Setting Up MCP2515 using FPGA

Mohammad Hossein Shafizadegan

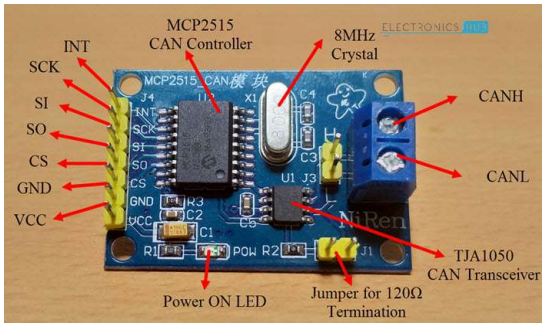
August 27, 2022

Contents

1 Introduction to MCP2515 module

MCP2515 is a CAN controller. This controller is a simple module that supports CAN protocol version 2.0B which can transfer data with rate of 1Mbps.
In order to communicate using a CAN Bus you need two of MCP2515 modules. One for the Transmitter and one for the Receiver.

This module includes a MCP2515 CAN Controller IC and also a TJA1050 CAN Transceiver IC. The MCP2515 IC is a CAN Controller itself and has pin required for SPI communication which you can use to communicate with local PCs like FPGA.



The MCP2515 is the main controller and it has 3 sub-modules including CAN Block, SPI Module and Control Logic Unit. CAN Block transmit and receive massages through the CAN Bus. Control Logic Unit is responsible for setting up the module and communication between different parts of module. The SPI Module handles communication using SPI protocol with local PCs.
TJA1050 IC connects the MCP2515 IC and the CAN Bus.

2 MCP2515 modes of operation

MCP2515 IC has 5 modes of operation :

1. Configuration Mode
2. Normal Mode
3. Sleep Mode
4. Listen-Only Mode
5. Loop-back Mode

2.1 Configuration Mode

When you power on the MCP2515, it will automatically goes to Configuration mode. You can also change mode to Configuration mode by sending a SPI instruction and set the value of CANCTRL[7:5] to 100.
In order to initialize this IC you have to set some internal registers.

1. First we set Configuration registers including CNF1, CNF2, CNF3. CNF1 is used to control baud rate prescaler. CNF2 deals with propagation segment.
2. Next step is to set TXRTCTRL register. By setting TXRTCTRL[2:0] to 1, TXnRTS pins will be used to request message transmission of TXBn buffers.
3. The CANINTE register contains the individual interrupt enable bits for each interrupt source. When an interrupt occurs, the INT pin is driven low by the MCP2515 and will remain low until the interrupt is cleared by the MCU. We should activate this feature by setting CANINTE register value.
4. Each MCP2515 module must have and ID. When a message is received by the module, it will compare ID bits of the message. If the message ID fits the module ID, it means that the message is related to this node and it will stored in receive buffers. We have to set this ID for the module by changing appropriate filter register. It is also needed to set the Mask register that defines which ID bits are used for comparison.

Note : these registers are only modifiable in configuration mode.

2.2 Loop-back Mode

Loop-back mode will allow internal transmission of messages from the transmit buffers to the receive buffers without actually transmitting messages on the CAN bus. This mode can be used in system development and testing.
In order to change operating mode to Loop-back mode you have to send a SPI byte write instruction and set the value of CANCTRL[7:5] to 010.

2.3 Normal Mode

Normal mode is the standard operating mode of the MCP2515. In this mode, MCP2515 will transmit messages over the CAN bus.

Other modes of operation are not mentioned in this article.

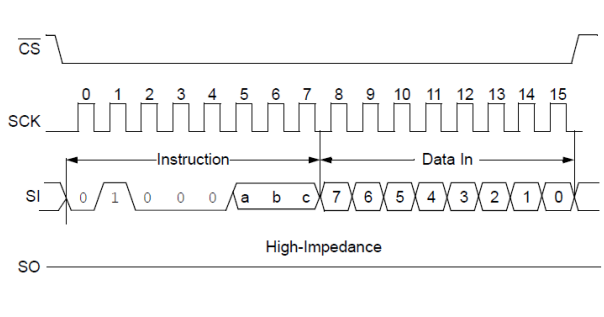
3 How MCP2515 works

3.1 Transmit Data

In order to send and transmit data through the CAN bus you have to do four steps bellow :

