CAN Node using HCS12

Ketan Kulkarni, Siddharth Dakshindas
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
e-mails: krkulkarni@oakland.edu, spdakshindas@oakland.edu

***Abstract*—The objective of this project is to demonstrate CAN capabilities of HCS12.A network of 2 CAN nodes will be created using 2 Dragon-12 boards. The MSCAN12 module will be set at 125 KHz baud rate. Both the boards will be communicating on different IDs.**

I. Introduction

The controller area network (CAN) has been a robust serial communication protocol since the past 3 decades. CAN vastly reduces complexity and decreases wiring considerably. CAN specifications were later published and most automotive micro-controllers now support CAN 2.0A and 2.0B. The HCS12 CAN Module provides the following features:

1. Full implementation of the CAN 2.0A/B protocol.
2. Five receive buffers with FIFO storage scheme.
3. Three transmit buffers with internal prioritization.
4. Programmable wake-up functionality with integrated low-pass filter.
5. Three low-power modes: sleep, power-down, and MSCAN12 enable.
6. Programmable loopback mode supporting self-test operation.
7. Clock source coming from either E-clock or oscillator clock.
8. Internal time for time stamping of received and transmitted message.
9. Global initialization of configuration registers.

The CAN protocol specifies the lowest two layers of the ISO seven-layer model: data link and physical layers.

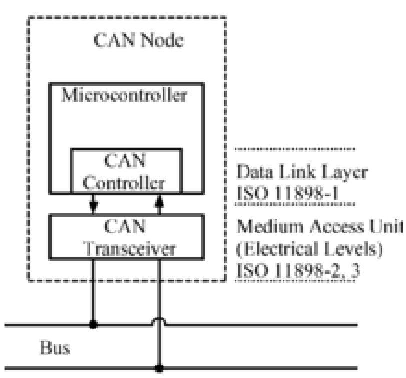


Figure 1 Layered Architecture of CAN

CAN is a message-based protocol with an identifier embedded in each message. The identifier helps the node to decide whether it is to receive and process the message. Also, in case of 2 messages on the bus at the same time, the identifier helps in arbitration. The lower the value of the message identifier, higher its priority.

Since an address is not used in the CAN system, there is no need to reconfigure the system whenever a node is added to or deleted from a system.

The CAN specifications use the terms "dominant" bits and "recessive" bits where dominant is a logical 0 (actively driven to a voltage by the transmitter) and recessive is a logical 1 (passively returned to a voltage by a resistor). The idle state is represented by the recessive level (Logical 1). If one node transmits a dominant bit and another node transmits a recessive bit then the dominant bit "wins" (a logical AND between the two). This is how, it is easy for message arbitration mechanism to decide the priority of the messages.

During arbitration, each transmitting node monitors the bus state and compares the received bit with the transmitted bit. If a dominant bit is received when a recessive bit is transmitted then the node stops transmitting (i.e., it lost arbitration). Arbitration is performed during the transmission of the identifier field. Each node starting to transmit at the same time sends an ID with dominant as binary 0, starting from the high bit. As soon as their ID is a larger number (lower priority) they will be sending 1 (recessive) and see 0 (dominant), so they back off. At the end of ID transmission, all nodes but one have backed off, and the highest priority message gets through unimpeded.

The CAN messages are transmitted and received in terms of frames. Bits in the frame are represent in fields and each field has a special function.

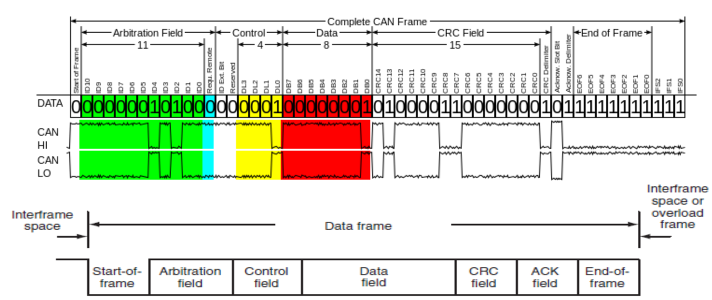


Figure 2 Standard CAN Frame

The HCS12's MSCAN12 module supports CAN 2.0A/B. It supports standard and extended data frames as well as remote frames with a programmable bit rate up to 1 Mbps2. The MSCAN12 has a triple transmit buffer scheme which allows multiple messages to be set up in advance and achieve an optimized performance. Received messages are stored in a five stage input FIFO. The module has a configurable hardware identifier filter which may be applied to incoming messages. A successful transmission or a message reception with a matching identifier will be flagged and can generate an interrupt request to the CPU.

