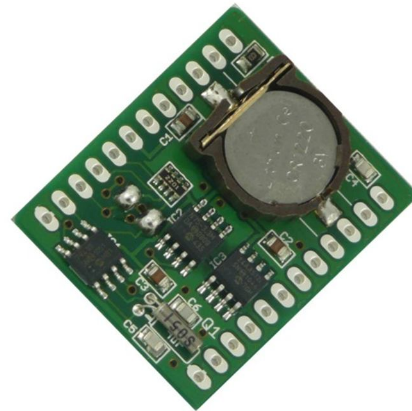
TDS1I2C1

Features

- Mini I²C Board
- Microchip MCP79410 Real Time Clock with additional 64 bytes of SRAM and 1Kbit EEPROM.
- Battery Backup of MCP79410 RTCC and SRAM with included CR1220 lithium cell.
- 24AA512 EEPROM memory with 64Kbytes of non-volatile storage.
- TCN75 temperature sensor.
- Wide operating voltage 2.7v - 5.5v. Temperature sensor optimised for 3.3v.
- Small 24 pin IC compatible package 0.1"x0.9" pitch.
- Selectable on-board I2C 2K2 pull-ups.
- RTCC and Temperature sensor interrupt outputs with resistor pull-ups.
- Overall size 28mm(w) x 34mm(l).
- MPASM and MC18 USB CDC Serial Demo code provided showing full use of the I2c peripherals.
- Pin Compatible with TDSDB146J50.



Description

The Tiertex I2C Add-on board is a combination of 3 useful I2c devices in a compact package. The board incorporates battery backed real time clock, large EEPROM non-volatile memory and temperature sensor.

Contents

1.0 Pin Diagram 3
1.1 Electrical Characteristics 3
1.2 I²C Address Table 3
2.0 On-Board Features 4
2.1 MCP79410 Real Time Clock 4
2.2 TCN75 MOA Temperature Sensor 5
2.3 24AA512 512Kbit EEPROM 5
2.4 I²C Pull-up resistors 5
3.0 Data Sheet References 6
4.0 Schematic 7
5.0 Example PIC18F C software 7

1.0 Pin Diagram

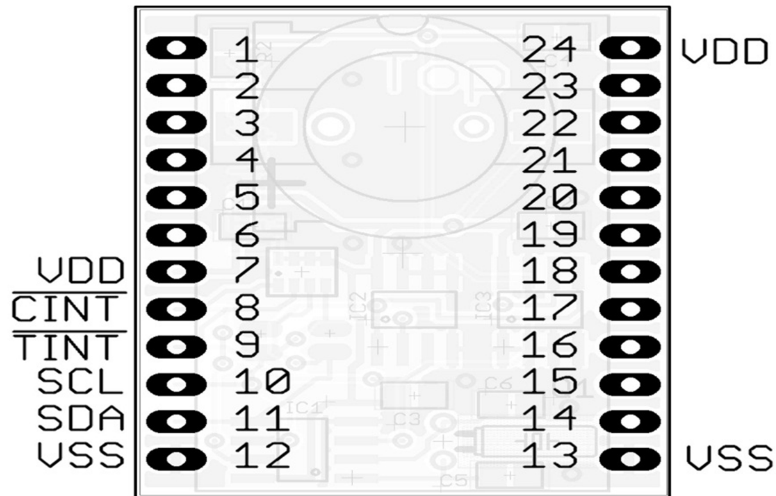


Figure 1

Dimensions 28mm (w) x 34mm (h).

