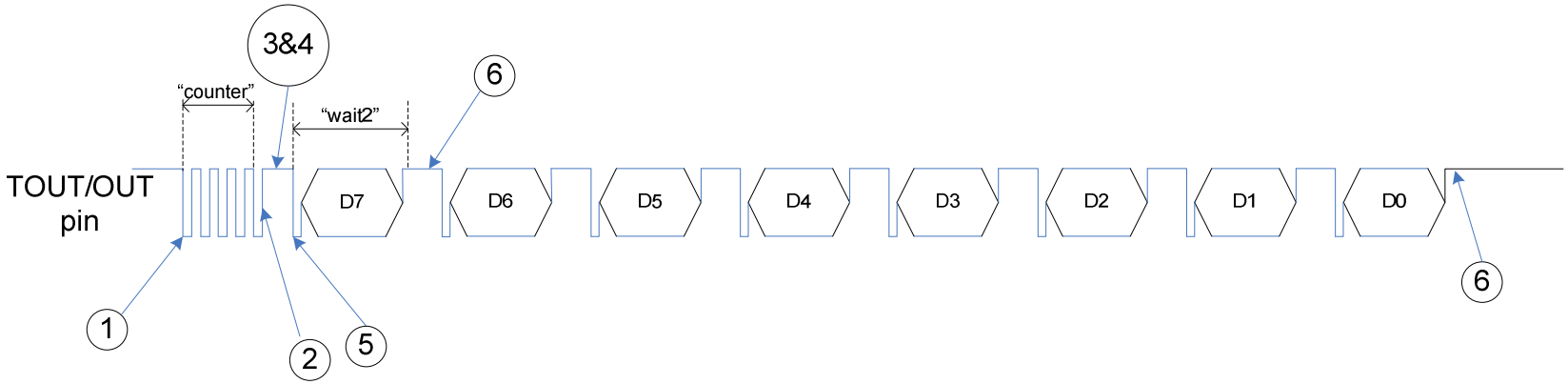
Application Note: AZD017
1-Wire Protocol Sample Code

1 Overview

Some ProxSense devices use a 1-wire protocol for communication with a micro controller (MCU). Easy implementation is possible using timer modules found on most MCUs, or the user should implement the code as done in Section 2 (custom counters). A consideration that should be taken when choosing a MCU is the instruction execution speed. Calculate if the MCU can set up the timers, check the timers and check pin polarity in the time of one data bit. The time for one bit is approximately “counter” or Timer2, which is the time for the 0xAA synchronising byte, divided by 8. Timer2 can vary between 144us and 176us, with 160us optimally. Bit time = 160us/8 = 20us.

2 Assembly implementation (PIC18F4550, 20Mhz, 1x8bit instruction every 4 clock cycles)



1) Wait for falling edge
2) Wait for 8 edges and then stop edge (rising edge is 9th edge), use “counter” to count during 8 edges
3) Do Calculations
   - bit period = “counter”/8 – 2 = "wait2" (subtract 2 for overheads)
4) Byte0
   - Set up timer (“wait1”) to wait until it is in the middle of the start bit

- "wait1" is time needed before starting "wait2" where "wait1" = ("wait2" + overheads) / 2)

