Freescale Semiconductor
Application Note

Document Number: AN4975
Rev. 0, 08/2014

Using MSCAN on the MagniV Family

by: Luis Olea

1 Introduction

This application note provides a simple MSCAN driver which will allow the user to enable CAN communications on any of the MagniV devices. It also provides a detailed explanation of the configuration of the MSCAN registers. This application note intends to provide a simple software driver for CAN communications on the MagniV devices. It does not intend to give a full in depth explanation of the CAN protocol. For more information on CAN, refer to application note AN1798: CAN Bit Timing Requirements and the Freescale Controller Area Network protocol page.

Contents

2 MSCAN module in MagniV devices

The MagniV devices have an integrated CAN communications module which is the S12MSCANV3. The module is a communication controller implementing the CAN 2.0 A/B protocol as defined in the Bosch specification dated September 1991.

For users to fully understand the MSCAN specification, it is recommended that the Bosch specification be read first to familiarize the reader with the terms and concepts contained within this document.

Though not exclusively intended for automotive applications, CAN protocol is designed to meet the specific requirements of a vehicle serial data bus, such as real-time processing, reliable operation in the EMI environment of a vehicle, cost-effectiveness, and required bandwidth.

MSCAN uses an advanced buffer arrangement resulting in predictable real-time behavior and simplified application software.

Some of the MagniV devices, such as the S12ZVC include an integrated CAN physical layer on chip, reducing the bill of materials and therefore reducing system cost. The rest of the MagniV family (and most CAN microcontrollers) require an external physical interface, to be able to communicate meeting the electrical specifications of the CAN bus.

Other MagniV parts aim to reduce system cost for specific applications, integrating features, such as voltage regulators, LIN physical layers, motor control gate drivers, stepper motor controllers, LCD segment display controllers, and more.

The following is a summary of the most notable MSCAN module features.

- Implementation of the CAN protocol version 2.0 A/B
    - Standard and extended data frames
    - Zero to eight bytes data length
    - Programmable bit rate up to 1 Mbps
- Five receive buffers with FIFO storage scheme
- Three transmit buffers with internal prioritization
- Flexible maskable identifier filter
- Wake up functionality with integrated low pass filter
- Loopback mode for self test operation
- Listen-only mode for CAN bus monitoring
- Bus off recovery functionality
- Signaling and interrupt capability for all error states (warning, error passive, bus off)
- Clock source can be chosen by software from either bus clock or oscillator clock
- Low power modes; sleep and power down

The following figure shows a block diagram of the MSCAN module.