1.1 Electrical Characteristics

Operating conditions at 25°C

Parameter	Min	Typ.	Max	Units
Supply voltage	2.7	3.3	5.5	V
Current Requirement			6.0	ma

Figure 2 Electrical Characteristics

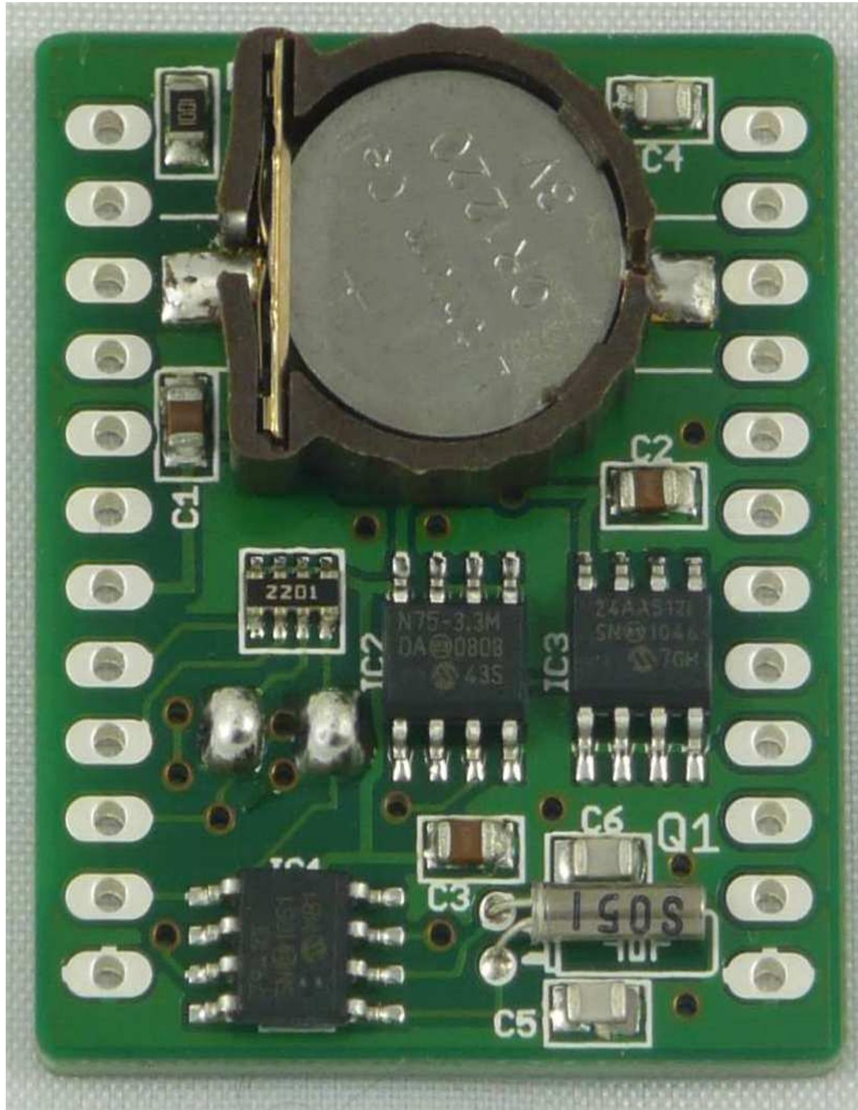
1.2 I²C Address Table

Device	Description	Read Address	Write Address
MCP79410 RTCC	Real Time Clock	0xdf	0xde
TCN75	Temperature Sensor	0x91	0x90
24AA512	512Kbit EEPROM	0Xa1	0xa0

Figure 3

Additional TCN75s and 24AA512s can be added to the I²C bus as long as their chip address a0-a2 are not set to 0.

2.0 On-Board Features



2.1 MCP79410 Real Time Clock

Microchip's MCP79410 RTCC implementation includes the following features

- Full on chip RTCC with on board battery backup.
- Dual Alarm with interrupt out.
- Shut down and start up time stamps.
- 64 bytes of battery backed SRAM
- 1024 Kbits of non-volatile EEPROM
- Programmable square wave output

- Digital calibration adjustment

The battery backup utilises a CR1220 lithium coin cell (included).

Pin 8, CINT is connected to the MCP79410 multifunction output pin and has a 2K2Ω pull-up resistor.

2.2 TCN75 MOA Temperature Sensor

Microchip's TCN75 temperature sensor has the following features

- -55°C to 125°C range to 0.5°C resolution.
- Maximum 8 addressable devices on same bus.
- Programmable output pin, temperature alarm interrupt, comparator/thermostat output.
- Programmable trip points with hysteresis.
- Low power. 250µa typical.

