

SPI COMMUNICATION TMS320F28379S CONTROLLER

I am using tms320f28379s C2000 series controller to interface SPI via NVRAM 23Icv1024. Configured the FIFO mode as per the example code provided in the control suite.

In spi, 8 bit data length is configured, the transmit data is left justified because it is less than 16bit word in the c2000 series.

While writing the data through spi into NVRAM, the data are written into nvram successfully.

Problem - While reading the data from nvram, 0xFF reads first and then only the original data (example: 0x25) is displayed. I am expecting 0x25 to be displayed initially and also subsequently.

Please find below the spi configuration and the screen shots.

Spi configuration details

```

void InitSpibGpio()
{
    EALLOW;
    GpioCtrlRegs.GPAPUD.bit.GPIO24 = 0; //Enable pull_upon GPIO24
    GpioCtrlRegs.GPAPUD.bit.GPIO25 = 0; //Enable pull_upon GPIO25
    GpioCtrlRegs.GPAPUD.bit.GPIO26 = 0; //Enable pull_upon GPIO26
    GpioCtrlRegs.GPAPUD.bit.GPIO27 = 0; //Enable pull_upon GPIO27

    GpioCtrlRegs.GPAQSEL2.bit.GPIO24 = 3; //Asynch INPUT GPIO24
    GpioCtrlRegs.GPAQSEL2.bit.GPIO25 = 3; //Asynch INPUT GPIO24
    GpioCtrlRegs.GPAQSEL2.bit.GPIO26 = 3; //Asynch INPUT GPIO24
    GpioCtrlRegs.GPAQSEL2.bit.GPIO27 = 3; //Asynch INPUT GPIO24

    GpioCtrlRegs.GPAGMUX2.bit.GPIO24 = 1; //Configure GPIO24 as SPI pin - MOSI GMUX Sel
    GpioCtrlRegs.GPAGMUX2.bit.GPIO25 = 1; // Configure GPIO25 as SPI pin - MISO GMUX Sel
    GpioCtrlRegs.GPAGMUX2.bit.GPIO26 = 1; // Configure GPIO26 as SPI_CLK- GMUX selection
    GpioCtrlRegs.GPAGMUX2.bit.GPIO27 = 1; // Configure GPIO27 as SPI_EN- GMUX selection

    GpioCtrlRegs.GPAMUX2.bit.GPIO24 = 2; // GPIO24 is MOSI
    GpioCtrlRegs.GPAMUX2.bit.GPIO25 = 2; // GPIO25 is MISO
    GpioCtrlRegs.GPAMUX2.bit.GPIO26 = 2; // GPIO26 is CLK
    GpioCtrlRegs.GPAMUX2.bit.GPIO27 = 2; // GPIO26 is CLK

    GpioCtrlRegs.GPADIR.bit.GPIO24 = 1; // Enable internal pull-up for the selected SPI pin -MOSI
    GpioCtrlRegs.GPADIR.bit.GPIO25 = 1; // Enable internal pull-up for the selected SPI pin -MISO
    GpioCtrlRegs.GPADIR.bit.GPIO26 = 1; // Enable internal pull-up for the selected SPI pin - CLK
    GpioCtrlRegs.GPADIR.bit.GPIO27 = 1; // Enable internal pull-up for the selected SPI pin - EN

    EDIS;
}

void InitSpi_B() // Initialize SPI-B
{
    EALLOW;
    SpibRegs.SPICCR.bit.SPISWRESET = 0; // Set reset low before configuration changes
    SpibRegs.SPICCR.bit.CLKPOLARITY = 1; // Clock polarity selection (0 == rising, 1 == falling)
    SpibRegs.SPICCR.bit.HS_MODE = 0; // SPI High Speed mode Enable Bit (0 == Disable, 1 == Enable)
    SpibRegs.SPICCR.bit.SPILBK = 0; // SPI Loop back Mode Select (0 == Disable, 1 == Enable)
}