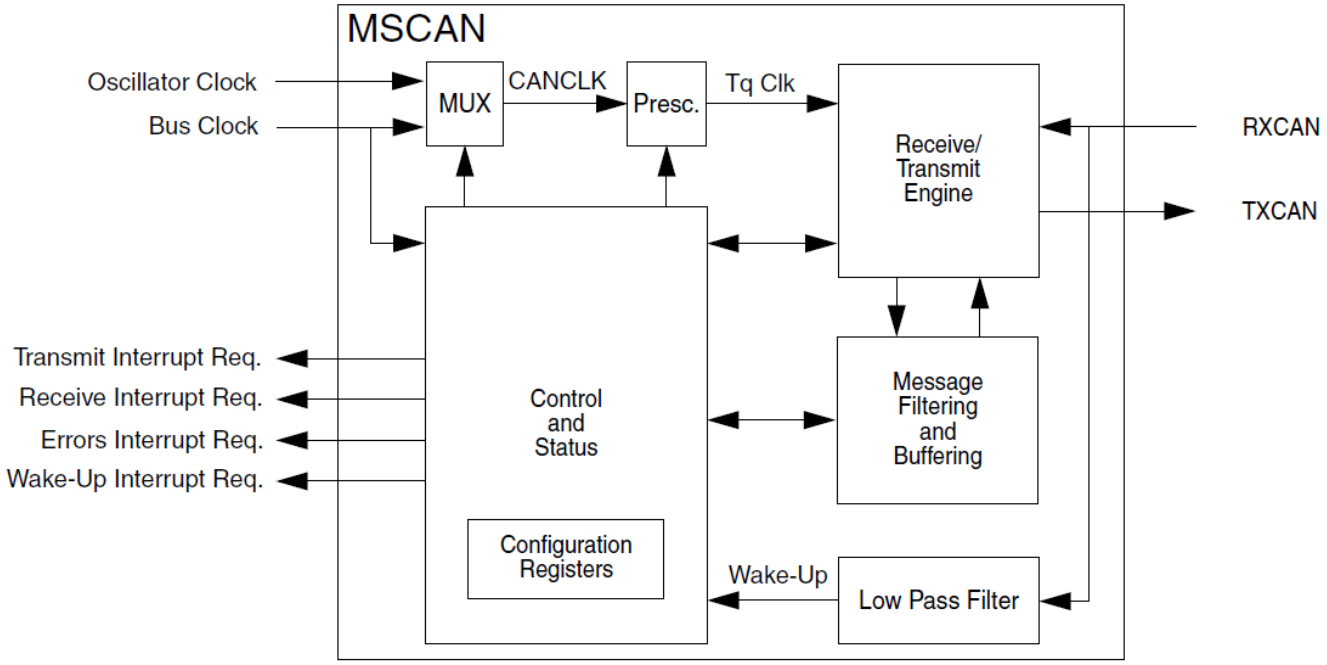

Figure 1. Block diagram of the MSCAN module

3 Initializing the MSCAN module

Before initializing the MSCAN module, we need to know what baud rate is going to be used. The baud rate is the speed of the bus; it states how many bits are sent per second. The following baud rates are some of the most typical speeds for CAN buses.

- 125 Kbps
- 250 Kbps
- 500 Kbps
- 1 Mbps

To configure which speed is used, change the macro CAN_BAUD_RATE in the file CAN_cfg.h. The options are CAN_X_KBPS where X is 125, 250, or 500 resulting in the corresponding bit rate. The MSCAN module is initialized by calling the “CAN_init” function.

NOTE

Further customization can be done which may cause each CAN receiver in the bus to sample the bus at different times, even if the baud rate is the same for all devices.

The following parameters are set to a default usable value in the driver, but may be customized in the final application if needed.

- Synchronization jump width: defines the maximum number of time quanta (MSCAN clock cycles) a bit can be shortened or lengthened to achieve resynchronization to data transitions.
- Time segments 1 and 2: These time segments within the bit time fix the number of clock cycles per bit time and the location of the sample point.

4 Calculating the baud rate

The software driver attached to this application note includes pre-calculated bit timings for some of the most common CAN baud rates. The user may want to define his own bit rate. To do this, the following equation is used.

$$Bit\ rate = \frac{f_{T_q}}{PROP\_SEG + PHASE\_SEG1 + PHASE\_SEG2}$$

Equation1. MSCAN bit rate calculation

5 Sending a CAN message

Before sending a CAN message, we need to know the following properties of the frame we want to generate.

- ID: The identifier of the message. The ID tells the other nodes in the CAN bus what does this message contain, or how to interpret the data on it. Nodes may discriminate based on ID and just ignore the message if the ID states that the information is not relevant for them. The ID can be 11-bit wide or 29-bit wide for standard and extended identifiers accordingly.
- DATA: The payload or actual bytes that are going to be contained in the frame. The data may be from 1 to 8 bytes in length.

After these two parameters are known, the “CAN_send” function can be called to put the message on the CAN bus. This function accepts the following parameters.

- ID: (16-bit wide) contains the desired ID
- PTR: (points to an unsigned 8-bit) contains the address of the first byte of data
- CNT: (8-bit wide) must contain a value between 1 and 8. Indicates how much bytes of data are going to be sent.

Before any call to “CAN_send”, the user must have called “CAN_init” to initialize the module. After calling the “CAN_send” function, the MSCAN module will transmit the requested data. The example code results in the CAN frame captured in Figure 2.

