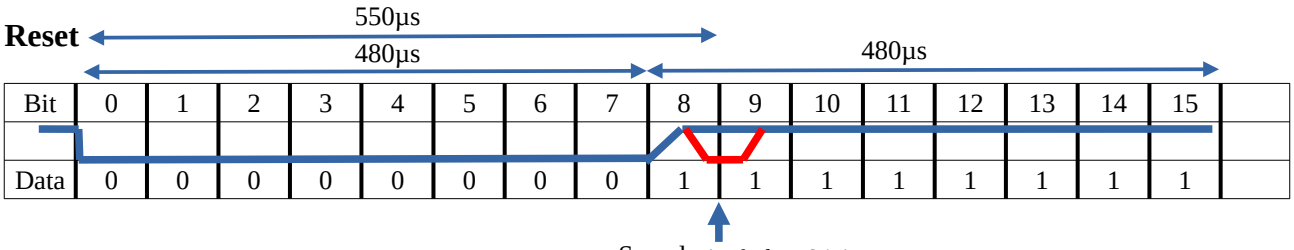
Nutzung der SPI-Hardware im PIC24 zur Steuerung des 1-Wire-Bus (Temperatursensoren).

Zur Steuerung wird die Hardwareeinheit SPI des PIC24 eingesetzt, diese formt das Signal und Timing autark mit der Hardware, nach jedem Symbol-Rahmen wird ein Interrupt (Rx) ausgelöst und die weitere Abfolge per Software gesteuert.
Vorteil, Zeitkritische Anteile des Signals werden per Hardware erzeugt, verbraucht auch keine CPU-Rechenleistung. Wenn dann nicht die zeitkritische Aneinanderreihung per Software gesteuert wird, können andere zeitkritische Aufgaben der CPU in höher priorisierten Interrupts abgearbeitet werden.

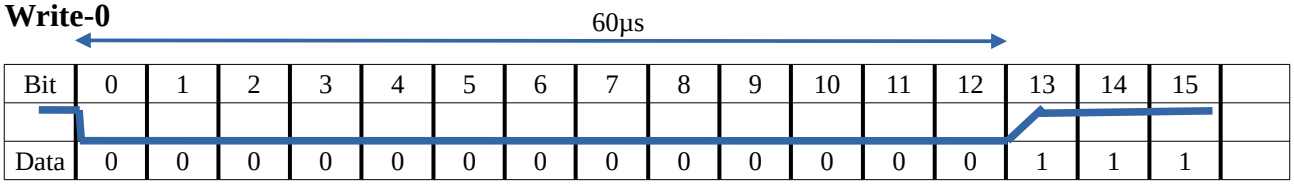
SPI-Config:
Pin Config: SDOx und SDIx an selben RPx-Pin; OpenDrain = 1; (extern Pull-Up an Pin; 4,7k); SCKx und CS not used;
Mode: 16-Bit; SDOx: CKE = 0; CKP = 0; Input-Sample SDIx: SMP = 0; (Wichtig, da sonst der IRQ SRMT zu früh ausgelöst wird!)
Baudrate-Generator (**BRG**) = 548 für Reset-Signal; 36 für DataBits-Signal (Basis 16 MHz CPU-Clock)
SPI-Send-Data für Reset-Signal: `0b0000000011111111` = 0x00FF (16-Bit)
SPI-Send-Data für Write-**0**-Signal: `0b0000000000000111` = 0x0003 (16-Bit)
SPI-Send-Data für Write-**1**-Signal: `0b0011111111111111` = 0x3FFF (16-Bit)
Zum Lesen vom 1-Wire-Bus wird auch ein Write-1-Signal gesendet und dann das gesampelte Bit-12 ausgewertet.
Der Präsens-Impuls wird in Bit 6 und 7 (mind. 1 der beiden auf Low) gelesen.