5) Wait for start bit to go low and start logging data
   - Wait for start bit to go low
   - Start "wait1" timer
   - Set up bit timer ("wait2") when "wait1" has counted out
   - Wait until "wait2" counts out
   - Read status of pin and record bit
   - Repeat until 8 bits of Byte0 are recorded
6) Record Byte1 to Byte7 the same way as done from step (4)
7) Move bytes to buffer ("counter" is period of synchronisation byte, "wait" is period of a bit)

```
/*****************************************************************************/
byte Stream127(byte *buf)
/*****************************************************************************/
{
  byte  err;

  err = 0;
  counter = 0;
  byte0 = 0x00;
  byte1 = 0x00;
  byte2 = 0x00;
  byte3 = 0x00;
  byte4 = 0x00;
  byte5 = 0x00;
  byte6 = 0x00;
  byte7 = 0x00;
  WriteOutput(TO4_SCK_126, LOW);

  buf = &buf[MES_DATA];

_asm

  #define rc0 0x0F82,0x00,0x0  //TOUT/OUT is on Port RC0 of PIC
  #define debug_pin 0x0F81,0x01,0x0  //connector P1pin9 is on Port RB1 of PIC
```

```
/* Find Start Bit of synch byte ***************************************/
// (1) wait for first falling edge on TOUT/OUT
waitP3low0:
  BTFSC rc0
  goto waitP3low0

/* (2) Wait for 8 edges on TOUT/OUT to determine average clock period ***/
waitP3high0:
  INCF  counter, 1, 1
  BTFSS rc0
  goto waitP3high0
waitP3low1:
  INCF  counter, 1, 1
  BTFSC rc0
  goto waitP3low1
waitP3high1:
  INCF  counter, 1, 1
  BTFSS rc0
  goto waitP3high1
waitP3low2:
  INCF  counter, 1, 1
  BTFSC rc0
  goto waitP3low2
waitP3high2:
  INCF  counter, 1, 1
  BTFSS rc0
  goto waitP3high2
waitP3low3:
  INCF  counter, 1, 1
  BTFSC rc0
  goto waitP3low3
waitP3high3:
  INCF  counter, 1, 1
  BTFSS rc0
  goto waitP3high3
waitP3low4:
  INCF  counter, 1, 1
  BTFSC rc0
```

```
    goto waitP3low4
  waitP3high4:
    BTFSS rc0        //wait for stop bit, no increment of counter anymore
    goto waitP3high4

    //(3) Do Calculations
    ADDLW 0x00         //simple way to clear carry bit
    RRCF counter, 1, 1 //divide by 2
    ADDLW 0x00         //simple way to clear carry bit
    RRCF counter, 1, 1 //divide by 2
    ADDLW 0x00         //simple way to clear carry bit
    RRCF counter, 1, 1 //divide by 2
    MOVFF counter, wait1  //wait1 is longer timer for delay without other overheads
    MOVFF counter, wait2  //wait2 is shorter timer for delays with bit check, byte shift, etc. overheads
    MOVLW 2
    SUBWF wait2, 1, 1     //subtract overheads

  /*(4) BYTE 0 ************************************************************/
    //prepare timer before waiting for start bit
    MOVFF wait1, timer
    ADDLW 0x00          //simple way to clear carry bit
    RRCF timer, 1, 1    //divide timer by 2 to get half a bit's timing
    //set up bit counter
    MOVLW 8
    MOVWF bitnr, 1

  //(5) wait for start bit to go low and start logging data
  waitP3start0:
    BTFSC rc0
    goto waitP3start0

    //wait to middle of start bit
  MiddleStartBit0:
    DECFSZ timer, 1, 1
    goto MiddleStartBit0

  ReadByte0:
  //wait till middle of bit 0
    MOVFF wait2, timer //setup timer
```

```
MidBit0:
  NOP
  DECFSZ timer, 1, 1
  goto MidBit0
  RLNCF byte0, 1, 1   //rotate to next bit position
  BTFSC rc0           //check bit, skip next step if low
  BSF byte0, 0, 1     //set bit_0 status if high
  DECFSZ bitnr, 1, 1
  goto ReadByte0      //if not last bit, then go to next bit
  //wait until middle of stop bit
  MOVFF counter, timer
MiddleStopBit0:
  DECFSZ timer, 1, 1
  goto MiddleStopBit0

/* BYTE 1 ************************************************************/
  //prepare timer before waiting for start bit
  MOVFF wait1, timer
  ADDLW 0x00          //simple way to clear carry bit
  RRCF timer, 1, 1    //divide timer by 2 to get half a bit's timing
