

# SPI and I2C communication with CMR3000-D01 using MSP430 ultra low-power microcontroller

## 1 INTRODUCTION

The objective of this document is to show how to set up SPI/I2C communication between VTI Technologies CMR3000-D01 digital angular rate sensor component and a Texas Instruments MSP430 microcontroller. In the code examples:

- The MSP430 MCU is configured
- CMR3000-D01 measurement mode is activated
- An interrupt is used to read angular rate output data registers when new data is available. When using the interrupt with I2C bus please note that Fast Mode (400 kbit/s) must be used to get all samples, with Standard Mode (100 kbit/s) the bus does not have time to handle the communication until a new interrupt is generated since CMR3000-D01 update rate is 2 kHz.

Please refer to document "CMR3000-D0X Product Family Specification 82112900" for further information on CMR3000-D01 register addressing and SPI/I2C communication. For MSP430 related information please see Texas Instruments web pages (<http://www.ti.com>).

## 2 DEVELOPMENT HARDWARE

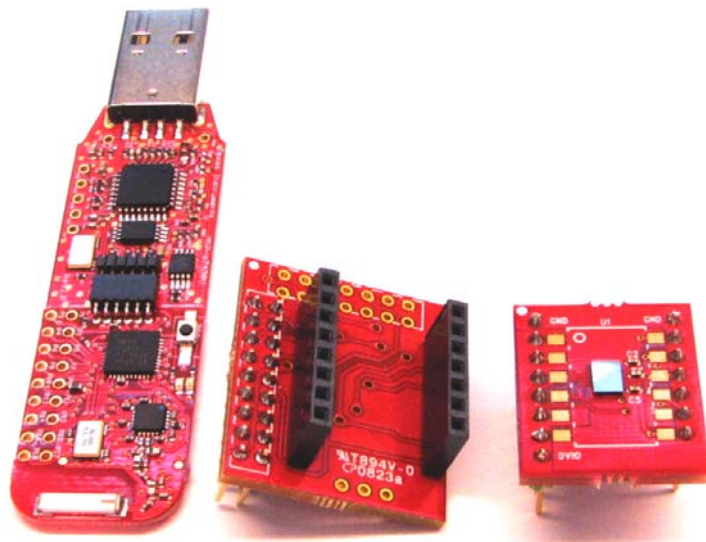


Figure 1. From left to right, eZ430-RF2500, adapter PCB VTI29631 and CMR3000-D01 chip carrier

