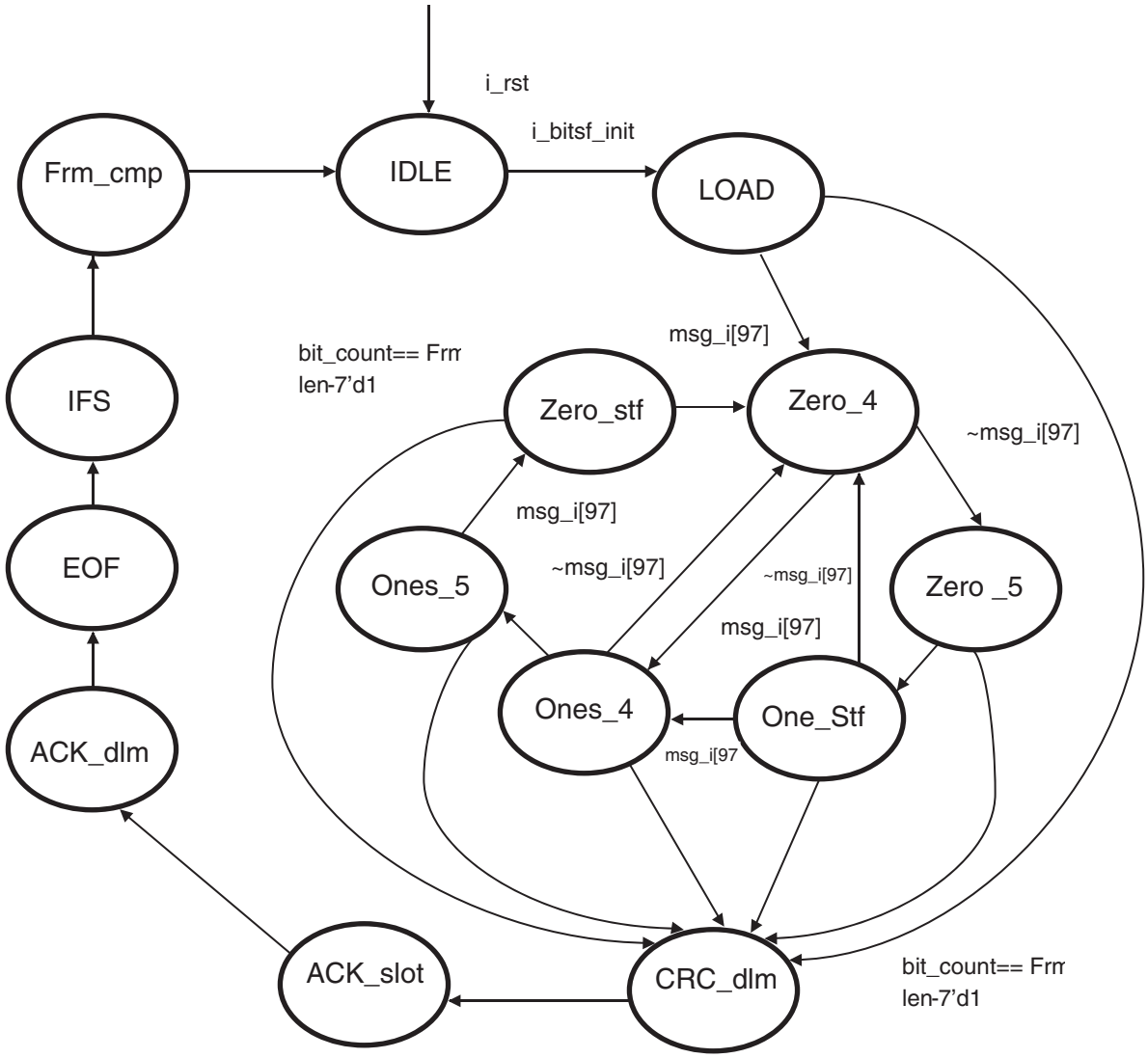
CAN Polynomial: x15 + x14 + x10 + x8 + x7 + x4 + x3 + 1

The remainder of this polynomial division gives the CRC sequence transmitted as part of the message frame.

Bit Stuff Unit

This unit performs the bit stuffing mechanism as specified by the CAN protocol, making the message suitable for transmission across the CAN network. As per the specifications, bit stuffing is done only on Data or Remote frames. Input to the stuffing unit are start bit, 11 bit message identifier, RTR bit, 6 bit control field, data field and 15 bit CRC field. The bit stuffing is initialized by the bit stuff initiate signal.