MSCAN12 block diagram:

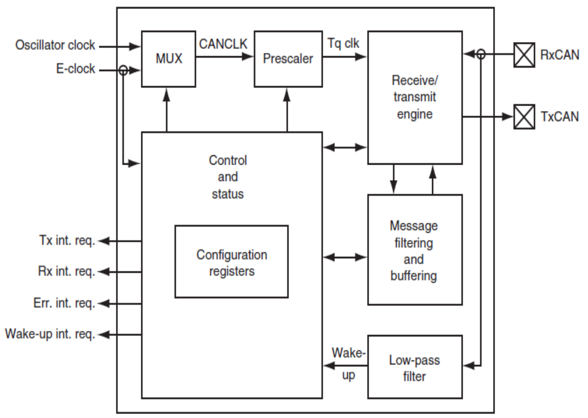
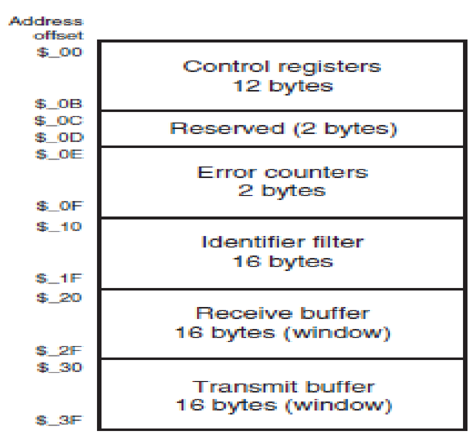


Figure 3 MSCAN12 block diagram

An HCS12 member may have up to five CAN modules and each CAN module occupies 4 bytes of memory space. The MSCAN register organization is shown in Figure below



II. METHODOLOGY

A. Setup & test MSCAN12 module in loop back mode:

Prior to connecting 2 boards together to communicate on CAN, the MSCAN12 configuration can be tested in "loop-back" mode. Since the module is configured in loop back mode, the MCU treats its own transmitted message as a message received from a remote node. This mode enables self-test operation, independent from any physical layer implementation.

The following steps are followed to setup and test MSCAN12 module:

1. Enter Initialization Mode by setting CAN0CTL0 = 0x01
2. Set MSCAN in loop back mode by setting CAN0CTL1 = 0xA0
3. Setup the sync jump width, TSEG1 and TSEG2 by setting the registers CAN0BTR0 and CAN0BTR1
4. Setup the receiver filters to receive ID 0x100 by setting the register CAN0IDAC
5. Check if the transmit buffer is full (CAN0TFLG) and if not full then load the ID in the IDR register (CAN0TXIDR0)
6. Set the Data length, priority and then fill the transmit buffer CAN0TFLG
7. Setup receive message ISR which get triggered when message is received. Inside the ISR, receive the data by reading the CAN0RXDSR0 register.

The data that is transmitted from the Tx bus is received in the receive buffer.

We confirmed that the messages that were being transmitted were being received at each node. After this, we connected the 2 boards together by using 2 wires.

B. Transmitting and receiving data to/from other board.

After successful communication in loop back mode we connected the two boards. Now here in this part Board one will transmit as well receive data and other board will also transmit and receive the data. For this we have connected two boards using wires. (CAN_HI => CAN_HI & CAN_LOW => CAN_LOW)

Now perform the same Initialization for the mode with loopback self-test disabled.

The following steps are followed to setup and test MSCAN12 module:

1. Enter Initialization Mode by setting CAN0CTL0 = 0x01
2. Set MSCAN in loop back mode by setting CAN0CTL1 = 0x80
3. Setup the sync jump width, TSEG1 and TSEG2 by setting the registers CAN0BTR0 and CAN0BTR1
4. Setup the receiver filters to receive ID 0x100 by setting the register CAN0IDAC
5. Check if the transmit buffer is full (CAN0TFLG) and if not full then load the ID in the IDR register (CAN0TXIDR0)
6. Set the Data length, priority and then fill the transmit buffer CAN0TFLG
7. Setup receive message ISR which get triggered when message is received. Inside the ISR, receive the data by reading the CAN0RXDSR0 register.

The data that is transmitted from the Tx bus is received in the receive buffer.

The normal mode can be broken up into 3 distinct steps – Initialization, Message Reception and Message Transmission. Below, the configuration of each register in every steps is discussed.