eZ430-RF2500 connector

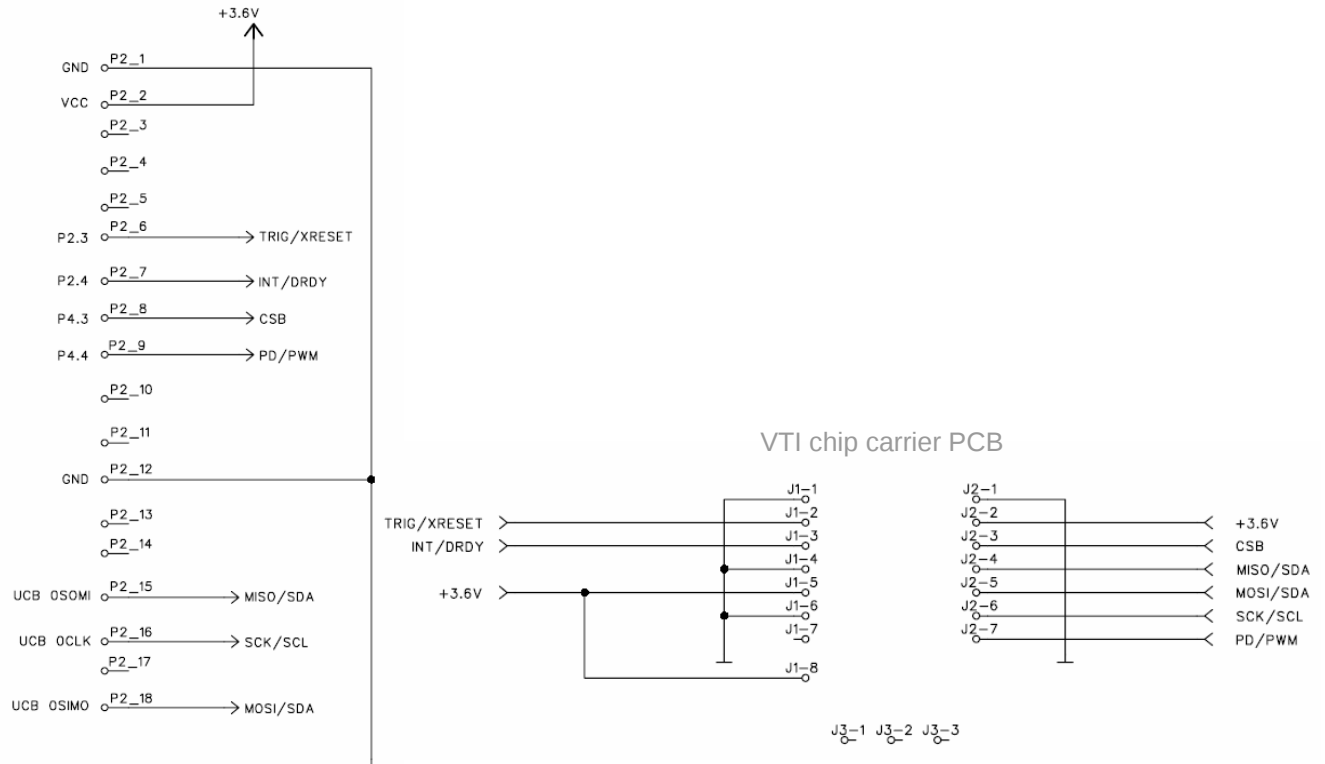


Figure 2. Adapter PCB VTI29631 circuit diagram

### 3 C-CODE

The example code is written for Texas Instruments eZ430-RF2500 Development Tool (MSP430F2274) using IAR Embedded Workbench IDE v5.10 for MSP430. To interface CMR3000-D01 to the eZ430-RF2500 an adapter PCB VTI29631 is used (available through Hy-Line's web shop (<http://www.hy-line.de>)).

The C-language software examples for SPI and I2C buses on the next pages shows an easy way to implement communication with the CMR3000-D01. Universal Serial Communication Interface peripheral inside the MSP430F2274 is used for communication.

The SPI code sets up an interrupt to wake up the MCU from low power mode thru CMR3000-D01 INT-pin when new data from the sensor is available. When no data is available (no interrupt) the MCU is kept in low power mode 4 (LPM4) to achieve lower current consumption. Clocks are set up so that the MCU clock frequency is 16 MHz, SPI clock is 250 kHz and I2C clock 400 kHz. Multiple read operation mode (decrement reading) is used to read all angular rate output data registers (0Ch...11h) within a single SPI/I2C read frame.

### 3.1 SPI Interface Example

Code flowchart:

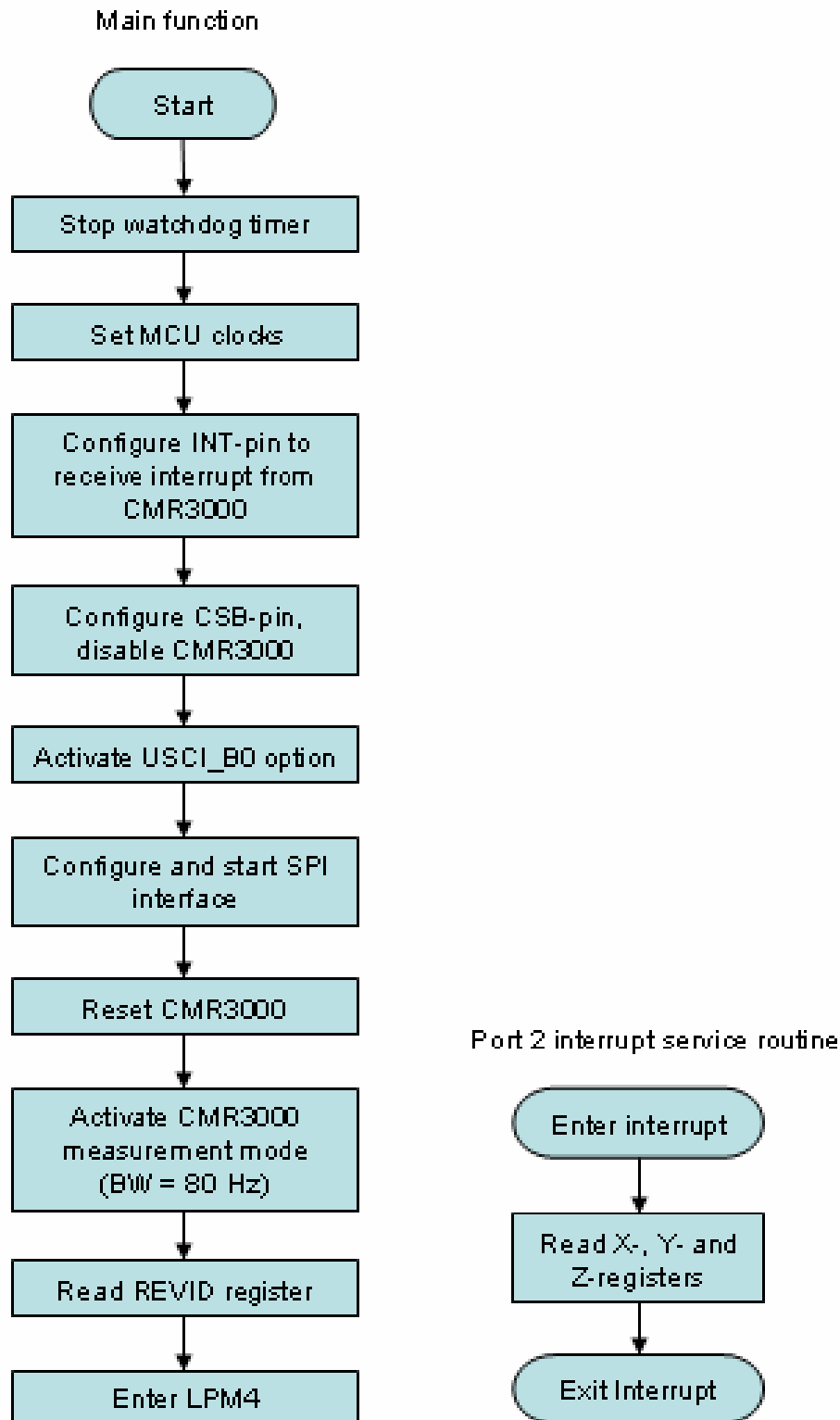


Figure 3. Example Code Flowchart

## C-Code Example, SPI Interface

```
//*****
//  MSP430F2274 Demo - USCI_B0, SPI Interface to CMR3000 Gyro Sensor
//
//  Uses Texas Instruments eZ430-RF2500 Development Tool with VTI Adapter PCB
//  VTI29631A0. Wireless connection not used.
//
//*****

// CONSTANTS
#define XTAL 16000000L
#define TICKSPERMS (XTAL / 1000 / 5 - 1)
#define TICKSPERUS (TICKSPERMS / 1000)
#define SPICLOCK 500 // SPI clock = 500kHz
#define SPIFRAMEDELAY ((1000 / SPICLOCK) * 6) // SPI interframe delay [us] = 6 * Tsck
#define SPICSBDELAY ((1000 / SPICLOCK) / 2) // SCK -> CSB delay [us] = 0.5 * Tsck

// LIBRARIES
#include "msp430x22x4.h"

// PORT DEFINITIONS
#define PORT_INT_IN P2IN
#define PORT_INT_OUT P2OUT
#define PORT_INT_DIR P2DIR
#define PORT_INT_IE P2IE
#define PORT_INT_IES P2IES
#define PORT_INT_IFG P2IFG
#define PORT_INT_VECTOR PORT2_VECTOR

#define PORT_CSB_OUT P4OUT
#define PORT_CSB_DIR P4DIR
#define PORT_SPI_DIR P3DIR
#define PORT_SPI_SEL P3SEL

// REGISTER AND FLAG DEFINITIONS
#define TX_BUFFER UCB0TXBUF
#define RX_BUFFER UCB0RXBUF
#define IRQ_REG IFG2
#define RX_IFG UCB0RXIFG
#define SPI_CTL0 UCB0CTL0
#define SPI_CTL1 UCB0CTL1
#define SPI_BR0 UCB0BR0
#define SPI_BR1 UCB0BR1

// CMR3000-DOX Registers
#define WHO_AM_I 0x00
#define REVID 0x01
#define CTRL 0x02
#define STATUS 0x03
#define X_LSB 0x0C
#define X_MSB 0x0D
#define Y_LSB 0x0E
#define Y_MSB 0x0F
#define Z_LSB 0x10
#define Z_MSB 0x11
#define I2C_ADDR 0x22
```

```
// Control Register setup
#define RESET          0x80          // Set device in reset
#define INT_LEVEL_LOW  0x40          // INT active high
#define I2C_DIS        0x10          // I2C disabled
#define MODE_PD        0x00          // Power Down
#define MODE_STBY      0x02          // Standby mode
#define MODE_20        0x04          // Measurement mode BW=20 Hz
#define MODE_80        0x06          // Measurement mode BW=80 Hz
#define INT_DIS        0x01          // Interrupts disabled

// PIN DEFINITIONS
#define PIN_INT         BIT4
#define PIN_CSB         BIT3
#define PIN_MOSI        BIT1
#define PIN_MISO        BIT2
#define PIN_SCK         BIT3

// FUNCTION PROTOTYPES
unsigned char ReadRegister(unsigned char Address);
unsigned char WriteRegister(unsigned char Address, unsigned char Data);
void ReadAngularRates(unsigned short *Xrate, unsigned short *Yrate, unsigned short *Zrate);
void wait_ms(unsigned short ms);
void wait_us(unsigned short us);

unsigned char Data;
unsigned char Data_MSB;
unsigned char Data_LSB;
unsigned char RevID;
unsigned short Xdata;
unsigned short Ydata;
unsigned short Zdata;

void main(void)
{
    WDTCTL = WDTPW + WDTHOLD;          // Stop watchdog timer

    BCSCTL1 = CALBC1_16MHZ;            // Set DCO to calibrated 16MHz
    DCOCTL = CALDCO_16MHZ;
    BCSCTL2 |= DIVS_3;                 // SMCLK to 2 MHz

    PORT_INT_DIR &= ~PIN_INT;
    PORT_INT_IE |= PIN_INT;            // INT pin interrupt enabled
    PORT_INT_IES = 0x00;               // Generate interrupt on Lo to Hi edge
    PORT_INT_IFG &= ~PIN_INT;          // Clear interrupt flag

    PORT_CSB_DIR |= PIN_CSB;
    PORT_CSB_OUT |= PIN_CSB;           // Unselect gyro sensor

    PORT_SPI_SEL |= PIN_MOSI | PIN_MISO | PIN_SCK; // P3.3, 2,1 USCI_B0 option select
    PORT_SPI_DIR |= BIT0;              // P3.0 output direction

    // Initialize SPI interface to gyro sensor
    SPI_CTL0 |= UCSYNC | UCMST | UCMSB | UCCKPH; // SPI master, 8 data bits, MSB first,
                                                // clock idle low, data output on falling edge
    SPI_CTL1 |= UCSSEL1;               // SMCLK as clock source
    SPI_BR0 = 0x04;                    // Low byte of division factor for baud rate (500kHz)
    SPI_BR1 = 0x00;                    // High byte of division factor for baud rate
    SPI_CTL1 &= ~UCSWRST;              // Start SPI hardware
}
```

```

Data = WriteRegister(CTRL, RESET);           // Reset device
wait_ms(2);                                 // Wait for reset to complete
Data = WriteRegister(CTRL, I2C_DIS | MODE_80); // INT level high, I2C disabled,
                                              // Measurement mode BW=80Hz, Interrupt enabled
RevID = ReadRegister(REVID);                 // Read REVID register

__bis_SR_register(LPM4_bits + GIE);          // Enter LPM4 w/interrupt
}

// Port 2 interrupt service routine, INT pin
#pragma vector=PORT_INT_VECTOR
__interrupt void Port_INT_ISR(void)
{
    if (PORT_INT_IN & PIN_INT)
    {
        ReadAngularRates(&Xdata, &Ydata, &Zdata);
        PORT_INT_IFG &= ~PIN_INT;           // Clear interrupt flag
    }
}

// Read a byte from the gyro sensor
unsigned char ReadRegister(unsigned char Address)
{
    unsigned char Result;

    Address <<= 2;                           // Address to be shifted left by 2 and RW bit to be reset

    PORT_CSB_OUT &= ~PIN_CSB;                // Select gyro sensor

    Result = RX_BUFFER;                       // Read RX buffer just to clear interrupt flag

    TX_BUFFER = Address;                     // Write address to TX buffer

    while (!(IRQ_REG & RX_IFG));               // Wait until new data was written into RX buffer
    Result = RX_BUFFER;                       // Read RX buffer just to clear interrupt flag

    TX_BUFFER = 0;                           // Write dummy data to TX buffer

    while (!(IRQ_REG & RX_IFG));               // Wait until new data was written into RX buffer
    Result = RX_BUFFER;                       // Read RX buffer

    wait_us(SPICSBDELAY);                     // Delay to satisfy 1/2 clk period delay from
                                              // falling edge of clk to CSB
    PORT_CSB_OUT |= PIN_CSB;                 // Deselect gyro sensor

    wait_us(SPIFRAMEDELAY);                   // Delay to satisfy 6x clk period for CS high state

    return Result;                           // Return new data from RX buffer
}