Reset

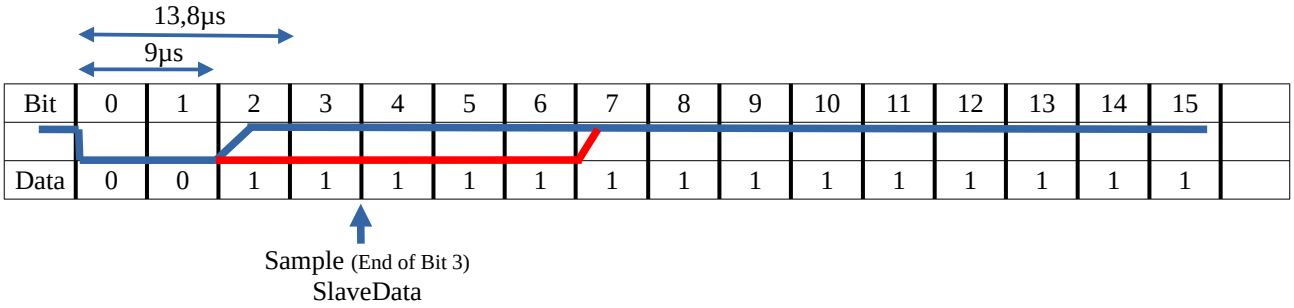


BRG: 548	**BRG**: 549 (Alternative zum Testen)
1 Bit ≙ 68,625 µs	1 Bit ≙ 68,75 µs
7 Bit ≙ 480,375 µs	7 Bit ≙ 481,25 µs
8 Bit ≙ 549 µs	8 Bit ≙ 550 µs

Write-0



Write-1 / Read-x



BRG: 36	**BRG**: 37 (Alternative zum Testen)
1 Bit ≙ 4,625 µs	1 Bit ≙ 4,75 µs
2 Bit ≙ 9,25 µs	2 Bit ≙ 9,5 µs
3 Bit ≙ 13,875 µs	3 Bit ≙ 14,25 µs
13 Bit ≙ 60,125 µs	13 Bit ≙ 61,75 µs

Master (vom SPI-Modul generiert):
Slave (vom Slave-Device / Temperatursensor generiert):

Code for usage with DS18B20 Temperaturesensor:

```
#define C_SPI_1W_Reset_BRG      548 // 7 LowBits = 480us; => 14583Hz => 16MHz / 14583Hz = 1097; Div2 -1 => 548
#define C_SPI_1W_Data_BRG       36 // 13 LowBits = 60us; => 216666Hz => 16MHz / 216666Hz = 74; Div2 -1 => 36
#define C_SPI_1W_Reset          0b1000000011111111
#define C_SPI_1W_Write_0        0b1000000000000011
#define C_SPI_1W_Write_1        0b1001111111111111


    typedef union __attribute__((packed))
    {
        uint8_t Val;

        struct __attribute__((packed))
        {
            uint8_t b0 : 1;
            uint8_t b1 : 1;
            uint8_t b2 : 1;
            uint8_t b3 : 1;
            uint8_t b4 : 1;
            uint8_t b5 : 1;
            uint8_t b6 : 1;
            uint8_t b7 : 1;
        }
        bits;
    }
    UINT8_VAL;

    typedef union __attribute__((packed))
    {
        uint16_t Val;
        uint8_t v[2];

        struct __attribute__((packed))
        {
            uint8_t LB;
            uint8_t HB;
        }
        byte;

        struct __attribute__((packed))
        {
            uint8_t b0 : 1;
            uint8_t b1 : 1;
            uint8_t b2 : 1;
            uint8_t b3 : 1;
            uint8_t b4 : 1;
            uint8_t b5 : 1;
            uint8_t b6 : 1;
            uint8_t b7 : 1;
            uint8_t b8 : 1;
            uint8_t b9 : 1;
            uint8_t b10 : 1;
            uint8_t b11 : 1;
            uint8_t b12 : 1;
            uint8_t b13 : 1;
            uint8_t b14 : 1;
            uint8_t b15 : 1;
        }
        bits;
    }
    UINT16_VAL;



    typedef union __attribute__((packed))
    {

        struct
        {
            uint8_t Val[11];
            uint8_t BufferLen;
            uint8_t BufferIdx;
            uint8_t BitCount;
            UINT8_VAL TxByte;
            UINT8_VAL RxByte;
            uint8_t OW_Reset;
            uint8_t PresencePulse;
            uint8_t Result; // 0:Runing; 1:Ready; 0xff:Error;
        };

        struct
        {
            uint8_t ROM_CMD;
            uint8_t Function_CMD;
            uint8_t Temp_LSB;
            uint8_t Temp_MSB;
            uint8_t Temp_H;
            uint8_t Temp_L;
            uint8_t Config;
            uint8_t Res[3];
            uint8_t CRC;
        } CC_Read;

        struct
        {
            uint8_t ROM_CMD;
            uint8_t Function_CMD;
            uint8_t Temp_H;
            uint8_t Temp_L;
            uint8_t Config;
        } CC_Write;

    }
    t_1WireBuffer;
```