MSCAN Control Registers:

For CAN Initialization:

1) CAN0CTL0 = 0x01;
// Put MSCAN Module in Initialization Mode

	7	6	5	4	3	2	1	0
	RXFRM	RXACT	CSWAI	SYNCH	TIME	WUPE	SLPRQ	INITRQ
Reset:	0	0	0	0	0	0	0	1

2) CAN0CTL1 = 0x80; //enable CAN

	7	6	5	4	3	2	1	0
	CANE	CLKSRC	LOOPB	LISTEN	0	WUPM	SLPAK	INITAK
Reset:	0	0	0	1	0	0	0	1

3) CAN0BTR0 = 0xC3; // Synch Jump = 4 Tq clock Cycles, prescalar = 4

	7	6	5	4	3	2	1	0
	SJW1	SJW0	BRP5	BRP4	BRP3	BRP2	BRP1	BRP0
Reset:	0	0	0	0	0	0	0	0

4) CAN0BTR1 = 0x3A; // Set Number of samples per bit, TSEG1 and TSEG2

7	6	5	4	3	2	1	0	
SAMP	TSEG22	TSEG21	TSEG20	TSEG13	TSEG12	TSEG11	TSEG10	Reset value = 0x00

The bit time is determined by the oscillator frequency, the baud rate prescaler, and the number of time quanta (t_Q) clock cycles per bit. The bit time is given by the following expression:

$$\text{Bit time} = \frac{\text{prescaler value}}{f_{CANCLK}} \times (1 + \text{TimeSegment1} + \text{TimeSegment2})$$

5) CAN0IDAC = 0x10; // Set four 16-bit Filters

7	6	5	4	3	2	1	0	
0	0	IDAM1	IDAM0	0	IDHIT2	IDHIT1	IDHIT0	Reset value = 0x00

For CAN Reception:

1) CAN0RFLG = 0xC3; // Reset Receiver Flags

7	6	5	4	3	2	1	0	
WUPIF	CSCIF	RSTAT1	RSTAT0	TSTAT1	TSTAT0	OVRIF	RXF	Reset value = 0x00

2) CAN0RIER = 0x01; // Enable Receive Buffer Full Interrupt

7	6	5	4	3	2	1	0	
WUPIE	CSCIE	RSTATE1	RSTATE0	TSTATE1	TSTATE0	OVRIE	RXFIE	Reset value = 0x00

For CAN Transmission:

7) CAN0TFLG:

7	6	5	4	3	2	1	0	
0	0	0	0	0	TXE2	TXE1	TXE0	Reset value = 0x07

8) CAN0TBSEL: /* Select lowest empty buffer */

7	6	5	4	3	2	1	0	
0	0	0	0	0	TX2	TX1	TX0	Reset value = 0x00

C. Setup cyclic messages to be sent every 10ms & 15ms using Real Time Interrupt (RTI):

CAN messages can be sent cyclically after a fixed interval. We decided this interval to be 10ms & 15ms. Thus 2 messages with IDs 0x100 and 0x200 will be sent from Board A to Board B and Board B to Board A respectively.

The delay can be generated using the Real Time Interrupt (RTI). From calculations, we found out that when RTICTL = 0x49, for an 8 MHz oscillator, a period of 10.24ms can be achieved and RTICTL = 0x4F creates a period of about 15ms.

When the ISR is executed, the Data length, ID and data is transmitted, the Transmit function is invoked thus transmitting the messages every 10ms and 15ms (approximately).

D. Setup Dynamic values to be sent from both sides

To achieve the dynamic values in CAN communication Board A will transmit value of VARISTOR and it is displayed on the LCD of board B. Board B will transmit the status of the DIP switch which will be displayed on boards A. Thus 2 messages with IDs 0x100 and 0x200 will be sent from Board A to Board B and Board B to Board A respectively.

III. Experimental Setup

In order to be able to drive the bus, the HCS12's CAN TXD and RXD pins are connected to the CAN High-Speed CAN Transceiver IC MCP2551 (http://ww1.microchip.com/downloads/en/DeviceDoc/21667f.pdf). This has already been done on the Dragon-12 board.

The RXD is connected to the RXD of the HCS12 which is pin PM0 on the board. The TXD is connected to TXD of the HCS12 which is pin PM1 on the board. The CANH and CANL are the CAN High and CAN Low to drive the bus. The RS pin is hardwired to ground. A 5V supply is connected to the Vcc of the transceiver.