Pin 9, TINT is connected to the TCN75 multifunction output pin and has a 2K2Ω pull-up resistor.

2.3 24AA512 512Kbit EEPROM

- The on-board 24AA512 512Kbit EEPROM has 64Kbytes of non-volatile memory.
- Low power, 400µa typical.
- 128 byte write buffer.
- Fully automatic erase and write cycle.
- Maximum 8 addressable devices on same bus.
- 1 million write/erase cycles, >200 year data retention life.

2.4 I²C Pull-up resistors

The board has 2K2Ω pull-up resistors on both the SCL and SDA open collector I²2 lines, these values are required for 400 KHz I²C communication at 3.3v. If you do not require pull-up resistors then there are some de-solderable links on the board to disconnect this feature.

3.0 Data Sheet References

Full data sheets at www.microchip.com

MCP79410 Real Time Clock

http://ww1.microchip.com/downloads/en/DeviceDoc/22266A.pdf

TCN75 MOA Temperature Sensor

http://ww1.microchip.com/downloads/en/DeviceDoc/21935D.pdf

24AA512 512Kbit EEPROM

http://ww1.microchip.com/downloads/en/DeviceDoc/21754M.pdf

4.0 Schematic

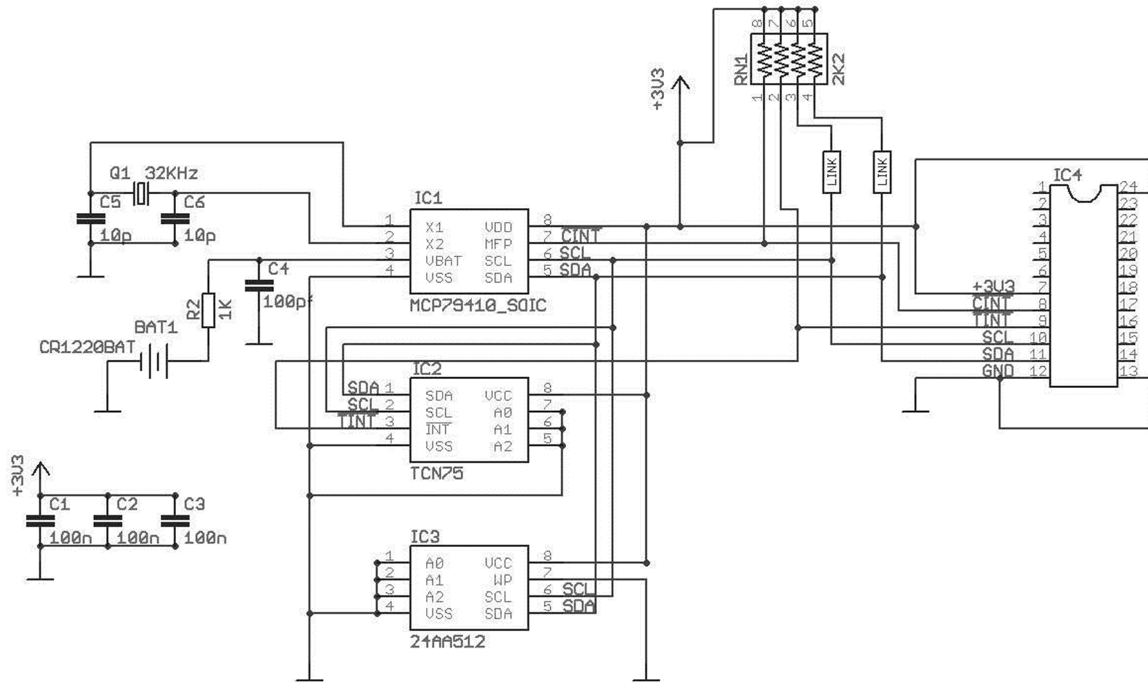


Figure 4

5.0 Example PIC18F C software

```
//  Example Code for a I2c implementation on Microchip PIC18f46J50
//  You are free to use as you wish, but it is supplied 'as is'
//  Tiertex Design Studios shall not be liable in any circumstances for any damges arising
//  when using this code.

// The code below uses simple software polling techniques for I2C communication. For more efficient
// code use interrupt polling instead.
// This code is not intended to compile as is but is a collection of routines taken from the USB
// CDCSerial Demo for the I2C add-card based on the Tiertex PIC18F46J50 demo board.
// The full project can be downloaded from wwww.tiertex.com

#define SCLK1           PORTBbits.SCL1
#define SDAT1           PORTBbits.SDA1
char    RTCCDATA[32];                   // 32 byte copy of the rtcc registers
char    RTCCSRAM[64];                   // 64 byte copy of rtcc sram
char  t7;


// Set up PIC's I2C module
//
void InitI2C()
{
        SCLK1 = 0;
        SDAT1 = 0;
```

```
        SSPSTAT = 0;            // disable SMBUS
        SSPSTATbits.SMP = 1;    // disable slew rate
        SSPADD = 0x18*4;        // 100kz I2C comms.
        SSPCON1bits.SSPM3 = 1;  // I2C master mode in hardware
        SSPCON1bits.SSPEN = 1;  // enable SSP module
        SSPCON2 = 0;            // clrear MSSP
        PIR1bits.SSPIF = 0;     // Clear SSP interrupt flag
        PIR2bits.BCLIF = 0;     // Clear Bit Collision flag
}


// Send a START command
//
void Start()
{
        PIR1bits.SSPIF = 0;              // Clear SSP interrupt flag
        SSPCON2bits.SEN = 1;             // Generate Start condition
        while ( PIR1bits.SSPIF == 0 ) {}
}

// A RESTART for multiple I2c commands
//
void Restart()
{
        PIR1bits.SSPIF = 0;              // Clear SSP interrupt flag
        SSPCON2bits.RSEN = 1;            // Generate Restart condition
        while ( PIR1bits.SSPIF == 0 ) {}
}

// STOP I2c command
//
void Stop()
{
        PIR1bits.SSPIF = 0;              // Clear SSP interrupt flag