1. **Load transfer buffers**
   First step for transmitting message is to lead the desired transfer buffer. In order to do so, you have to send Load TX Buffer SPI Instruction. You can see this instruction's format bellow :

FIGURE 12-5: LOAD TX BUFFER INSTRUCTION



a	b	c	Address Points to	Addr
0	0	0	TX Buffer 0, Start at TXB0SIDH	0x31
0	0	1	TX Buffer 0, Start at TXB0D0	0x36
0	1	0	TX Buffer 1, Start at TXB1SIDH	0x41
0	1	1	TX Buffer 1, Start at TXB1D0	0x46
1	0	0	TX Buffer 2, Start at TXB2SIDH	0x51
1	0	1	TX Buffer 2, Start at TXB2D0	0x56

2. The next step is to Set the message ID. By doing this you can set who is this message for.
   Use byte write SPI instruction and set appropriate address and value.

3. In this step you have to send Ready To Send (RTS) SPI instruction. The message will not be sent unless you send this SPI instruction.

4. The last step is to be sure that the message has been sent successfully.
   We check the value of TXERR bit of TXBnCTRL register to see whether there is an error in transmission. If an error occurs we try to send message again for particular times for example 10 times. If transmission still failed we abort this message.

3.2 Receive Data

Message Reception includes these steps :

1. Whenever a message pass the acceptance filters and be stored in receive buffers and interrupt will be occurs and value of INT pin will be set to '0'.
   Under this condition we will send Read RX Buffer SPI Instruction and will receive the data through SPI module.

2. after receiving a message INT pin will be '0' and won't be reset to '1' so in order to be able to receive more new messages you should related bit if CANINTF register by sending Byte modify SPI Instruction.

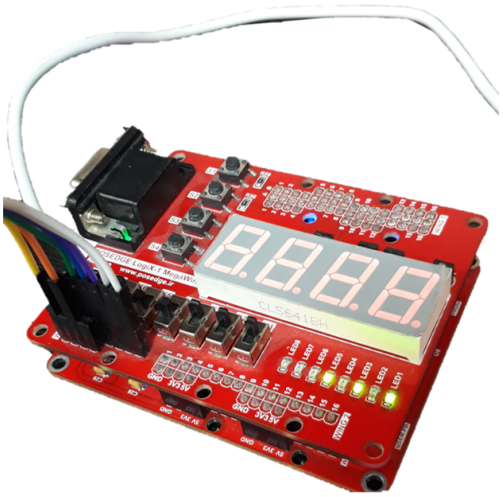
Note : Click here to Download MCP2515 data sheet