```

```

    SpibRegs.SPICCR.bit.SPICHAR      = (8-1); // Character length control bit - 8-bit

    SpibRegs.SPICTL.bit.OVERRUNINTENA = 0; // Overrun Interrupt Enable bit (0 == Disable, 1 == Enable)
    SpibRegs.SPICTL.bit.CLK_PHASE     = 0; // SPI CLOCK PHASE SELECT (0 == normal, 1 == delayed)
    SpibRegs.SPICTL.bit.MASTER_SLAVE  = 1; // Enable Master mode (0 == slave, 1 == master)
    SpibRegs.SPICTL.bit.TALK          = 1; // Enable Transmission (Talk)
    SpibRegs.SPICTL.bit.SPIINTENA    = 0; // SPI interrupts are disabled

    SpibRegs.SPIBRR.bit.SPI_BIT_RATE = SPI_BRR; // Set the baud rate

    SpibRegs.SPIPRI.bit.FREE = 1; // Set FREE bit // Halting on a breakpoint will not halt the SPI

    SpibRegs.SPICCR.bit.SPISWRESET = 1; // Release the SPI from reset WHEN the value = 1 SPI ready to transmit and receive
    EDIS;
}

void spi_fifo_init() // Initialize SPI FIFO registers
{
    SpibRegs.SPIFFTX.all = 0xE040; // FIFO Transmit register 1.Interrupt clear 2.Release transmit FIFO from reset 3.Enhancement enable 4.SPI FIFO resume transmit or receive.
    SpibRegs.SPIFFRX.all = 0x2044; // SPI FIFO Receive Register // To clear receive FIFO interrupt ,re enable receive FIFO operation.
    SpibRegs.SPIFFCT.all = 0x0; // SPI FIFO Control Register // delayed transfer.
    InitSpi_B(); // Initialize core SPI registers
}

// END OF SPI CONFIGURATION //

//SPI Transmit function //
unsigned char rdata;

unsigned char spi_xmit(unsigned char sdata)
{
    SpibRegs.SPITXBUF = (sdata << 8); // SPI Serial Output Buffer Register
    rdata=SpibRegs.SPIRXBUF; // Right shift no need
    return rdata;
}

//Clock configuration//

#define SPI_CLK_in_KHz 500 // 2500 -> 2.5 MHz // 1000 -> 1MHz

#if CPU_FRQ_200MHZ
#define LSPCLK_in_KHz 50000 // (200 MHz / 4) = 50 MHz
#define SPI_BRR (LSPCLK_in_KHz / SPI_CLK_in_KHz) - 1
#endif

// End of clock configuration//

```

```

// Nvram write function //



//*****
//      Function : chip_select_NVRAM1()
//      Purpose  : Sets the SS pin to the low state in order to enable SPI mode
//*****
*****



void chip_select_NVRAM1(void)
{
    GpioDataRegs.GPACLEAR.bit.GPIO27 = 1;
}

//*****
//      Function : chip_deselect_NVRAM1()
//      Purpose  : Sets the SS pin to the high state in order to disable SPI
mode
//*****
*****



void chip_deselect_NVRAM1(void)
{
    GpioDataRegs.GPASET.bit.GPIO27 = 1;
}

//*****
//      Function : NVRAM1_send_address()
//      Purpose  : sending address in a specified format as mentioned in
datasheet
//*****
*****



void NVRAM1_send_address(unsigned long int address)
{
    spi_xmit((address >> 16) & 0xFF);
    spi_xmit((address >> 8) & 0xFF);
    spi_xmit((address) & 0xFF);
}

//*****
//      Function : NVRAM1_set_mode()
//      Purpose  : Send the address to the mode register in order to Set the
memory operation mode (byte, page, sequential <default>)
//*****
*****



void NVRAM1_set_mode(unsigned char mode)
{
    chip_select_NVRAM1();
    spi_xmit(RAM_INS_WRMR);
    spi_xmit(mode);
    chip_deselect_NVRAM1();
}