```

```
// Write a byte to the gyro sensor
unsigned char WriteRegister(unsigned char Address, unsigned char Data)
{
    unsigned char Result;

    Address <= 2;                // Address to be shifted left by 2
    Address |= 2;                // RW bit to be set

    PORT_CSB_OUT &= ~PIN_CSB;    // Select gyro sensor

    Result = RX_BUFFER;          // Read RX buffer just to clear interrupt flag

    TX_BUFFER = Address;         // Write address to TX buffer

    while (!(IRQ_REG & RX_IFG)); // Wait until new data was written into RX buffer
    Result = RX_BUFFER;          // Read RX buffer just to clear interrupt flag

    TX_BUFFER = Data;            // Write data to TX buffer

    while (!(IRQ_REG & RX_IFG)); // Wait until new data was written into RX buffer
    Result = RX_BUFFER;          // Read RX buffer

    wait_us(SPICSBDELAY);        // Delay to satisfy 1/2 clk period delay from
                                // falling edge of clk to CSB
    PORT_CSB_OUT |= PIN_CSB;     // Deselect gyro sensor

    wait_us(SPIFRAMEDELAY);      // Delay to satisfy 6x clk period for CS high state

    return Result;
}

// Read all 6 rate registers using the decrement reading feature
// and convert to 16-bit unsigned values
void ReadAngularRates(unsigned short *Xrate, unsigned short *Yrate, unsigned short *Zrate)
{
    unsigned char Result;
    unsigned char Address = Z_MSB;

    Address <= 2;                // Addr shifted left by 2 and RW bit to be reset
    PORT_CSB_OUT &= ~PIN_CSB;    // Select gyro sensor

    Result = RX_BUFFER;          // Read RX buffer just to clear interrupt flag
    TX_BUFFER = Address;         // Write address to TX buffer
    while (!(IRQ_REG & RX_IFG)); // Wait until new data was written into RX buffer
    Result = RX_BUFFER;          // Read RX buffer

    TX_BUFFER = 0;               // Write dummy data to TX buffer
    while (!(IRQ_REG & RX_IFG)); // Wait until new data was written into RX buffer
    Result = RX_BUFFER;          // Read RX buffer
    *Zrate = (unsigned short) ((Result < 8) & 0xFF00); // store Z MSByte

    TX_BUFFER = 0;               // Write dummy data to TX buffer
    while (!(IRQ_REG & RX_IFG)); // Wait until new data was written into RX buffer
    Result = RX_BUFFER;          // Read RX buffer
    *Zrate |= (unsigned short) (Result & 0x00FF);      // store Z LSByte

    TX_BUFFER = 0;               // Write dummy data to TX buffer
    while (!(IRQ_REG & RX_IFG)); // Wait until new data was written into RX buffer
    Result = RX_BUFFER;          // Read RX buffer
    *Yrate = (unsigned short) ((Result < 8) & 0xFF00); // store Y MSByte
}
```

```

TX_BUFFER = 0;
while (!(IRQ_REG & RX_IFG));
Result = RX_BUFFER;
*Yrate |= (unsigned short) (Result & 0x00FF);           // store Y LSByte

TX_BUFFER = 0;
while (!(IRQ_REG & RX_IFG));
Result = RX_BUFFER;
*Xrate = (unsigned short) ((Result << 8) & 0xFF00);    // store X MSByte

TX_BUFFER = 0;
while (!(IRQ_REG & RX_IFG));
Result = RX_BUFFER;
*Xrate |= (unsigned short) (Result & 0x00FF);           // store X LSByte

wait_us(SPICSBDELAY);                                // Delay to satisfy 1/2 clk period delay from
                                                    // falling edge of clk to CSB
PORT_CSB_OUT |= PIN_CSB;                            // Deselect gyro sensor

wait_us(SPIFRAMEDELAY);                             // Delay to satisfy 6x clk period for CSB high state
}

// wait ms
void wait_ms(unsigned short ms)
{
    unsigned short a, b;

    for (a = ms; a > 0; a--)                          // outer loop takes 5 ck per round
        for (b = TICKSPERMS; b > 0; b--)              // inner loop takes 5 ck per round
            asm("nop");
}

// wait us
void wait_us(unsigned short us)
{
    unsigned short a;

    us *= TICKSPERUS;
    for (a = us; a > 0; a--)                          // loop takes 5 ck per round
        asm("nop");
}