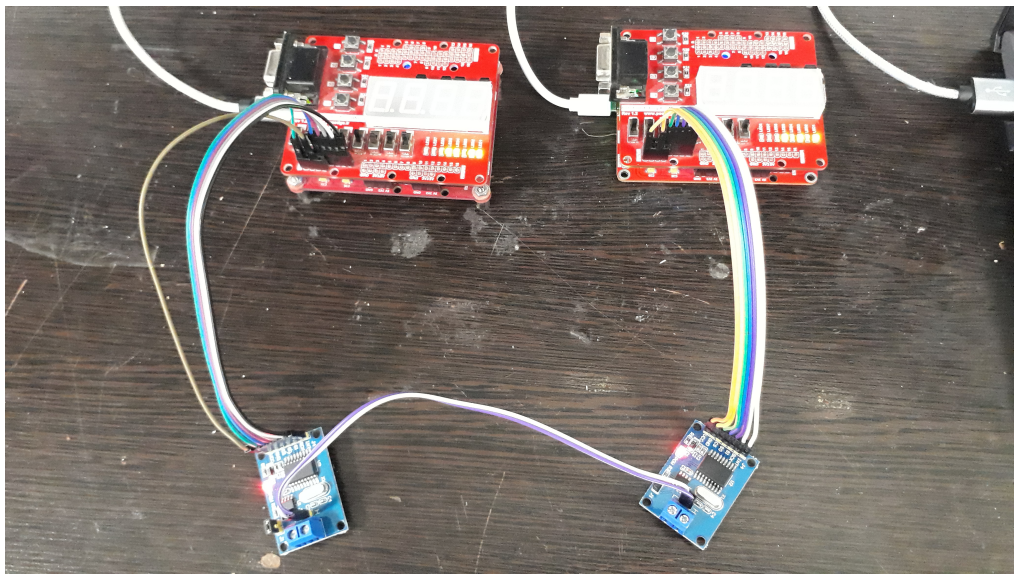
4 FPGA Code

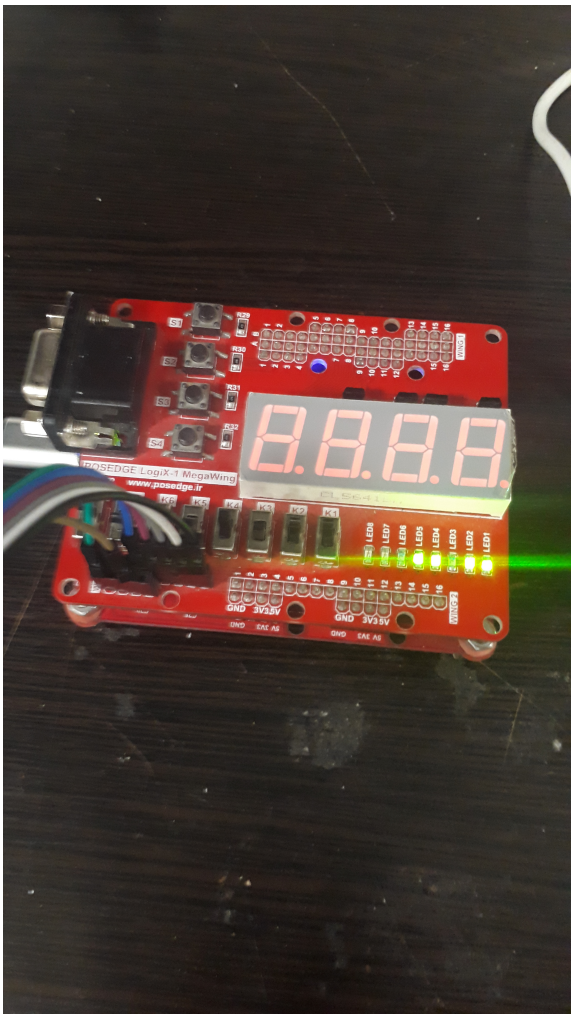
4.1 Overview

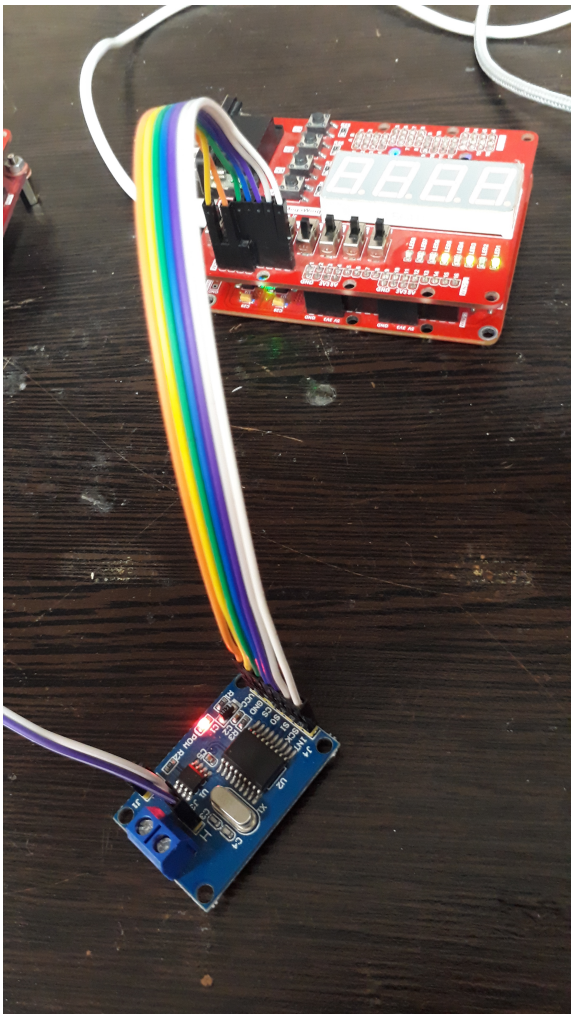
In this section we will define a mini project.
The subjective is to send a byte from and FPGA-1 to another one.
Show the received data by LEDs on the FPGA-2.
Defining a byte by deep switches on the FPGA-2.
Add this value to the received data and send the result to the FPGA-1.