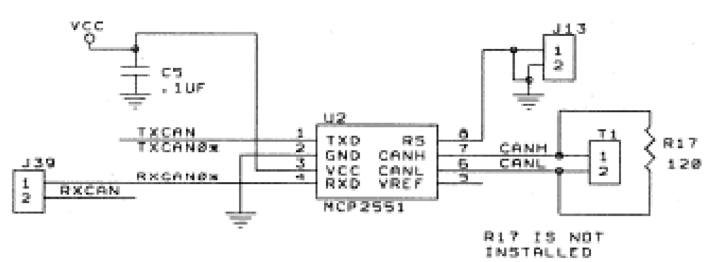


Figure 4 CAN Transceiver Connections

Since the plan is to setup a CAN network of 2 nodes, the CANH and CANL of 2 Dragon12 boards are connected as shown below.

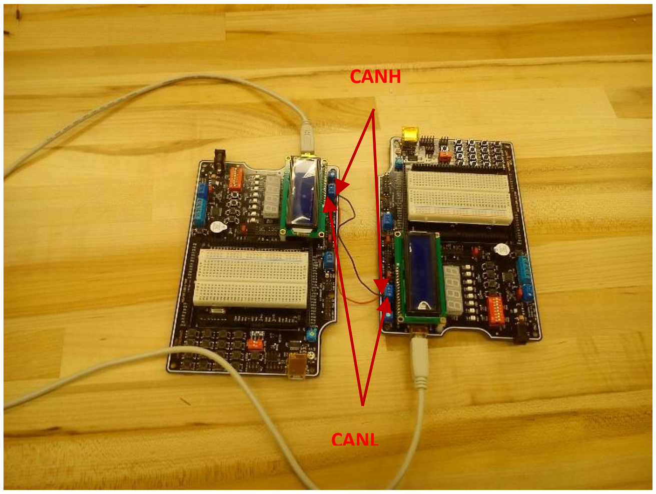


Figure 5 Setup with 2 Dragon-12 boards

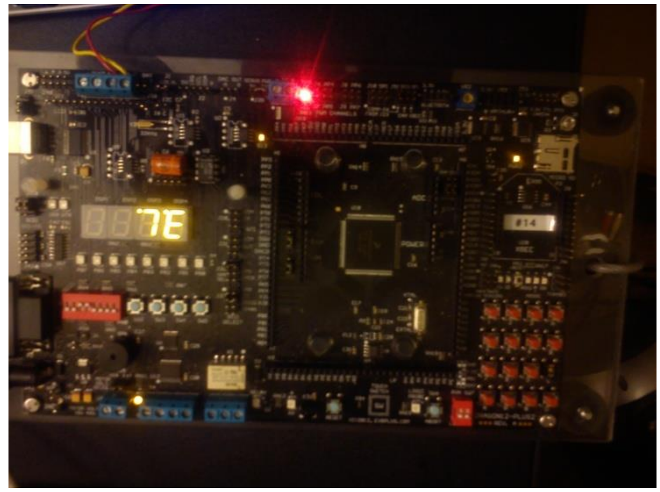


Figure 6 Node#1 showing the value of the DIP Switch Status

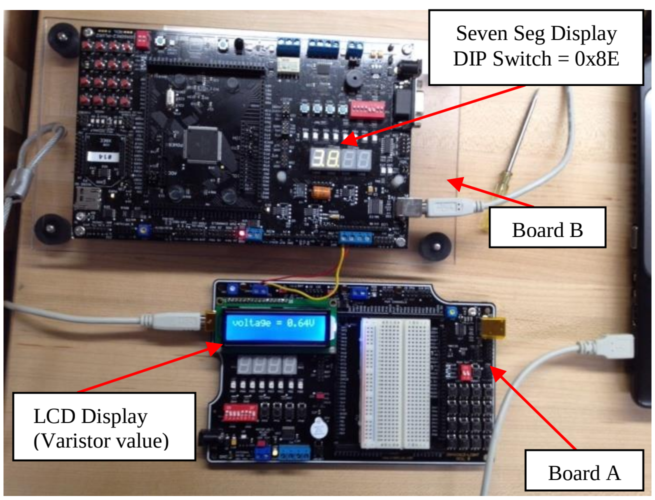


Figure 7 Setup showing the display and DIP switches

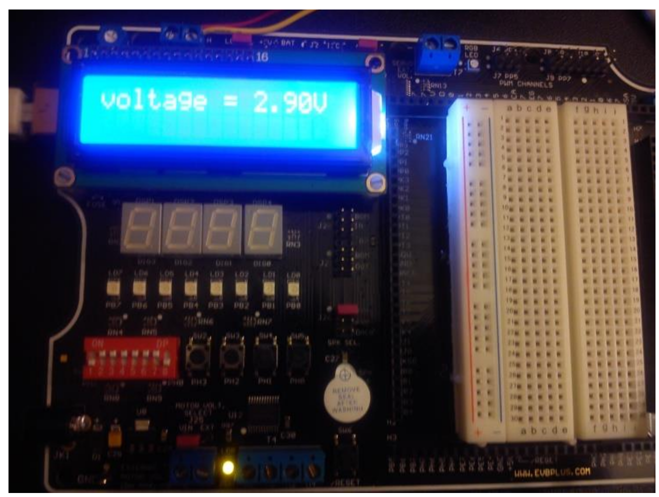


Figure 8 Node#2 showing the value which changes with changes in Varistor value

I. Results

The values sent from each board are correctly displayed on other board.

The below figure shows 0x100 Message on Logic Analyzer. The identifier field, the 8 bytes of data, CRC and other fields are seen.

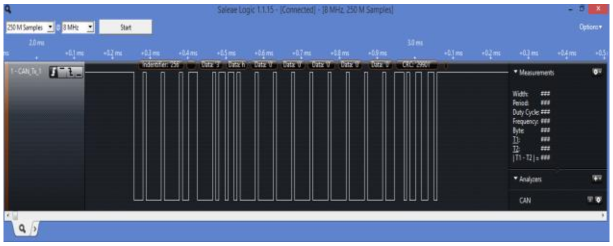


Figure 9 Message 0x100

The below figure shows 0x100 message which is sent every 15ms. The glitch seen is introduced by the transceiver.

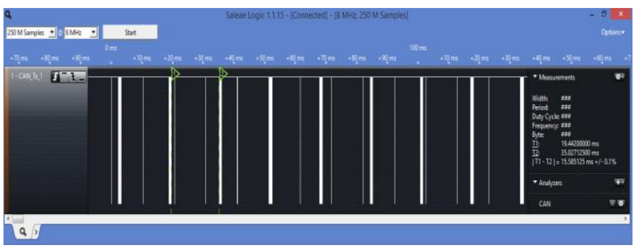