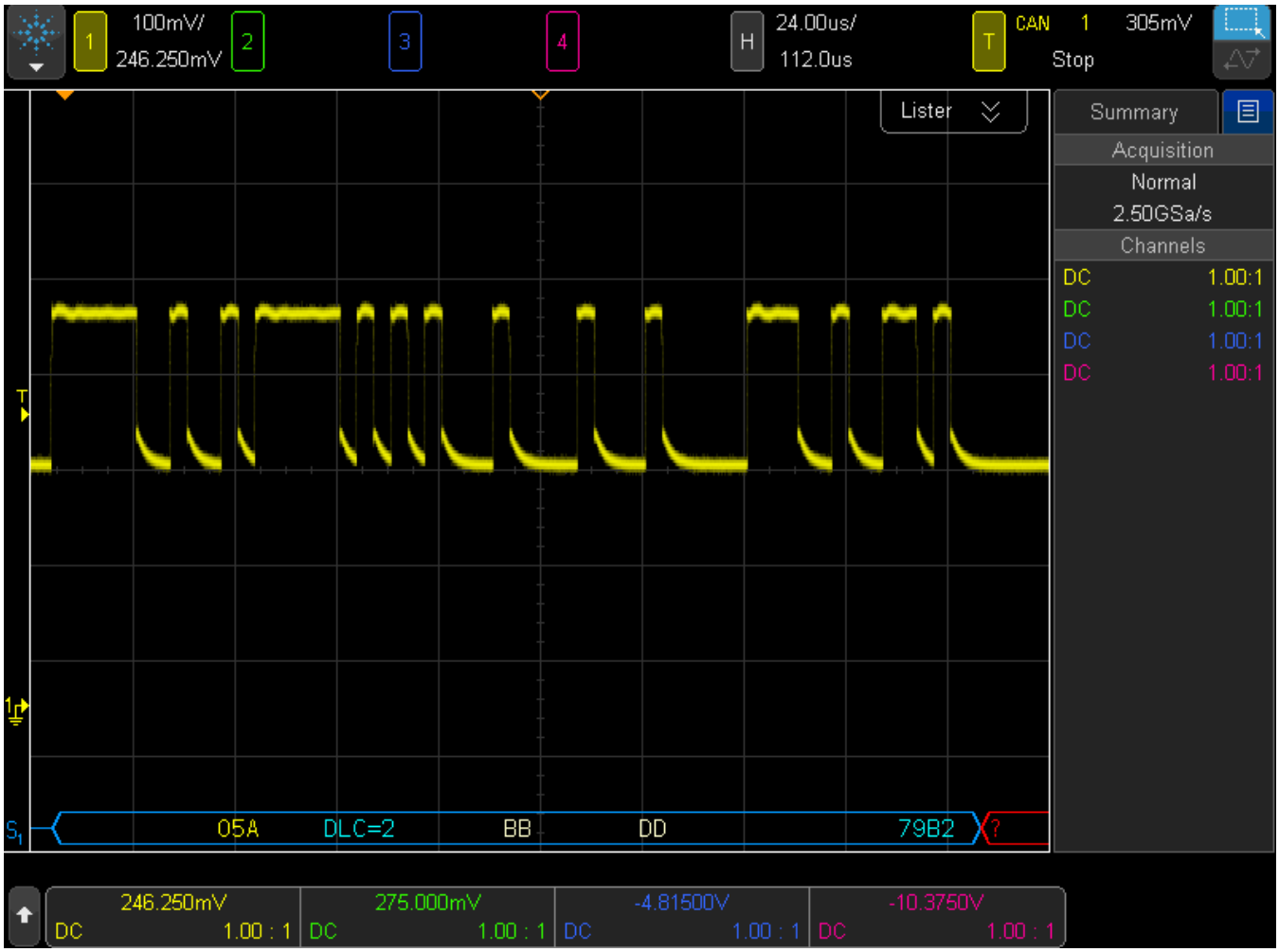

Figure 2. CAN frame sent by the MagniV device

6 Receiving a CAN message

Receiving a CAN message with the MSCAN module can be implemented using either interrupts or flag polling. In case of the attached CAN driver, interrupts are used. The CAN RX (receive) interrupt routine implements the following steps.

1) Copy the received CAN ID to a RAM variable
2) Copy the received CAN Data length to a RAM variable
3) Copy the received CAN Data to a RAM buffer
4) Execute a user callback function to process the data. This user function may take action on the received ID and Data.
5) Clear the CAN reception flag.

Before the CAN RX interrupt service routine can execute the callback function, the user must provide a pointer to this function. To do this the “CAN_set_rx_callback” is called. The only parameter of this function is a pointer to the user callback. The user callback must receive the following parameters.

- ID: a 16-bit variable where the CAN RX ISR will place the received frame ID.
- DATA[8]: an array of 8 locations, each location is 8-bit wide. This array will be filled by the CAN RX ISR using the data that was received through the CAN bus.
- LENGTH: an 8-bit value. The CAN Rx ISR will fill this value to indicate how much valid data is available on the DATA[8] buffer.

Examples of sending and receiving messages for different devices are available in the companion software files AN4975SW.zip.

7 References

The following references are available at freescale.com.

Product pages:

- MagniV
- S12VR
- S12ZVC
- S12ZVH
- S12ZVL
- S12ZVM

Application notes:

- AN4723: S12Z MagniV Bootloader
- AN4704: Sensorless BLDC Motor Control
- AN4722: Real Time Counter
- AN4851: High resolution timer and PWM
- AN4643: S12VR HW design guidelines
- AN4842: LIN enabled RGB LED lighting
- AN4841: LIN enabled ultrasonic distance measurement
- AN1798: CAN bit timing requirements

For more information, visit:

- www.freescale.com/CAN

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number AN4975
Revision 0, 08/2014