There is a button for reset(Btn1) and also a button for sending message(Btn4). In this project we have use two posedge boards including Xilinx spartan-6 FPGA and two MCP2515 modules.









4.2 Code

As mentioned before, MCP2515 communicates through SPI protocol so we need to define a VHDL module for this. According to SPI instruction list, instructions include 8, 16, 24, 32 bits to transfer via SPI pins so our VHDL module has a parameter N that defines number of bits to transfer and then in the main code we instantiate 4 of these module. You can use codes for SPI controller available on the Internet.

After that we create another VHDL module named "MCP_driver". In this module we control the can IC for example, we configure the IC and then do whatever needed to send and receive data through the CAN bus.

Here we create a SPI component from the code we wrote before :

```
component spi_master
generic(
data_length : integer := 8
);
port(
clk      : in      std_logic;                                    --system clock
reset_n  : in      std_logic;                                    --asynchronous active low reset
enable   : in      std_logic;                                    --initiate communication
miso     : in      std_logic;                                    --master in slave out
sclk     : out     std_logic;                                    --spi clock
ss_n     : out     std_logic;                                    --slave select
mosi     : out     std_logic;                                    --master out slave in
busy     : out     std_logic;                                    --master busy signal
tx     : IN      std_logic_vector(data_length-1 downto 0);  --data to transmit
rx      : out     std_logic_vector(data_length-1 downto 0);  --data received
tx_end   : out     std_logic                           --data received
);

end component;

```

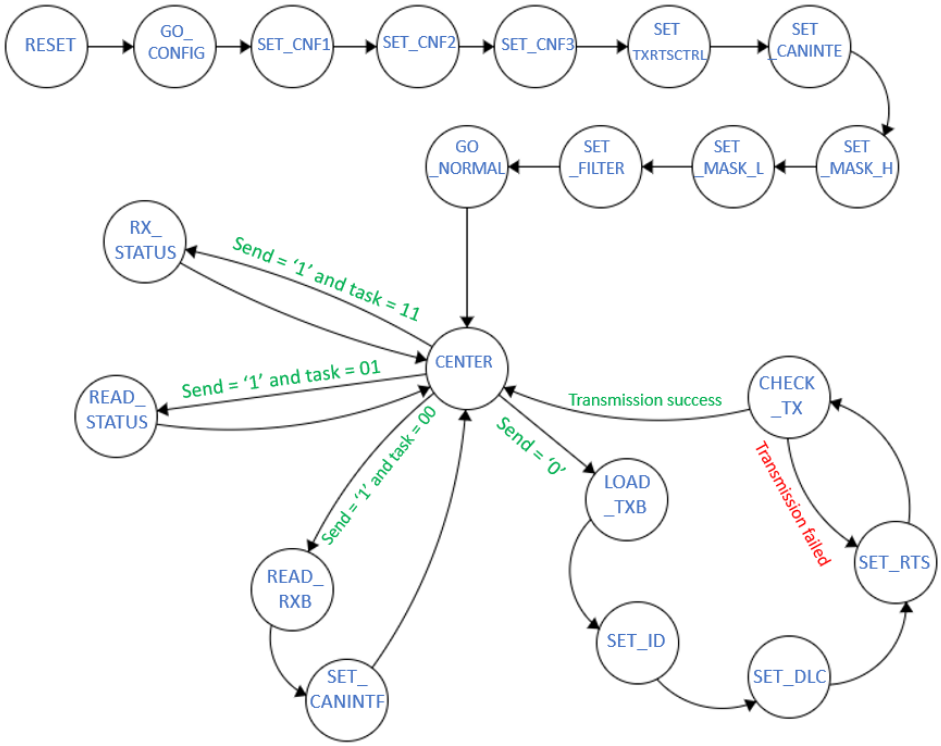
In this module we create a FSM to handle configuration, transmission, reception and ... of MCP2515 IC.

```
type FSM_STATE is (RESET, GO_CONFIG, SET_CNF1, SET_CNF2, SET_CNF3, CHECK_TX, READ_REG,
SET_MASK_H, SET_MASK_L,
SET_TXRTSCTRL, SET_CANINTE, GO_NORMAL, LOAD_TXB, CENTER, SET_ID_H, READ_ID, SET_DLC,
SET_RTS, READ_RXB, READ_STATUS, RX_STATUS, SET_CANINTF, SET_FILTER);