        SSPCON2bits.PEN = 1;             // Generate Stop condition
        while ( PIR1bits.SSPIF == 0 ) {}
}

// Send a byte on the I2C bus
//
void TXbyte(BYTE a)
{
        PIR1bits.SSPIF = 0;              // Clear SSP interrupt flag
        SSPBUF = a;                      // Write byte out to device
        while ( PIR1bits.SSPIF == 0 ) {}
}

// Read byte from the I2C bus
//
BYTE RXbyte()
{
        SSPCON2bits.ACKDT = 0;           // No Ack as this is one of many reads
        PIR1bits.SSPIF = 0;              // Clear SSP interrupt flag
        SSPCON2bits.RCEN = 1;            // Initiate reception of byte
        while ( PIR1bits.SSPIF == 0 ) {} // wait till reception

        PIR1bits.SSPIF = 0;              // Clear SSP interrupt flag
        SSPCON2bits.ACKEN = 1;           // Generate ACK/NO ACK bit
        while ( PIR1bits.SSPIF == 0 ) {} // wait till complete
        return SSPBUF;
}

// Read the last byte of a read sequence from the I2C bus, the ack lets the I2c slave that this is the
last requested byte
//
BYTE RXbyteACK()
{
        SSPCON2bits.ACKDT = 1;           // Ack as this is the last read
        PIR1bits.SSPIF = 0;              // Clear SSP interrupt flag
        SSPCON2bits.RCEN = 1;            // Initiate reception of byte
```

```
        while ( PIR1bits.SSPIF == 0 ) {} // wait till reception
        PIR1bits.SSPIF = 0;              // Clear SSP interrupt flag
        SSPCON2bits.ACKEN = 1;           // Generate ACK/NO ACK bit
        while ( PIR1bits.SSPIF == 0 ) {} // wait till complete
        return SSPBUF;
}



void ReadI2Crtcc()
{
                Start();                 // start I2C exchange
                TXbyte(0xde);            // send write address to RTCC
                TXbyte(0x00);            // set ptr to 0
                Restart();
                TXbyte(0xdf);            // send read address to RTCC
 // I'm reading only 9 bytes (upto and including trim, there is 31 register in total if required)
                for ( t7=0;t7<8;t7++) RTCCDATA[t7] = RXbyte();
                RTCCDATA[8] = RXbyteACK();      // last byte, NO 9 use ack
                Restart();
                TXbyte(0xde);            // send write address to RTCC
                TXbyte(0x20);            // set ptr to 20, start of sram
                Restart();
                TXbyte(0xdf);            // send read address to RTCC
// I'm reading all 64 bytes of sram
                for ( t7=0;t7<63;t7++) RTCCSRAM[t7] = RXbyte();
                RTCCSRAM[63] = RXbyteACK();
                Stop();                  // Generate stop bit
}

void WriteI2Crtcc()
{
                Start();                 // start I2C exchange
                TXbyte(0xde);            // send write address to RTCC
                TXbyte(0x00);            // set ptr to 0
                for ( t7=0;t7<9;t7++)
                        TXbyte(RTCCDATA[t7]);   // write out registers 0 - 8
                Stop();                         // Generate stop bit
}

// A wake up routine for the RTCC to ensure all the correct register are on.
//
//
void RTCCSwitchon()
{
                Start();