  //set up bit counter
  MOVLW 8
  MOVWF bitnr, 1
  // wait for start bit to go low
waitP3start1:
  BTFSC rc0
  goto waitP3start1
  //wait to middle of start bit
MiddleStartBit1:
  DECFSZ timer, 1, 1
  goto MiddleStartBit1

ReadByte1:
//wait till middle of bit 0
  MOVFF wait2, timer //setup timer
MidBit1:
  NOP
  DECFSZ timer, 1, 1
  goto MidBit1
```

```
  RLNCF byte1, 1, 1   //rotate to next bit position
  BTFSC rc0           //check bit, skip next step if low
  BSF byte1, 0, 1     //set bit_0 status if high
  DECFSZ bitnr, 1, 1
  goto ReadByte1      //if not last bit, then go to next bit
  //wait until middle of stop bit
  MOVFF counter, timer
MiddleStopBit1:
  DECFSZ timer, 1, 1
  goto MiddleStopBit1

// (6)
/* BYTE 2 *************************************************************/
  //prepare timer before waiting for start bit
  MOVFF wait1, timer
  ADDLW 0x00          //simple way to clear carry bit
  RRCF timer, 1, 1    //divide timer by 2 to get half a bit's timing
  //set up bit counter
  MOVLW 8
  MOVWF bitnr, 1
  // wait for start bit to go low
waitP3start2:
  BTFSC rc0
  goto waitP3start2
  //wait to middle of start bit
MiddleStartBit2:
  DECFSZ timer, 1, 1
  goto MiddleStartBit2

ReadByte2:
//wait till middle of bit 0
  MOVFF wait2, timer //setup timer
MidBit2:
  NOP
  DECFSZ timer, 1, 1
  goto MidBit2
  RLNCF byte2, 1, 1   //rotate to next bit position
  BTFSC rc0           //check bit, skip next step if low
  BSF byte2, 0, 1     //set bit_0 status if high
```

```
    DECFSZ bitnr, 1, 1
    goto ReadByte2      //if not last bit, then go to next bit
    //wait until middle of stop bit
    MOVFF counter, timer
  MiddleStopBit2:
    DECFSZ timer, 1, 1
    goto MiddleStopBit2

  /* BYTE 3 *************************************************************/
    //prepare timer before waiting for start bit
    MOVFF wait1, timer
    ADDLW 0x00          //simple way to clear carry bit
    RRCF timer, 1, 1    //divide timer by 2 to get half a bit's timing
    //set up bit counter
    MOVLW 8
    MOVWF bitnr, 1
    // wait for start bit to go low
  waitP3start3:
    BTFSC rc0
    goto waitP3start3
    //wait to middle of start bit
  MiddleStartBit3:
    DECFSZ timer, 1, 1
    goto MiddleStartBit3

  ReadByte3:
  //wait till middle of bit 0
    MOVFF wait2, timer //setup timer
  MidBit3:
    NOP
    DECFSZ timer, 1, 1
    goto MidBit3
    RLNCF byte3, 1, 1   //rotate to next bit position
    BTFSC rc0           //check bit, skip next step if low
    BSF byte3, 0, 1     //set bit_0 status if high
    DECFSZ bitnr, 1, 1
    goto ReadByte3      //if not last bit, then go to next bit
    //wait until middle of stop bit
    MOVFF counter, timer
```

```
MiddleStopBit3:
  DECFSZ timer, 1, 1
  goto MiddleStopBit3

/* BYTE 4 *************************************************************/
  //prepare timer before waiting for start bit
  MOVFF wait1, timer
  ADDLW 0x00          //simple way to clear carry bit
  RRCF timer, 1, 1    //divide timer by 2 to get half a bit's timing
  //set up bit counter
  MOVLW 8
  MOVWF bitnr, 1
  // wait for start bit to go low
waitP3start4:
  BTFSC rc0
  goto waitP3start4
  //wait to middle of start bit
MiddleStartBit4:
  DECFSZ timer, 1, 1
  goto MiddleStartBit4

ReadByte4:
//wait till middle of bit 0
  MOVFF wait2, timer //setup timer
MidBit4:
  NOP
  DECFSZ timer, 1, 1
  goto MidBit4
  RLNCF byte4, 1, 1   //rotate to next bit position
  BTFSC rc0           //check bit, skip next step if low
  BSF byte4, 0, 1     //set bit_0 status if high
  DECFSZ bitnr, 1, 1
  goto ReadByte4      //if not last bit, then go to next bit
  //wait until middle of stop bit
  MOVFF counter, timer
MiddleStopBit4:
  DECFSZ timer, 1, 1
  goto MiddleStopBit4
```