```

For example the code for going to configuration mode is shown bellow :

```
------------------ Go to Config mode of Operation ------------------
when GO_CONFIG =>
if tx_end_24 = '1' then
c_state <= SET_CNF1;
else
c_state <= GO_CONFIG;
end if;

data_in_24 <= "00000010" & "00001111" & "10000000";
spi_sel <= 3;

if busy_24 = '0' then
busy_counter <= busy_counter + 1;
if busy_counter = 3 then
enable_24 <= '1';
busy_counter <= 1;
end if;
else
enable_24 <= '0';
end if;

```

FSM block diagram :

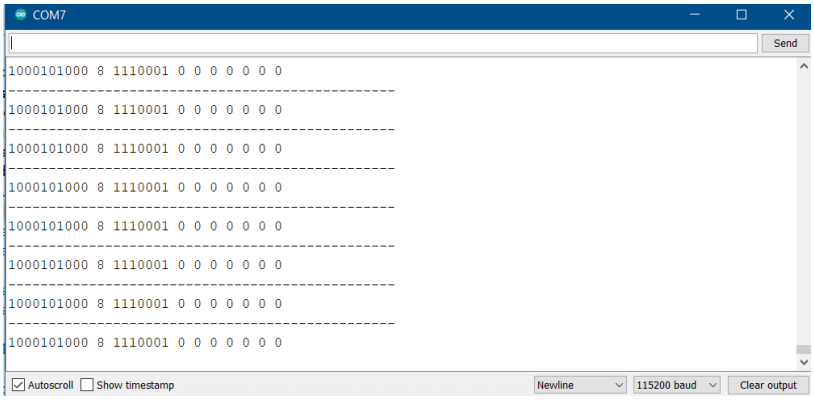


After these steps we create a VHDL file as our top module and instantiate a MCP_Driver component.

```
    receiver_module : MCP_driver
    port map(
    clk => clk,
    rst => rst,
    SO => SO,
    SI => SI,
    SCLK => SCLK,
    SS => CS,
    INT => INT,
    data_in => data_in,
    data_out => data_out,
    task => task,
    send => send
    );

    process(clk)
    begin
    if rising_edge(clk) then
      if INT = '0' then
      LED_reg <= data_out;
      end if;
    end if;
    end process;
    data_in <= LED_reg + ("00" & num);
    o_LED <= LED_reg;

```

It is also possible to communicate between a FPGA board like spartan-6 and an Arduino Uno board using CAN protocol.
Here we assume the transmitter is the Arduino Uno and the receiver is our spartan-6 FPGA.
We send a byte using Arduino serial interface and also we receive new data from FPGA using the send button and show the received data using serial.
Here we just use CAN and SPI libraries available in the Internet.

```
#include <SPI.h>
#include <mcp2515.h>

struct can_frame canMsg1;
struct can_frame canMsg;

MCP2515 mcp2515(10);

int temp;

void setup() {
  canMsg1.can_id  = 0x88;
  canMsg1.can_dlc = 1;
  canMsg1.data[0] = 0x00;

  while (!Serial);
  Serial.begin(115200);

  mcp2515.reset();
  mcp2515.setBitrate(CAN_125KBPS, MCP_8MHZ);
  mcp2515.setNormalMode();
  mcp2515.sendMessage(&canMsg1);
  Serial.println("Example: Write to CAN");
}

void loop() {
  if (mcp2515.readMessage(&canMsg) == MCP2515::ERROR_OK) {
    Serial.println("-----------------------------------------------------------------");
    Serial.print(canMsg.can_id, BIN); // print ID
    Serial.print(" ");
    Serial.print(canMsg.can_dlc, HEX); // print DLC
    Serial.print(" ");

    for (int i = 0; i<canMsg.can_dlc; i++)  {  // print the data
      Serial.print(canMsg.data[i],BIN);
      Serial.print(" ");
    }

    Serial.println();
  }

  if (Serial.available() > 0){
    temp = Serial.read();
    if (temp != 10){
      canMsg1.data[0] = temp;
      //       Serial.println(temp);
      mcp2515.sendMessage(&canMsg1);
    }
  }

}

```


COM7
Send
1000101000 8 1110001 0 0 0 0 0 0 0
Autoscroll
Show timestamp
Newline
115200 baud
Clear output

-