// setup oscilator on
                TXbyte(0xde);            // send write address to RTCC
                TXbyte(0x00);            // set ptr to register 0
                Restart();
                TXbyte(0xdf);            // send Read address to RTCC
                RTCCDATA[0] = RXbyteACK();      // read seconds  + oscillator flag
                Restart();
                TXbyte(0xde);            // send write address to RTCC
                TXbyte(0x00);                   // set ptr to register 0
                TXbyte(RTCCDATA[0]|0x80);       // send back seconds with Osc flag set
                Restart();
// now ensure the battery backup is enabled
                TXbyte(0xde);            // send write address to RTCC
                TXbyte(0x03);                   // set ptr to register 3
                Restart();
                TXbyte(0xdf);            // send Read address to RTCC
                RTCCDATA[3] = RXbyteACK();      // read seconds  + oscillator flag
                Restart();
                TXbyte(0xde);            // send write address to RTCC
                TXbyte(0x03);                           // set ptr to register 3
                TXbyte(RTCCDATA[3]|0x08);       // send back seconds with Osc flag set
                Restart();
```

```
// now setup general flags
                TXbyte(0xde);           // send write address to RTCC
                TXbyte(0x07);           // set ptr to 7, flags register
                TXbyte(0x40);           // sqw output at 1hz.
                Stop();                             // Generate stop bit
}

//      read the tnc75 temperature sensor
//
void ReadTemperature()
{
                Start();                                // assert Start condition
                TXbyte(0x90);                           // send 0x90 write address
                TXbyte(0x00);                           // set read ptr to 0
                Restart();                      // Generate another start bit
                TXbyte(0x91);                           // 0x91 read address

                temperature=0;
                temperature =  (WORD)RXbyte()<<8;       // read high byte
                temperature |= (WORD)RXbyteACK();       // read low byte
                Stop();                                 // Generate stop bit
                temperature >>= 7;                      // shift down 7 bits for temp is now 6:1
}

// EEPROM read.  The EEPROM write is more complicated as you will need to let the
// device finish its erase/write cycle. This is done by checking for an acknowledge
// from the EEPROM, see section 7.0 from the Microchip data sheet.
//
BYTE ReadI2cEEbyte(WORD a)
{
                Start();
                TXbyte(0xa0);                           // 24lc512 write address
                TXbyte((BYTE)(w1>>8));          // set address high
                TXbyte((BYTE)(w1&0xff)); // set address low
                Restart();
                TXbyte(0xa1);                           // 24lc512 read address
                t1 = RXbyteACK();
                Stop();
                return t1;
}
```