Hardware-Setup:

```
    ODCxbits.ODCxy = 1;
    LATxbits.LATxy = 1;
    TRISxbits.TRISxy = 0;
    _RPx = _RPOUT_SDO1; // SDO1 and SDI1 on same Pin.
    _SDI1R = x; // SDO1 and SDI1 on same Pin.

   // SPI Config; used for 1-Wire-Bus
    SPI1CON1 = 0;
    SPI1CON1Lbits.ENHBUF = 0; // Enhanced Buffer mode is disabled
    SPI1CON1Lbits.MCLKEN = 0; // Peripheral clock is used by the BRG
    SPI1CON1Lbits.DISSCK = 1; // SCKx pin is not used by the module; pin is controlled by the port function
    SPI1CON1Lbits.MSTEN = 1; // Host mode
    SPI1CON1Lbits.SSEN = 0; // SSx pin is not used by the macro (SSx pin will be controlled by the port I/O)
    SPI1CON1Lbits.CKE = 0; // Transmit happens on transition from Idle clock state to active clock state
    SPI1CON1Lbits.SMP = 0; // Input data are sampled at the middle of data output time
    SPI1CON1Lbits.MODE = 0b01; // 16-Bit;
    SPI1CON1H = 0;
    SPI1CON1Hbits.FRMEN = 0; //Framed SPI support is disabled
    SPI1CON1Hbits.IGNTUR = 1; // A Transmit Underrun (TUR) is NOT a critical error and data indicated by URDTEN are transmitted until
the SPI1TXB is not empty
    SPI1CON1Hbits.IGNROV = 1; // A Receive Overflow (ROV) is NOT a critical error; during ROV, data in the FIFO are not overwritten
by the receive data
    SPI1CON1Hbits.AUDEN = 0; // Audio protocol is disabled
    SPI1CON2L = 0; // See MODE[32,16] bits in SPI1CON1L[11:10]
    SPI1STATL = 0;
    SPI1STATH = 0;
    SPI1BRGL = C_SPI_1W_Reset_BRG;
    SPI1IMSKL = 0;
    SPI1IMSKLbits.SRMTEN = 1; // Shift Register Empty (SRMT) generates interrupt events
    SPI1IMSKLbits.SPIRBFEN = 1; // SPI receive buffer full generates an interrupt event
    SPI1IMSKH = 0;
    SPI1URDTL = 0;
    SPI1URDTH = 0;
//    SPI1CON1Lbits.SPIEN = 1;
    _SPI1IP = 1; // Priority 1
    _SPI1IF = 0;
    _SPI1IE = 1;
    _SPI1RXIP = 7; // Priority 7, Achtung hier hohe Priorität, es wird nur 1 ConfigBit gesetzt und Irq-Flag gelöscht. Siehe Irq-
Function
    _SPI1RXIF = 0;
    _SPI1RXIE = 1;
```

Helper Function:

```
//CRC

int calc_crc(int data_byte, int crc)
{
    int bit_mask = 0, carry_check = 0, temp_data = 0;
    temp_data = data_byte;
    for (bit_mask = 0; bit_mask <= 7; bit_mask++)
    {
        data_byte = data_byte ^ crc;
        crc = crc / 2;
        temp_data = temp_data / 2;
        carry_check = data_byte & 0x01;
        if (carry_check)
        {
            crc = crc ^ 0x8C;
        }
        data_byte = temp_data;
    }
    return ( crc);
}

//============================================================
//calculates checksum for n bytes of data
//and compares it with expected checksum
//input:  data[] checksum is built based on this data
//
// nbrOfBytes checksum is built for n bytes of data checksum
//return: error:
//  expected checksum CHECKSUM_ERROR = checksum does not match 0 = checksum matches
//============================================================
uint8_t CheckCrc(uint8_t data[], uint8_t nbrOfBytes, uint8_t checksum)
{
    uint8_t crc = 0, i;
    for (i = 0; i < nbrOfBytes; i++)
    {
        crc = calc_crc(data[i], crc);
    }
    if (crc != checksum)
    {
        return 1;
    }
    else
    {
        return 0;
    }
}
```