```

```

//*****
***** Function : NVRAM1_read_mode()
// Purpose : To know which mode is selected for the write operation among
byte, page, sequential operations
//*****
*****



unsigned char NVRAM1_read_mode(void)
{
    unsigned char data;

    chip_select_NVRAM1();
    spi_xmit(RAM_INS_RDMR);
    data = spi_xmit(0);
    chip_deselect_NVRAM1();

    return data;
}

//*****
***** Function : ram_write_data()
// Purpose : To write data into the SRAM page wise
//*****
*****


void ram_write_data(unsigned long int write_address, unsigned char *write_buffer, unsigned int
write_buffer_size) { //function to write data on the specified address
    unsigned long int write_page;
    chip_select_NVRAM1();
    spi_xmit(RAM_INS_WRITE);
    NVRAM1_send_address(write_address);
    for (write_page = 0; write_page < write_buffer_size; ++write_page) {
        spi_xmit(* (write_buffer + write_page));
    }
    chip_select_NVRAM1();
}

//*****
***** Function : read_data()
// Purpose : To read data into the SRAM page wise
//*****
*****

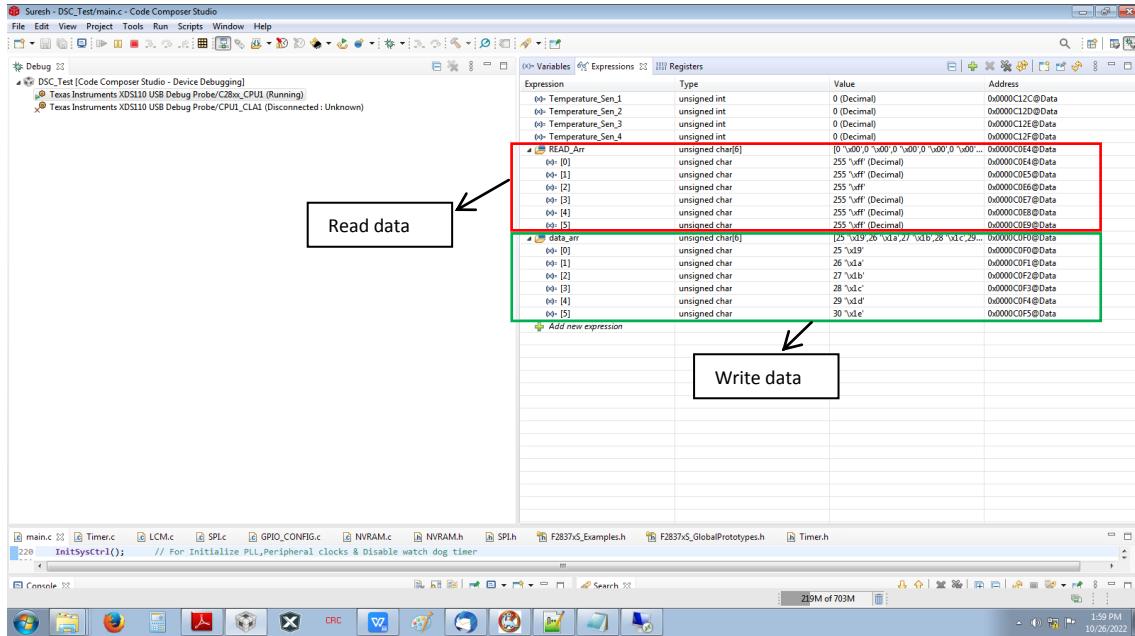

void read_data(unsigned long int Read_address, unsigned char *read_buffer, unsigned int
read_buffer_size) {
    chip_select_NVRAM1();
    spi_xmit(RAM_INS_READ);
    NVRAM1_send_address(Read_address);
    while (read_buffer_size) {
        *read_buffer = spi_xmit(0);
        ++read_buffer;
        --read_buffer_size;
    }
    chip_deselect_NVRAM1();
}

// END OF NVRAM FUNCTION //

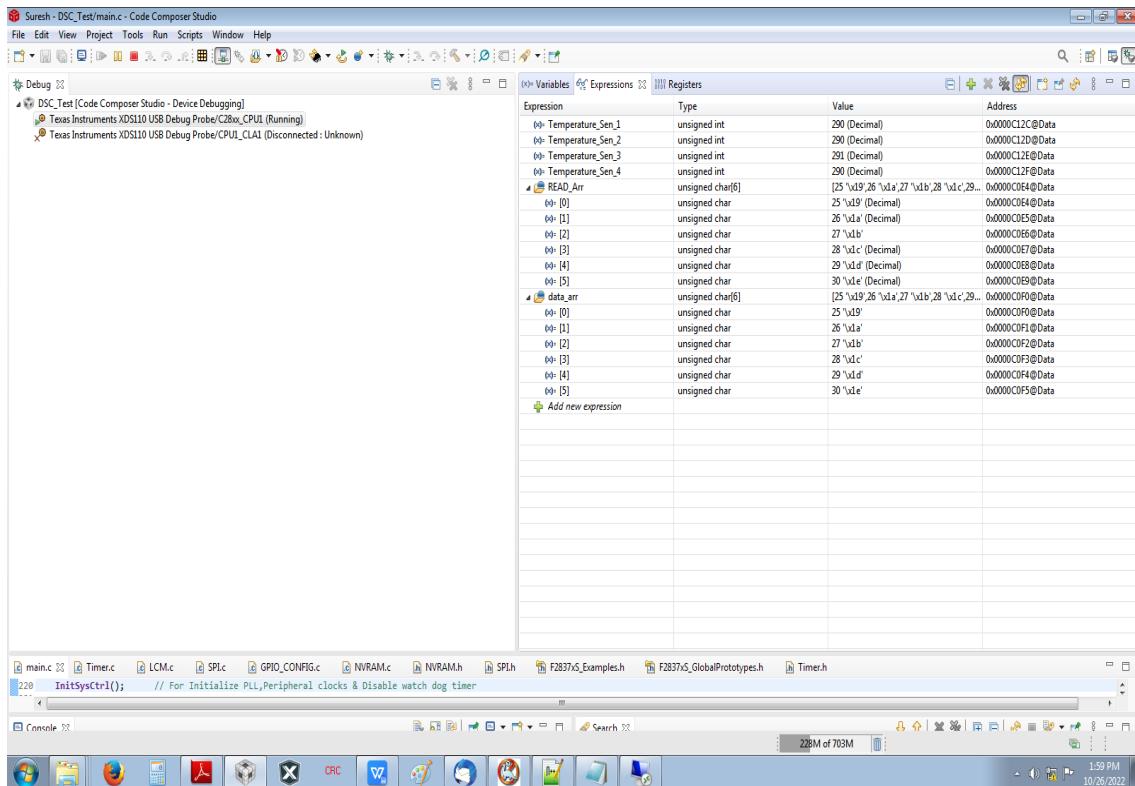
```

Result Screenshots:

Attached below is the screen shot of received data initially "0xFF"



After that it is displaying the originally receive data 0x25



So please look at this and let me know the solution.

Notes: Without FIFO mode, code is not working it shows always 0xFF.