Figure 6. Bit Stuff State Diagram



On receiving bit stuff initializing signal, the message stream is stored in a temporary register named as msg_i. Then the MSB of that temporary register is checked for every clock cycle. If it is a 1, a counter variable for one is incremented or if it is 0, a counter variable for zero is incremented. If the sequence of one's is less than five then

state machines stays in state ones_4 and outputs logic 1. If it is equal to 5, then state machine enters the ones_5 state and outputs logic 1 and next state would be zero_stuff state. In the zero_stuff state outputs a logic 0, which is the opposite polarity of logic one. Similarly when a sequence of five zero's detected the state machine enters the one_stuff state and outputs a logic 1.

The CRC Delimiter, ACK slot, ACK delimiter and EOF bits are appended to the bit stuffed message to form the Data/Remote Frame; these bits are transmitted as recessive bits.

Overload/Error Frame Generation

On detecting an overload or error condition, an overload frame or an Error frame is transmitted. A message received when both the buffers are full cannot be stored in the receive buffers and will be lost. To avoid this situation an overload frame is generated by the receiver to indicate an overload condition to the other participating nodes. The transmission of the next message on the bus is delayed by the transmission of the Overload Frame.

Similarly when an error is detected the node sends out an Error Frame. The transmission of the Error Frame produces an error condition in the other participating nodes and causes the message to be retransmitted.

The Error Confinement process is also taken care of by the Overload /Error Frame Generation unit. It is designed as per the specifications in the Data Link Layer of the CAN protocol.

Serialized Frame Transmitter

This unit transmits the Data /Remote Frame or the Error /Overload Frame or a dominant bit during the acknowledgement slot based on the prevalent conditions. The Data / Remote transmission is initialized by asserting the data remote frame transmission signal high. The transmitting node continues to transmit the message until the last bit, provided there is no error condition encountered during the transmission. In the event of an error the node starts transmitting the Error / Overload Frame. The node doesn't transmit an Error Frame when the node is in Bus Off state.

Message Processor

The message processor is the central unit provides all the control and the status signal to the various other blocks in the controller. The success of a transmission or reception is indicated by this block. During arbitration message if a node loses arbitration it has to contend for bus access only after the completion of the current transmission. By asserting a re-transmission signal high the re-transmission of message which lost the arbitration can be done. The overload condition is also indicated by message processor.

Arbitration Control

The arbitration controller is responsible for indicating the arbitration status of the node. The i_arbtr_sts signal indicates the arbitration status of the node. If the i_arbtr_sts of the node is logic 1 then the node is a transmitter if it is logic 0 then the node has lost arbitration and functions as a receiver of the ongoing message.

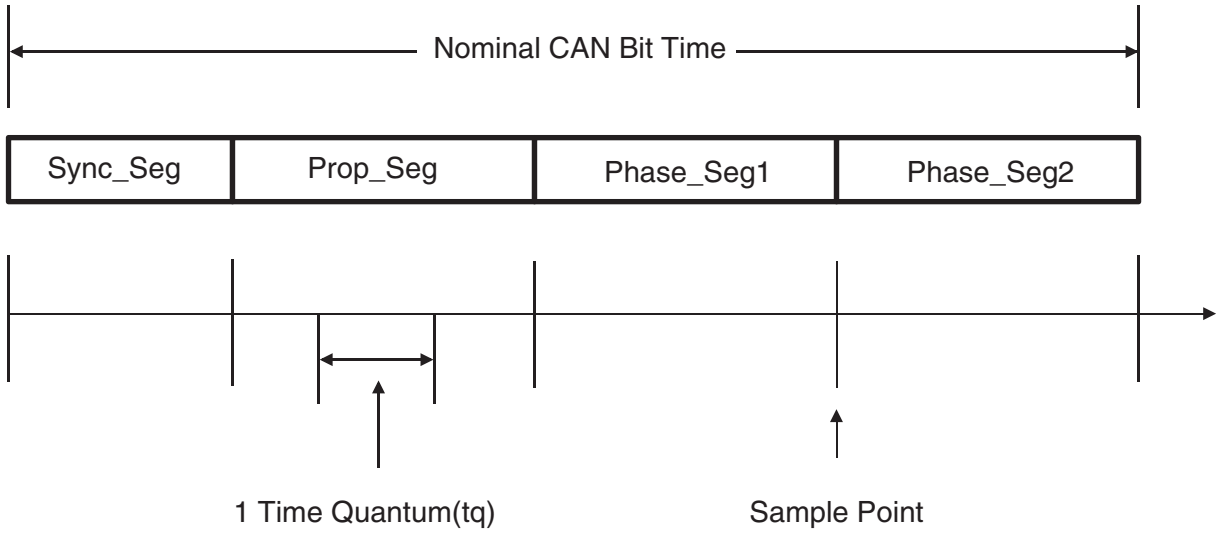
Bit Timing Logic(BTL) or Synchronizer

The Bit Timing Logic monitors the serial CAN – bus line and handles the bus line – related timing. 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). The BTL also provides programmable time segments to compensate for the propagation delay times and phase shifts and to define the sample point and the number of samples to be taken within a bit time.