```

### 3.2 I2C Interface Example

VTI adapter PCB VT129631 does not support I2C use with CMR3000-D01 so to enable I2C communication with the eZ430-RF2500 the PCB needs to be modified as follows:

- Cut MISO/SDA trace close to J2
- Cut SCK/SCL trace close to J2
- Wire J2-4 to ground
- Wire P2-15 to J2-6
- Add 10 kOhm pull-up resistors from MOSI/SDA and J2-6 to +3.6V

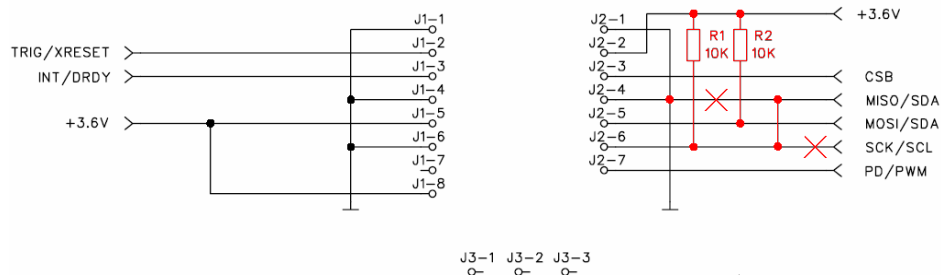


Figure 4. PCB VT129631 modification for I2C use

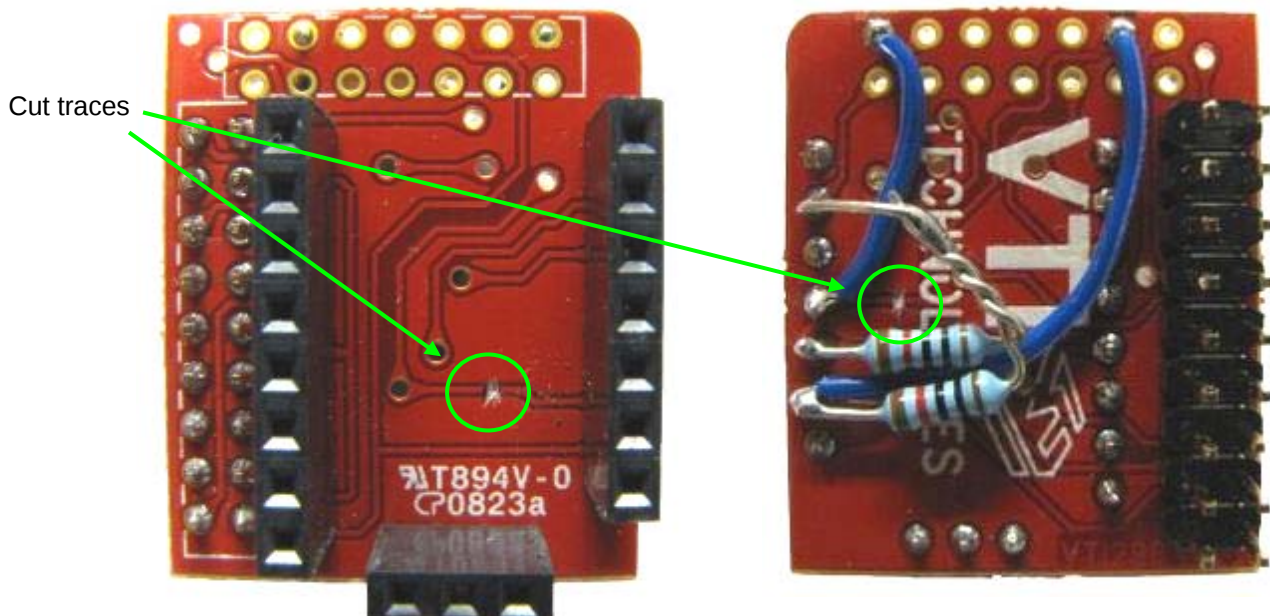


Figure 5. Modified VT129631 PCB

Code flowchart:

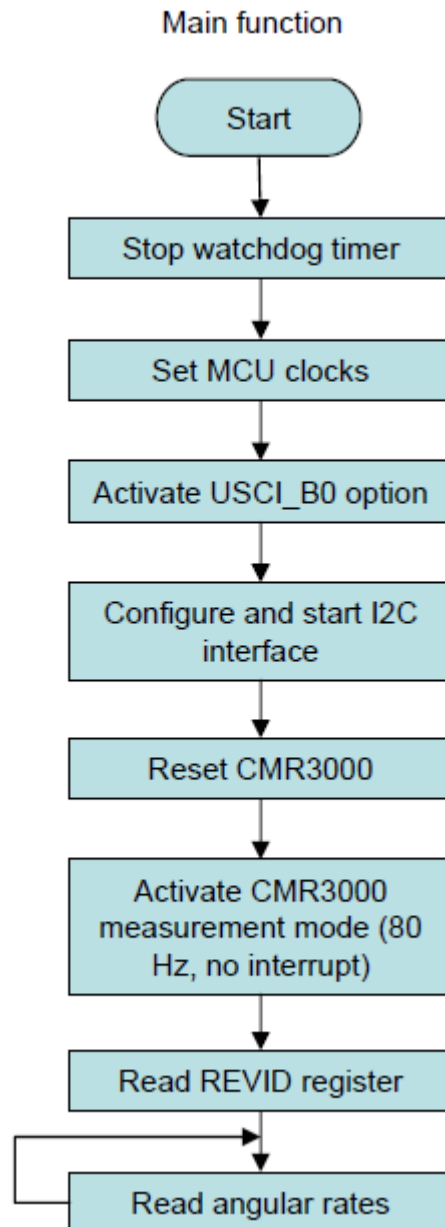


Figure 6. I2C Example Code Flowchart

## C-Code Example, I2C Interface

```
//*****
//  MSP430F2274 Demo - USCI_B0, I2C Interface to CMR3000 Gyro Sensor
//
//  Uses Texas Instruments eZ430-RF2500 Development Tool with VTI Adapter PCB
//  VTI29631A0. Wireless connection not used. Adapter PCB VTI29631 needs to be
//  modified for I2C to work.
//
//*****

// CONSTANTS
#define XTAL          16000000L
#define TICKSPERMS    (XTAL / 1000 / 5 - 1)
#define TICKSPERUS     (TICKSPERMS / 1000)

// LIBRARIES
#include "msp430x22x4.h"

// PORT DEFINITIONS
#define PORT_INT_IN     P2IN
#define PORT_INT_DIR     P2DIR
#define PORT_INT_IE      P2IE
#define PORT_INT_IES     P2IES
#define PORT_INT_IFG     P2IFG
#define PORT_INT_VECTOR PORT2_VECTOR

#define PORT_I2C_DIR     P3DIR
#define PORT_I2C_SEL     P3SEL
#define PORT_I2C_OUT     P3OUT

// REGISTER AND FLAG DEFINITIONS
#define TX_BUFFER        UCB0TXBUF
#define RX_BUFFER        UCB0RXBUF
#define IRQ_REG          IFG2
#define RX_IFG           UCB0RXIFG
#define TX_IFG           UCB0TXIFG

// CMR3000-DOX Registers
#define WHO_AM_I         0x00
#define REVID            0x01
#define CTRL             0x02
#define STATUS           0x03
#define X_LSB            0x0C
#define X_MSB            0x0D
#define Y_LSB            0x0E
#define Y_MSB            0x0F
#define Z_LSB            0x10
#define Z_MSB            0x11
#define I2C_ADDR         0x22

// Control Register setup
#define RESET            0x80                // Set device in reset
#define INT_LEVEL_LOW    0x40                // INT active high
#define I2C_DIS          0x10                // I2C disabled
#define MODE_PD          0x00                // Power Down mode
#define MODE_STBY        0x02                // Standby mode
#define MODE_80          0x04                // Measurement mode BW=80 Hz
#define MODE_20          0x06                // Measurement mode BW=20 Hz
#define INT_DIS          0x01                // Interrupts disabled
```

```
// PIN DEFINITIONS
#define PIN_INT          BIT4
#define PIN_CSB          BIT3
#define PIN_MOSI         BIT1
#define PIN_MISO         BIT2
#define PIN_SCK          BIT3

// FUNCTION PROTOTYPES
unsigned char ReadRegister(unsigned char Address);
unsigned char WriteRegister(unsigned char Address, unsigned char Data);
void ReadAngularRates(unsigned short *Xrate, unsigned short *Yrate, unsigned short *Zrate);
void wait_ms(unsigned short ms);
void wait_us(unsigned short us);

unsigned char Data;
unsigned char Data_MSB;
unsigned char Data_LSB;
unsigned char RevID;
unsigned short Xdata;
unsigned short Ydata;
unsigned short Zdata;

void main(void)
{
    WDTCTL = WDTPW + WDTHOLD;                // Stop watchdog timer

    BCSCTL1 = CALBC1_16MHZ;                  // Set DCO to calibrated 16MHz
    DCOCTL = CALDCO_16MHZ;
    BCSCTL2 |= DIVS_3;                       // SMCLK to 2 MHz

    PORT_INT_DIR &= ~PIN_INT;
    PORT_INT_IE |= PIN_INT;                 // INT pin interrupt enabled
    PORT_INT_IES = 0x00;                   // Generate interrupt on Lo to Hi edge
    PORT_INT_IFG &= ~PIN_INT;              // Clear interrupt flag

    // Initialize I2C interface to gyro sensor
    PORT_I2C_DIR |= BIT0;                  // Set port 3 pin 0 as output and set high.
    PORT_I2C_OUT |= BIT0;

    UCBOCTL1 |= UCSWRST;                   // Put state machine in reset
    UCBOCTL1 |= UCSSEL_2;                  // SMCLK as clock source
    PORT_I2C_SEL |= BIT2 | BIT1;          // Set I/O pins 1 and 2 for port 3 to be for the
    // I2C peripheral
    UCBOCTL0 |= UCMST + UCSYNC + UCMODE_3; // I2C mode, master, synchronous
    UCBOBR0 = 5;                          // 2MHz/5 = 400kHz
    UCBOBR1 = 0;
    UCBOI2CSA = 0x1E;                     // Slave Address is 1Eh
    UCBOCTL1 &= ~UCSWRST;                 // Start I2C state machine

    Data = WriteRegister(CTRL, RESET);     // Reset device
    wait_ms(2);                           // Wait for reset to complete
    Data = WriteRegister(CTRL, MODE_80 | INT_DIS); // INT level high, I2C enabled,
    // Measurement mode BW=80Hz, Interrupt disabled
    RevID = ReadRegister(REVID);           // Read REVID register

    __bis_SR_register(LPM4_bits + GIE);    // Enter LPM4 w/interrupt
}
```

```

// Port 2 interrupt service routine, INT pin
#pragma vector=PORT_INT_VECTOR
__interrupt void Port_INT_ISR(void)
{
    if (PORT_INT_IN & PIN_INT)
    {
        ReadAngularRates(&Xdata, &Ydata, &Zdata);           // Read angular rates
        PORT_INT_IFG &= ~PIN_INT;                           // Clear interrupt flag
    }
}

// Read a byte from the gyro sensor
unsigned char ReadRegister(unsigned char Address)
{
    unsigned char Result;

    // Read data from I2C slave device at the DeviceAddress specified.

    UCB0CTL1 |= UCTR + UCTXSTT;                               // I2C TX, start condition

    while (!(IRQ_REG & TX_IFG));                               // Wait for slave address transmit to complete

    TX_BUFFER = Address;                                       // Load TX buffer with register address

    while (!(IRQ_REG & TX_IFG));                               // Wait for transmit to complete

    IRQ_REG &= ~TX_IFG;                                         // Clear USCI_B0 TX int flag

    // Send restart with transmit/receive bit NOT set
    UCB0CTL1 &= ~UCTR;                                         // Toggle transmitter bit
    UCB0CTL1 |= UCTXSTT;                                       // I2C start condition

    while(UCB0CTL1 & UCTXSTT);                                  // Wait for start to complete
    UCB0CTL1 |= UCTXSTP;                                       // I2C stop condition

    while (!(IRQ_REG & RX_IFG));                               // Wait for receive buffer to fill

    Result = RX_BUFFER;                                         // Fill receive data buffer with received byte

    while (UCB0CTL1 & UCTXSTP);                                 // Wait for stop condition to complete

    IRQ_REG &= ~RX_IFG;                                         // Clear USCI_B0 RX int flag

    return Result;                                             // Return new data from RX buffer
}

// Write a byte to the gyro sensor
unsigned char WriteRegister(unsigned char Address, unsigned char Data)
{
    // Write data to I2C slave device at the DeviceAddress specified.

    UCB0CTL1 |= UCTR + UCTXSTT;                               // I2C TX, start condition

    while (!(IRQ_REG & TX_IFG));                               // Wait for slave address transmit to complete

    TX_BUFFER = Address;                                       // Load TX buffer with register address