Interrupt-Function:

```
volatile t_1WireBuffer Ch1_1WireBuffer;

void __attribute__((__interrupt__, no_auto_psv)) _SPI1RXInterrupt(void)
{
    SPI1CON1Lbits.DISSDO = 1;
    _SPI1RXIF = 0;
}

void __attribute__((__interrupt__, no_auto_psv)) _SPI1Interrupt(void)

{
    _SPI1IF = 0;

    t_UINT16_VAL RxBuff = {.Val = SPI1BUFL};
    SPI1STATL = 0;
    if (Ch1_1WireBuffer.OW_Reset)
    { // OW Reset
        Ch1_1WireBuffer.OW_Reset = 0;
        Ch1_1WireBuffer.RxByte.Val = 0;
        Ch1_1WireBuffer.TxByte.Val = Ch1_1WireBuffer.Val[0];
        Ch1_1WireBuffer.BitCount = 7;
        Ch1_1WireBuffer.BufferIdx = 0;
        Ch1_1WireBuffer.PresencePulse = (!RxBuff.bits.b6) || (!RxBuff.bits.b7);
        if (Ch1_1WireBuffer.PresencePulse)
        {
            SPI1CON1Lbits.SPIEN = 0;
            SPI1BRGLbits.BRG = C_SPI_1W_Data_BRG;
            _SPI1IE = 0;
            SPI1CON1Lbits.SPIEN = 1;
            if (Ch1_1WireBuffer.TxByte.bits.b0 & 0x01)
            {
                _GIE = 0;
                SPI1BUFL = C_SPI_1W_Write_1;
                SPI1CON1Lbits.DISSDO = 0;
                _GIE = 1;
            }
            else
            {
                _GIE = 0;
                SPI1BUFL = C_SPI_1W_Write_0;
                SPI1CON1Lbits.DISSDO = 0;
                _GIE = 1;
            }
            _SPI1IF = 0;
            _SPI1IE = 1;
        }
        else
        {
            Ch1_1WireBuffer.Result = 0xff;
        }
    }
    else
    { // DataTransfer
        //        int8_t i = 0; // Alternativ kann über 3 Bits bei verrauschten Leitungen das Bit ermittelt werden.
        //        if (RxBuff.bits.b12)
        //        {
        //            i++;
        //        }
        //        else
        //        {
        //            i--;
        //        }
        //        if (RxBuff.bits.b11)
        //        {
        //            i++;
        //        }
        //        else
        //        {
        //            i--;
        //        }
        //        if (RxBuff.bits.b10)
        //        {
        //            i++;
        //        }
        //        else
        //        {
        //            i--;
        //        }
        //        RxBuff.bits.b3 = (i > 0);
        Ch1_1WireBuffer.RxByte.Val >>= 1;
        Ch1_1WireBuffer.RxByte.bits.b7 = RxBuff.bits.b12;
        if (Ch1_1WireBuffer.BitCount)
        {
            Ch1_1WireBuffer.BitCount--;
            Ch1_1WireBuffer.TxByte.Val >>= 1;
            if (Ch1_1WireBuffer.TxByte.bits.b0 & 0x01)
            {
                _GIE = 0;
                SPI1BUFL = C_SPI_1W_Write_1;
                SPI1CON1Lbits.DISSDO = 0;
                _GIE = 1;
            }
            else
            {
                _GIE = 0;
                SPI1BUFL = C_SPI_1W_Write_0;
                SPI1CON1Lbits.DISSDO = 0;
                _GIE = 1;
            }
        }
        else
        { // Check Next Byte
            Ch1_1WireBuffer.Val[Ch1_1WireBuffer.BufferIdx] = Ch1_1WireBuffer.RxByte.Val;
            Ch1_1WireBuffer.BufferIdx++;
            if (Ch1_1WireBuffer.BufferIdx < Ch1_1WireBuffer.BufferLen)
            {
                Ch1_1WireBuffer.BitCount = 7;
```

```
                Ch1_1WireBuffer.TxByte.Val = Ch1_1WireBuffer.Val[Ch1_1WireBuffer.BufferIdx];
                if (Ch1_1WireBuffer.TxByte.bits.b0 & 0x01)
                {
                    _GIE = 0;
                    SPI1BUFL = C_SPI_1W_Write_1;
                    SPI1CON1Lbits.DISSDO = 0;
                    _GIE = 1;
                }
                else
                {
                    _GIE = 0;
                    SPI1BUFL = C_SPI_1W_Write_0;
                    SPI1CON1Lbits.DISSDO = 0;
                    _GIE = 1;
                }
            }
            else
            { // transfer complete
                if (CheckCrc((uint8_t*) & Ch1_1WireBuffer.Val[2], 8, Ch1_1WireBuffer.CC_Read.CRC))
                    //                if (CheckCrc((uint8_t*) & Ch1_1WireBuffer.Val[2], Ch1_1WireBuffer.BufferLen - 3,
Ch1_1WireBuffer.CC_Read.CRC)) todo
                { // CRC Error
                    Ch1_1WireBuffer.Result = 0xfe;
                }
                else
                { // CRC OK
                    Ch1_1WireBuffer.Result = 0x01;
                }
            }
        }
    }
}
```

