Application Notes One Wire Digital Output

1 Introduction

The pressure transmitter automatically outputs pressure data, and when appropriate temperature data, in a fixed interval. The host simply waits for the next data packet. For the digital transmission, the ZACWire™ protocol has been used. It is a two or three byte transmission with Manchester encoding. The first two bytes represent the pressure value. The third byte is the digital temperature value. Huba sensors are by default set to output both temperature and pressure data.

Data interval:
Pressure transmitter: typically 1.64ms
Pressure transmitter with Temperature: typically 1.97ms

Pressure value P0%...P100% = digital value 3000…11000 (0xBB8…0x44C)
Temperature value -50°C…+150°C = digital value 0…255 (0x0…0xFF)

2 Electrical Parameters for One Wire Interface

No.	Parameter	Symbol	Min	Typ	Max	Unit	Comments
1	Pull-up resistor (sensor)	R_{pu}		30		kΩ	
2	Pull-up resistor (customer)	R_{pu_ext}	150			Ω	
3	Rise time	T_{rise}			5	µs	
4	Line resistance 1)	R_{load}			3.9	kΩ	
5	Load capacitance 1)	C_{load}	0	1	5	nF	
6	Voltage low level	V_{low}		0	20	%	Of supply voltage
7	Voltage high level	V_{high}	80	100		%	Of supply voltage
1) The rise time must be $T_{rise} = 2 * R_{load} * C_{load} \leq 5$ µs. If using a pull-up resistor instead of a line resistor, it must meet this specification. The absolute maximum for C_{load} is 5nF.							

Table 1: typical values and parameters

3 Start and Data Transmission

After power-on, the sensor requires a maximum of 8ms for start-up. After that the sensor starts sending its data. The master does not have to initialize the transmission or send a start signal.
On the other hand, the transmission cannot be suppressed.
After finishing a data transmission there is an idle time of approximately 1ms, at the fastest update rate. This means a complete transmission of bridge data, including idle time, typically takes 1.64ms. A complete transmission of bridge and temperature data, including idle time, typically takes 1.97ms.

4 Bit Encoding

The bit format is duty-cycle encoded:

Start bit => 50% duty cycle used to set up strobe time

Logic 1 => 75% duty cycle

Logic 0 => 25% duty cycle

Stop Time

The ZACWire™ bus will be held high for 32µs (nominal) between consecutive data packets regardless of baud rate.

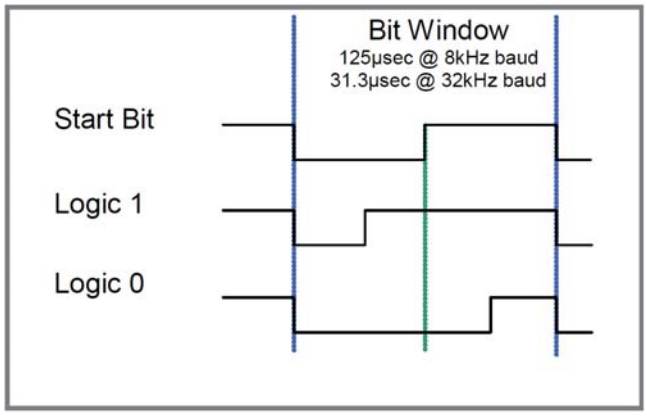

Figure 1: Manchester Duty Cycle

5 Communication Speed and Data Format

The communication speed depends on the update rate of the sensor. There are 4 possible update rates, at 8 Hz and 40 Hz update rate the communication speed is typically 8 kHz (maximum 9.6 kHz) and at 200 Hz and 1 kHz update rate the communication speed is 32 kHz (minimum 26 kHz).
Without a special customer calibration, Huba sensors have an update rate of 1 kHz which results in a minimum baud rate of 26 kHz. The maximum baud rate deviation is in the range of +/- 20%.
Data Bits parity check is Even-Parity.