Figure 10Message 0x100 sent every 15ms

The below figure shows 0x200 Message on Logic Analyzer. The identifier field, the 8 bytes of data, CRC and other fields are seen.

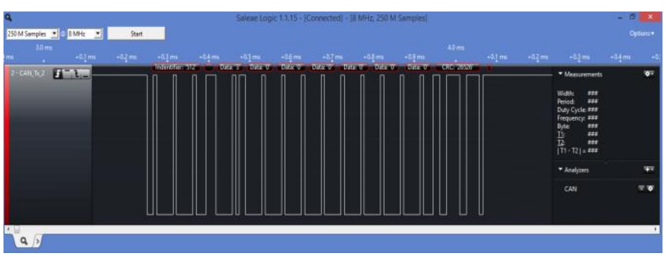


Figure 11 Message 0x200

The below figure shows 0x200 message which is sent every 10ms. The glitch seen is introduced by the transceiver.

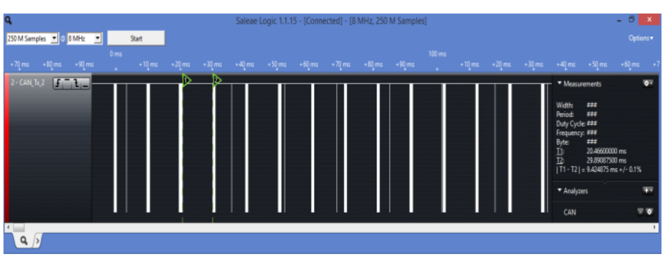


Figure 12Message 0x200 sent every 10ms

Thus it is seen that CAN is a robust protocol which accurately transmits and receives data using only 2 lines. It will be used for a long time till an even better and cost effective alternative is found.

The working of the project can be seen on video at this link: https://www.youtube.com/watch?v=VdNLzp-K8ME&feature=youtu.be

References

[1] The HCS12 - 9S12 - An Introduction to Software and Hardware Interfacing 2nd - Huang.

[2] Wikipedia (http://en.wikipedia.org/wiki/CAN_bus)

[3] Freescale Documentation

Conclusions

HCSCAN12 module provides an efficient and optimized method to send and receive CAN messages.