```

```

while (!(IRQ_REG & TX_IFG));           // Wait for transmit to complete

TX_BUFFER = Data;                     // Load TX buffer with data byte

while (!(IRQ_REG & TX_IFG));           // Wait for transmit to complete
UCB0CTL1 |= UCTXSTP;                   // I2C stop condition

while (UCB0CTL1 & UCTXSTP);            // Wait for stop condition to complete

IRQ_REG &= ~TX_IFG;                   // Clear USCI_B0 TX int flag

return 0;
}

// Read all 6 rate registers using the decrement reading feature
// and convert to 16-bit unsigned values
void ReadAngularRates(unsigned short *Xrate, unsigned short *Yrate, unsigned short *Zrate)
{
    unsigned char Result;
    unsigned char Address = Z_MSB;

    // Read data from I2C slave device at the DeviceAddress specified.

    UCB0CTL1 |= UCTR + UCTXSTT;         // I2C TX, start condition

    while (!(IRQ_REG & TX_IFG));         // Wait for slave address transmit to complete

    TX_BUFFER = Address;                 // Load TX buffer with register address

    while (!(IRQ_REG & TX_IFG));         // Wait for transmit to complete

    IRQ_REG &= ~TX_IFG;                 // Clear USCI_B0 TX int flag

    // Send restart with transmit/receive bit set
    UCB0CTL1 &= ~UCTR;                  // Toggle transmitter bit
    UCB0CTL1 |= UCTXSTT;                 // I2C start condition

    while(UCB0CTL1 & UCTXSTT);           // Wait for start to complete

    while (!(IRQ_REG & RX_IFG));         // Wait for receive buffer to fill

    Result = RX_BUFFER;                  // Fill receive data buffer with received byte
    *Zrate = (unsigned short) ((Result << 8) & 0xFF00); // store Z MSByte

    while (!(IRQ_REG & RX_IFG));         // Wait for receive buffer to fill

    Result = RX_BUFFER;                  // Fill receive data buffer with received byte
    *Zrate |= (unsigned short) (Result & 0x00FF); // store Z LSByte

    while (!(IRQ_REG & RX_IFG));         // Wait for receive buffer to fill

    Result = RX_BUFFER;                  // Fill receive data buffer with received byte
    *Yrate = (unsigned short) ((Result << 8) & 0xFF00); // store Y MSByte

    while (!(IRQ_REG & RX_IFG));         // Wait for receive buffer to fill

    Result = RX_BUFFER;                  // Fill receive data buffer with received byte
    *Yrate |= (unsigned short) (Result & 0x00FF); // store Y LSByte