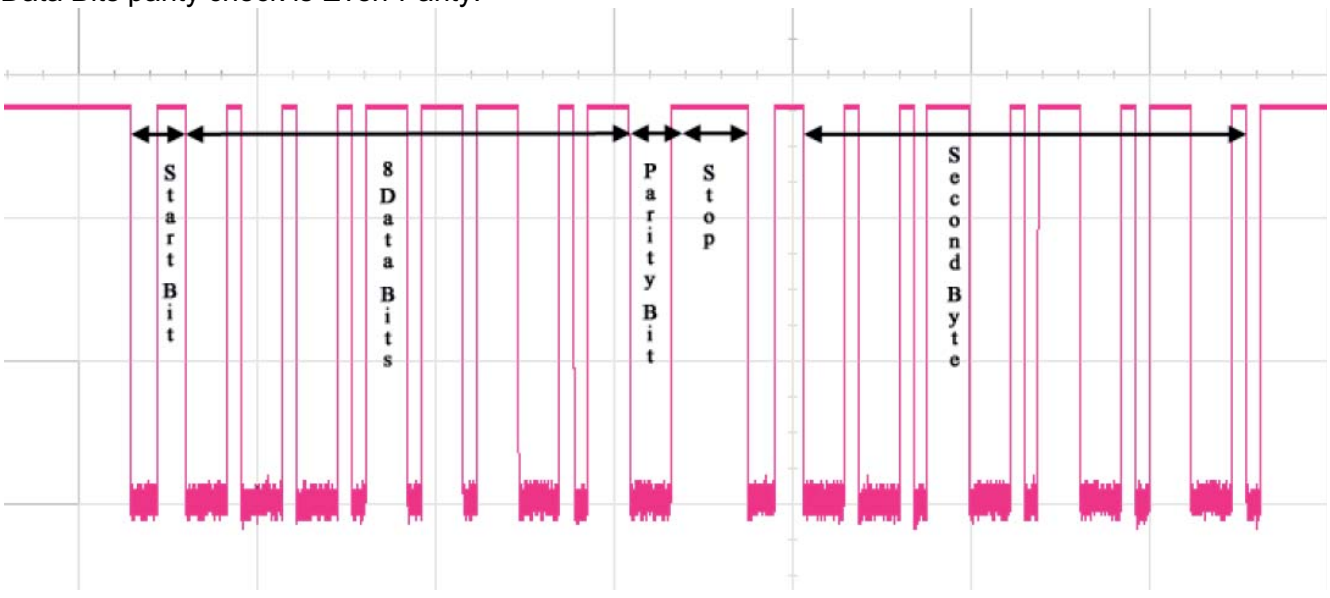

Figure 2: Scope plot of One Wire transmission of 2 Bytes

The 14 bit digital pressure output format is shown in Figure 3 (only pressure, without temperature).

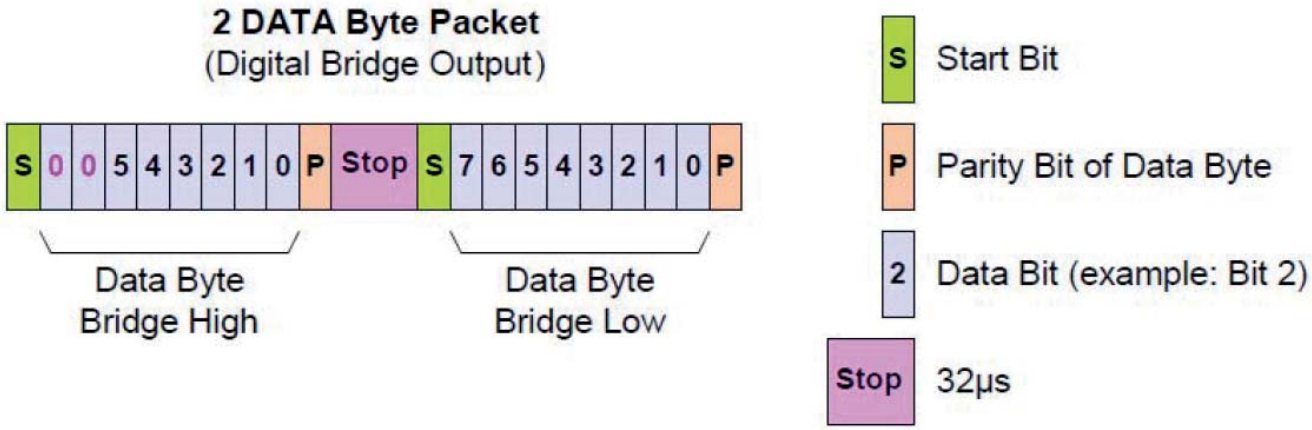

Figure 3: Digital Output Pressure Readings

The Huba sensors normally transmit the pressure and the temperature values. The temperature byte represents an 8-bit temperature quantity spanning from -50 to 150°C.