According to CAN specification the bit time (i.e reciprocal of bit rate) is divided into four segments, The Synchronization Segment, the Propagation Time segment, the Phase Buffer Segment1, and the Phase Buffer Segment 2. Each segment consists of a specific, programmable number of time quanta. The length of time quantum (tq) , which is the basic time unit of the bit time, is defined by the CAN Controllers system clock fsys and the Baud rate pre - scaler (BRP).

The Synchronization segment Sync_Seg is that part of the bit time where edges of the CAN bus level are expected to occur; the distance between an edge that occurs outside of Sync_Seg and Sync_Seg is called the phase error of that edge. The Propagation time segment Prop_Seg is intended to compensate for the physical delay times within the CAN network. The phase buffer segments Phase_Seg1 and Phase_Seg2 surrounds the sample point. The Re – synchronization Jump width (SJW) defines how far a resynchronization may move the Sample point inside the limits defined by the Phase Buffer Segments to compensate for edge phase errors.

Figure 7. Bit Time



Bit De Stuff 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.

The de-stuffing process is initialized by the high assertion of bit de-stuff initialization signal. As soon as the de-stuffing process is initialized the CRC calculation of the received bit stream is enabled. A 64 bit temporary register stores the received and de-stuffed bits. The temporary register is shifted to the left by one bit position for every de-stuffed bit and incoming de-stuffed bit is moved into the 0th bit position of the temporary register.

The incoming bit is checked during every clock cycle. If it is a 1, a counter variable for one, one_count is incremented or if it is a 0, a counter variable for zero_count is incremented. These counters are incremented even if a stuff bit is encountered. After coming across a sequence of five ones or five zeros the next bit is removed from the bit stream and it is not stored in the temporary register.

CRC Checker

The CAN controller performs the CRC comparison between received CRC frame and generated CRC frame. If there is a mismatch in the comparison a CRC error will be flagged by asserting a crc error signal high.

Bit Stuff Error

This unit signals a stuff error when six consecutive bits of equal polarity are detected in between start of frame and CRC delimiter of the received message. The one count and zero count are fed as input to the Bit stuff error module.

A stuff error is flagged if the one count is equal to five and the serial input is equal to logic 1 or if the zero count is equal to five and the serial input is equal to logic 0.

Form Checker

This unit checks for the serial input at the fixed form fields which are the CRC Delimiter bit, Acknowledge Delimiter bit and the End of Frame space bits. If the receiver detects a dominant bit in any of these fields a Form error is signaled.

Bit Monitor

A CAN node acting as the transmitter of a message, samples back the bit from the CAN bus after putting out its own bit. A bit error occurs if a transmitter sends a dominant bit but detects a recessive bit on the bus line or, sends a recessive bit but detects a dominant bit on the bus line. When a dominant bit is detected instead of a recessive bit, no error occurs during the Arbitration field or the Acknowledge Slot because these fields may be overwritten by dominant bit in order to achieve arbitration and acknowledge functionality.

Acknowledgement Checker

During the transmission of the acknowledgement slot a transmitter transmits a recessive bit and expects to receive a dominant bit. If the transmitting node receives a recessive bit in the acknowledgement slot it is understood that none of the nodes in the network received the message correctly and an Ack error is signaled. If the node receives a dominant bit in the acknowledgement slot then it is understood that at least one other node, has received the frame correctly.

Acceptance Filter

This unit checks the incoming message ID and determines valid frame. The Acceptance filter, based on the Mask and Code parameter of the host application, filters the received message and stores only those required by the host application. Thus the filtering mechanism ensures only relevant messages are processed and the rest are ignored. All messages that are let through the filter must be read and checked by the Host processor.

The design of the Acceptance filter is defined by two parameters, Mask parameter and Code parameter. The acceptance Mask parameter specifies which particular bits to compare in the acceptance code parameter with the identifier of the message. The set bits in the Mask parameter indicate the bits that are relevant in the code parameter. The code parameter and the inbound message ID must be equal in those positions which are marked relevant by the Mask parameter bits. For standard identifiers, the Mask parameter and the code parameter must be both, in the range of 0 o 2047. The Mask parameter and the Code parameter are user defined. These values can be changed using the registers available for the same.

The filtering process compares the Message Identifier with the filters, composed of the acceptance Code and Mask registers. The filtering process evaluates the following binary expression to check the acceptance of the message

AND (XOR (code parameter, received message ID), mask parameter) = 11'b00000000000

If the equation evaluates to zero, the received message is accepted. Even if one of the 11 bits is logic 1, the received message is deemed to have failed the filtering mechanism and id rejected.