```

```

while (!(IRQ_REG & RX_IFG));           // Wait for receive buffer to fill

Result = RX_BUFFER;                   // Fill receive data buffer with received byte
*Xrate = (unsigned short) ((Result << 8) & 0xFF00); // store X MSByte

UCB0CTL1 |= UCTXSTP;                  // I2C stop condition

while (!(IRQ_REG & RX_IFG));           // Wait for receive buffer to fill

Result = RX_BUFFER;                   // Fill receive data buffer with received byte
*Xrate |= (unsigned short) (Result & 0x00FF);      // store X LSByte

while (UCB0CTL1 & UCTXSTP);           // Wait for stop condition to complete

IRQ_REG &= ~RX_IFG;                   // Clear USCI_B0 RX int flag
}

// wait ms
void wait_ms(unsigned short ms)
{
    unsigned short a, b;

    for (a = ms; a > 0; a--)           // outer loop takes 5 ck per round
        for (b = TICKSPERMS; b > 0; b--) // inner loop takes 5 ck per round
            asm("nop");
}

// wait us
void wait_us(unsigned short us)
{
    unsigned short a;

    us *= TICKSPERUS;
    for (a = us; a > 0; a--)           // loop takes 5 ck per round
        asm("nop");
}

```

## 4 RESULT WAVEFORMS

There is no display on the development hardware so a logic analyzer was used to verify the results.

Figure 7 (SPI) and Figure 9 (I2C) show how the register reading is triggered from the CMR3000-D01 INT-pin. After the interrupt has activated, the output registers are read out in sequence using multiple read operation mode (decrement reading).

In Figure 8 (SPI) and Figure 10 (I2C) it can be seen how the interrupt takes place immediately after CMR3000-D01 has new data available for reading. It also shows how the INT pin is automatically cleared by reading the rate output data.

Note that in case of I2C the Fast Mode (400 kbit/s) has been used. With Standard Mode (100 kbit/s) the read frame takes longer than interval of the CMR3000-D01 data ready interrupts (500 us, 2 kHz) so it will not be possible to read every sample and the software must take care that two read operations are not attempted concurrently. Use of separate interrupt (timer) is recommended in this case.

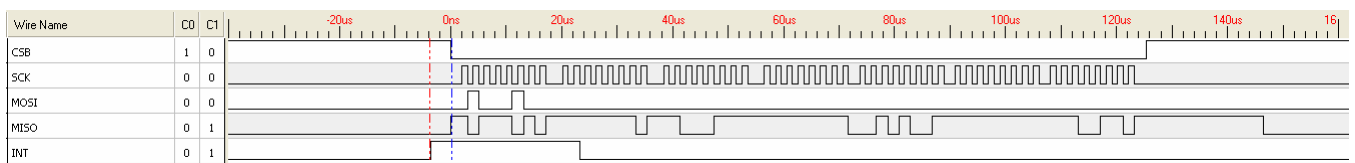


Figure 7. SPI waveforms when reading X-, Z- and Y-registers (decrement reading)

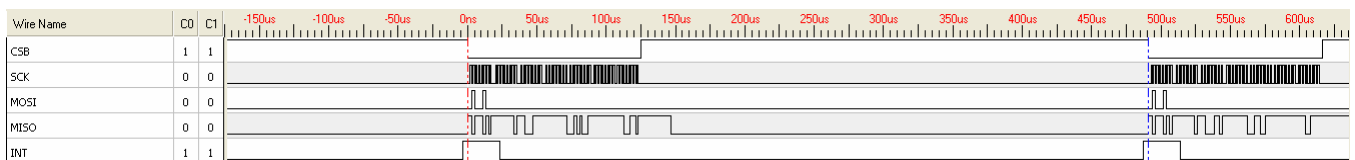


Figure 8. Register reading is triggered by CMR3000-D01's INT signal, SPI bus

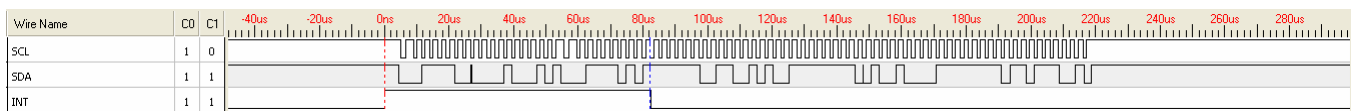


Figure 9. I2C waveforms when reading X-, Z- and Y-registers

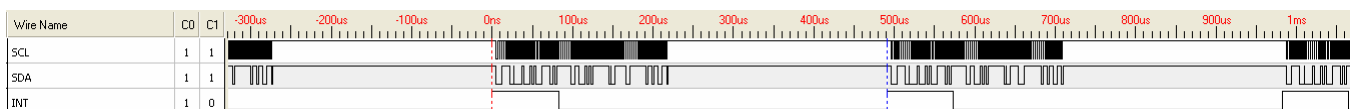


Figure 10. Register reading is triggered by CMR3000-D01's INT signal, I2C bus