Control-Function:

```
void OW_Ch1_TransferBuffer(uint8_t Read1Bit)
{
    Ch1_1WireBuffer.Result = 0;
    if (Read1Bit)
    {
        Ch1_1WireBuffer.OW_Reset = 0;
        Ch1_1WireBuffer.RxByte.Val = 0;
        Ch1_1WireBuffer.TxByte.Val = 0;
        Ch1_1WireBuffer.BitCount = 0;
        Ch1_1WireBuffer.BufferIdx = 0;
        Ch1_1WireBuffer.BufferLen = 0;
        _GIE = 0;
        SPI1BUFL = C_SPI_1W_Write_1;
        SPI1CON1Lbits.DISSDO = 0;
        _GIE = 1;
    }
    else
    {
        SPI1CON1Lbits.SPIEN = 0;
        SPI1BRGLbits.BRG = C_SPI_1W_Reset_BRG;
        _SPI1IE = 0;
        SPI1CON1Lbits.SPIEN = 1;
        Ch1_1WireBuffer.OW_Reset = 1;
        SPI1BUFL = C_SPI_1W_Reset;
        SPI1CON1Lbits.DISSDO = 0;
        _SPI1IF = 0; // Nach SPIEN = 1; wird das IRQ-Flag gesetzt, muss gelöscht werden da IRQ erst nach übertragung der Buffer Daten
ausgelöst werden soll.
        _SPI1IE = 1;
    }
}
```

Usage:

Set Ch1_1WireBuffer then call OW_Ch1_TransferBuffer(0); and Wait for Completition of BufferTransfer
`(Ch1_1WireBuffer.Result != 0)` 1=OK; 0xff= Error

```
memset(Ch1_1WireBuffer.Val, 0xff, sizeof(Ch1_1WireBuffer.Val));
Ch1_1WireBuffer.CC_Write.ROM_CMD = 0xcc; // SKIP ROM COMMAND
Ch1_1WireBuffer.CC_Write.Function_CMD = 0x44; // CONVERT TEMPERATURE
Ch1_1WireBuffer.BufferLen = 2;
OW_Ch1_TransferBuffer(0);
```

Wait for `(Ch1_1WireBuffer.Result == 1)` and Conversion-Time of Sensor (750ms)

```
memset(Ch1_1WireBuffer.Val, 0xff, sizeof(Ch1_1WireBuffer.Val)); // 0xff in all „read-Byte“ is necessary!
Ch1_1WireBuffer.CC_Write.ROM_CMD = 0xcc; // SKIP ROM COMMAND
Ch1_1WireBuffer.CC_Write.Function_CMD = 0xBE; // READ SCRATCHPAD
Ch1_1WireBuffer.BufferLen = 2;
OW_Ch1_TransferBuffer(0);
```

Wait for `(Ch1_1WireBuffer.Result == 1)`
`Temp = Ch1_1WireBuffer.CC_Read.Temp.Val;`
`if(Ch1_1WireBuffer.Result == 0xff)` => NoResult / No Sensor or Error / invalid Value
`if(Ch1_1WireBuffer.Result == 0xfe)` => CRC-Error / invalid Value

If no Time-Wait is used, the Sensor will indicate when ready with conversion.
Then after Conversion-CMD (0x44), repeatly use:
`OW_Ch1_TransferBuffer(1);` (read only 1 Bit)
and Wait for `(Ch1_1WireBuffer.Result == 1)`
In `Ch1_1WireBuffer.Val[0]` ist the Status-Bit from TempSensor.