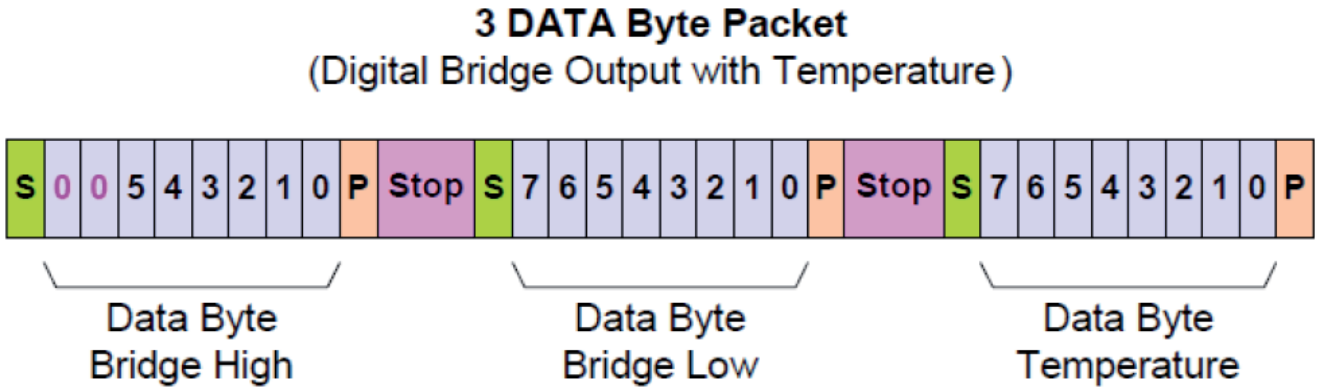

Figure 4: Digital Output Pressure Readings with Temperature

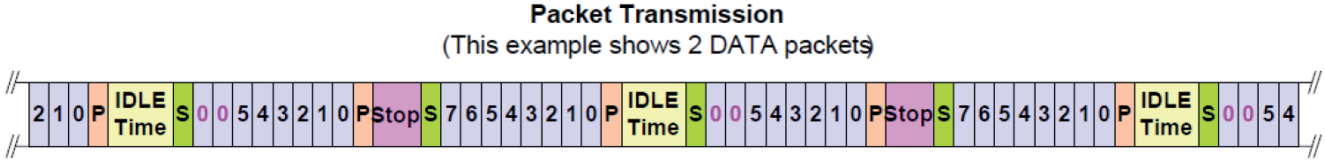

Figure 5: Transmission example of 2 data packets (only pressure data)

6 Temperature Measurement

As the temperature changes more slowly than the pressure signal, the measurement is done in longer intervals.
The temperature is measured every 128 pressure measurement cycles. With an update rate of 1 kHz, such a measurement cycle, including temperature measurement, takes approximately 1ms longer than a pure pressure measurement.

7 Digital Output vs. Resolution

7.1 Pressure signal

The digital pressure output signal is a 14 bit value.
The digital value 3000 represents the 0% FSO point and the digital value 11000 represents the 100% FSO point. That does not mean that the real resolution of the sensor is 8000 Digits!
Huba Control AG normally guarantees minimum 10 bit resolution for their standard products.

7.2 Temperature signal

The digital temperature output signal is an 8 bit value.
The digital value 0 represents -50°C and the digital value 255 represents +150°C.
The temperature range is dependent on the sensor product.
Huba Control AG normally guarantees an 8 bit resolution for their standard products.

8 Sensor failure

In case of a sensor failure the sensor generates a parity error or the data transmission stops.