Receive Buffer(RXB)

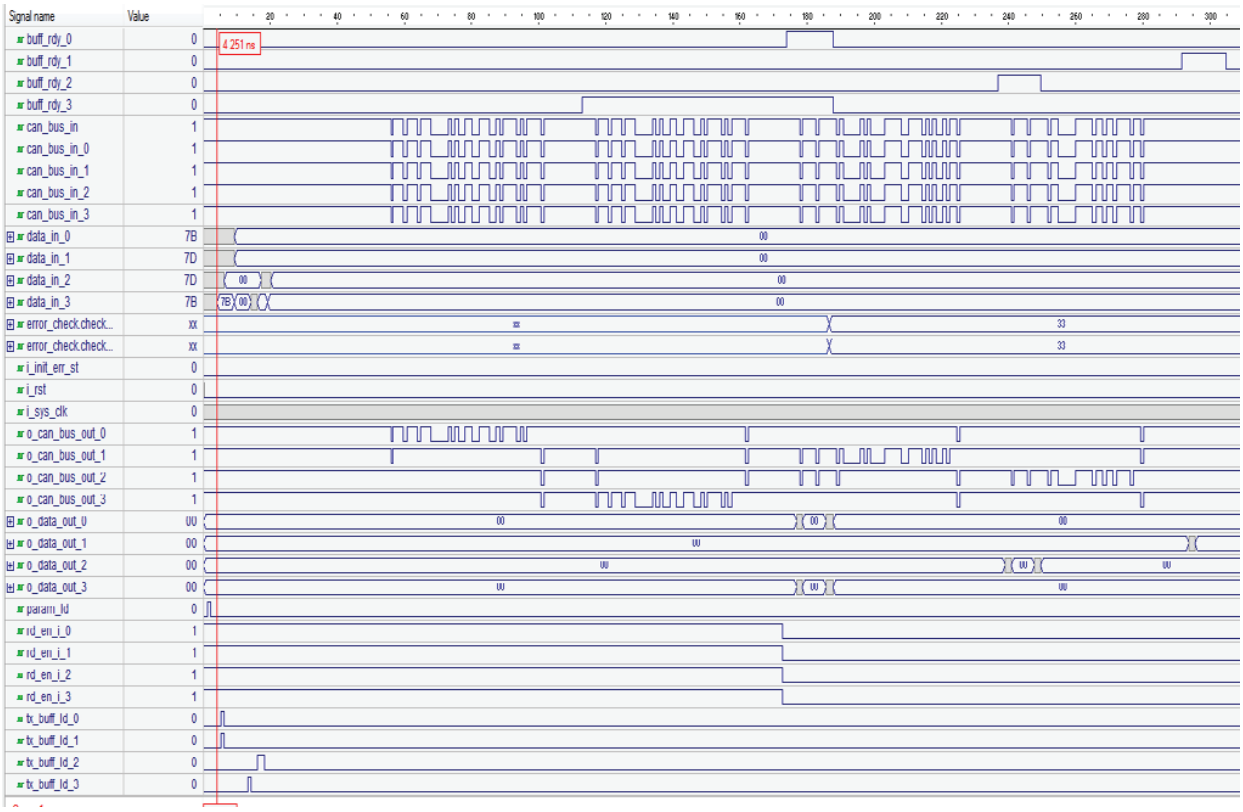
The Receiver buffer is an interface between the acceptance filter and the Host controller that stores the received and accepted messages from the CAN – bus line. There are two 10 byte buffers, o_rx_buff0 and o_rx_buff_1 that are used alternatively to store the message receive from the CAN bus. This enables the host CPU to process a message while another message is being received by the controller.

Interface Management Logic (IML)

The interface between the host application and the CAN controller consists of an 8-bit data bus to transfer the message to the controller's transmit buffer, an 8-bit data read bus which reads the message received from the controller's receive buffers and status signal to perform the requisite handshaking for these operations.

Simulation Waveforms

Figure 8. Simulation Waveforms



Implementation

This design is implemented in VHDL. When using this design in a different device, density, speed or grade, performance and utilization may vary.

Table 2. Performance and Resource Utilization

Family	Language	Utilization (LUTs)	f_{MAX} (MHz)	I/Os	Architectural Resources
iCE40[1]	Verilog	2335	>50	25	(352/440)PLBs

1. Performance and utilization characteristics are generated using iCE40LP4K-CM121 with iCEcube2 design software.

References

- iCE40 Family Handbook

Technical Support Assistance

Hotline: 1-800-LATTICE (North America)
+1-503-268-8001 (Outside North America)
e-mail: techsupport@latticesemi.com
Internet: www.latticesemi.com

Revision History

Date	Version	Change Summary
April 2013	01.0	Initial release.