9 An Example of PIC1 C Code for Reading the ZACwire™

The following C code routine (onewire_int) is for reading a byte of ZACwire™ transmission from a pressure sensor. In this application, the ZACwire™ was connected to pin 0 of portB (PORTBbits.RB0). This is the interrupt pin of a PIC18F1220. It is best to have the ZACwire™ connected to a pin of the µController that can cause an interrupt on a falling edge. This routine should be called from the ISR when a falling edge of the ZACwire™ signal (1st falling edge of a start bit) is detected. In this case, the µController (PIC18F1220) is running at 24MHz, although this routine should work well for speeds between 16MHz and 24MHz.

```
/*****************************
* Global Variable Definitions *
*****************************/
unsigned char data_byte;


unsigned char onewire_int (void)
{
        /*********************************************************
        * This routine returns two bytes. data_byte is a global *
        * unsigned char which will hold the data read from the *
        * onewire. The unsigned char the routine returns will *
        * be a status indicating the state of the read *
```

```
* 0x01 = Good *
* 0x00 = Time out *
* 0x05 = Parity error *
*********************************************************/
unsigned char strb_cnt,bit_cnt,Tstrobe,prty_cnt;
unsigned char time_out;
/*******************************
* Time the start bit low period *
*******************************/
Tstrobe=0x02;                   /* start non-zero to account for overhead (ISR latency) */
while (!(PORTBbits.RB0)) {      /* 7 state loop (access RAM mode) */
        if (++Tstrobe==0x00) {
                        return(0x00); /* if Tstrobe rolled over then we have a timeout */
        }
}
/***********************************************
* Tstrobe now contains a time start bit was low *
* Now clear data_byte and parity counter before *
* next fall of onewire *
***********************************************/
data_byte = 0x00;
prty_cnt=0x00;
/*****************************************************
* Now for the 8 bits in a byte wait for falling edge *
*****************************************************/
for (bit_cnt=0; bit_cnt<8; bit_cnt++) {
        time_out=0x00;
        strb_cnt=0x00;
        data_byte = (data_byte<<1);     /* shift while there is time */
        while (PORTBbits.RB0) {
                if (++time_out==0x00) { /* wait for falling edge of data bit */
                return(0x00);           /* if time_out rolls over then have a timeout */
                }
        }
        /***************************************
        * Timer loop until strb_cnt = Tstrobe *
        * NOTE: it is important that this loop *
        * takes the same time as the loop above *
        * that timed the start bit and setup *
        * Tstrobe *
```

```
        ****************************************/
        while (strb_cnt!=Tstrobe) { /* 7 state loop when compiled */
                Nop(); /* in access RAM mode */
                strb_cnt++;
        }
        /*********************************
        * Now sample state of onewire *
        *********************************/
        if (PORTBbits.RB0) {
                data_byte++; /* if onewire high then set the LSB of data_byte */
                prty_cnt++; /* if onewire high then increment the parity counter */
        }
        /************************************************
        * Now wait for rise if it has not already happened *
        ************************************************/
        time_out =0x00;
        while (!(PORTBbits.RB0)) { /* wait for rising edge of data bit */
                if (++time_out==0x00) { /* if time_out rolls over then have a timeout */
                        return(0x00);
                }
        }
}
strb_cnt=0x00
/*********************************
* Now wait for falling edge of parity bit *
*********************************/
time_out = 0x00;
while (PORTBbits.RB0) {
        if (++time_out==0x00) {
                return(0x00);
        }
}
/*************************************
* Timer loop till strb_cnt = Tstrobe *
*************************************/
while (strb_cnt!=Tstrobe) {
        Nop(); /* 7 state loop */
        strb_cnt++;
}
/************************
```

```
        * Now sample parity bit *
        ************************/
        if (PORTBbits.RB0) {
                prty_cnt++;
        }
        if (prty_cnt % 2) { /* parity_cnt must be even */
                return(0x05); /* parity error */
        }
        /***********************************
        * Now wait for rise of parity if it *
        * has not already happened *
        ***********************************/
        time_out=0x00;
        while (!(PORTBbits.RB0)) {
                if (++time_out==0x00) {
                        return(0x00);
                }
        }
        return(0x01); /* Good data in data_byte */
}
```