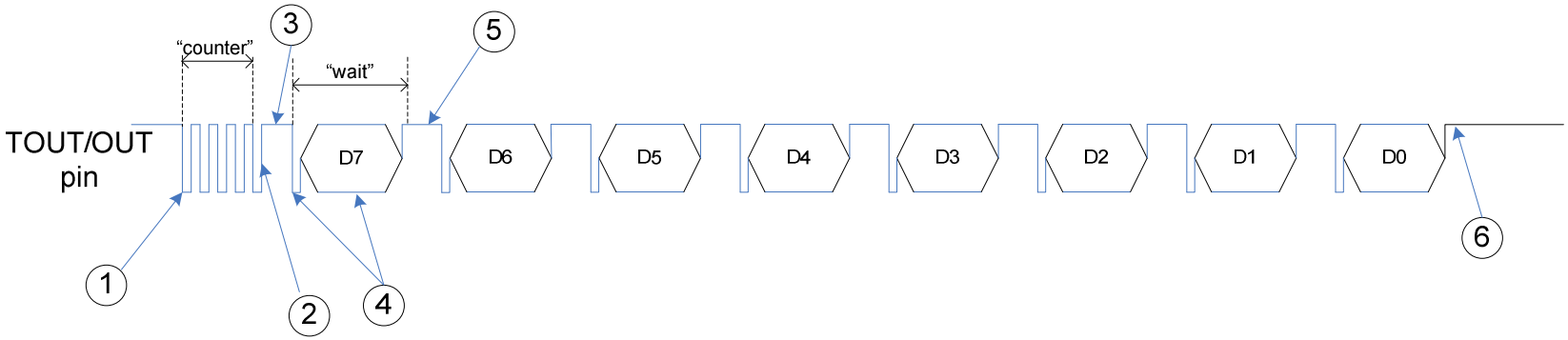
```
/* BYTE 5 ************************************************************/
  //prepare timer before waiting for start bit
  MOVFF wait1, timer
  ADDLW 0x00          //simple way to clear carry bit
  RRCF timer, 1, 1    //divide timer by 2 to get half a bit's timing
  //set up bit counter
  MOVLW 8
  MOVWF bitnr, 1
  // wait for start bit to go low
waitP3start5:
  BTFSC rc0
  goto waitP3start5
  //wait to middle of start bit
MiddleStartBit5:
  DECFSZ timer, 1, 1
  goto MiddleStartBit5

ReadByte5:
//wait till middle of bit 0
  MOVFF wait2, timer //setup timer
MidBit5:
  NOP
  DECFSZ timer, 1, 1
  goto MidBit5
  RLNCF byte5, 1, 1   //rotate to next bit position
  BTFSC rc0           //check bit, skip next step if low
  BSF byte5, 0, 1     //set bit_0 status if high
  DECFSZ bitnr, 1, 1
  goto ReadByte5      //if not last bit, then go to next bit
  //wait until middle of stop bit
  MOVFF counter, timer
MiddleStopBit5:
  DECFSZ timer, 1, 1
  goto MiddleStopBit5

/* BYTE 6 ************************************************************/
  //prepare timer before waiting for start bit
  MOVFF wait1, timer
  ADDLW 0x00          //simple way to clear carry bit
```

```
    RRCF timer, 1, 1    //divide timer by 2 to get half a bit's timing
    //set up bit counter
    MOVLW 8
    MOVWF bitnr, 1
    // wait for start bit to go low
  waitP3start6:
    BTFSC rc0
    goto waitP3start6
    //wait to middle of start bit
  MiddleStartBit6:
    DECFSZ timer, 1, 1
    goto MiddleStartBit6

  ReadByte6:
  //wait till middle of bit 0
    MOVFF wait2, timer //setup timer
  MidBit6:
    NOP
    DECFSZ timer, 1, 1
    goto MidBit6
    RLNCF byte6, 1, 1   //rotate to next bit position
    BTFSC rc0           //check bit, skip next step if low
    BSF byte6, 0, 1     //set bit_0 status if high
    DECFSZ bitnr, 1, 1
    goto ReadByte6      //if not last bit, then go to next bit
    //wait until middle of stop bit
    MOVFF counter, timer
  MiddleStopBit6:
    DECFSZ timer, 1, 1
    goto MiddleStopBit6

  /* BYTE 7 ************************************************************/
    //prepare timer before waiting for start bit
    MOVFF wait1, timer
    ADDLW 0x00          //simple way to clear carry bit
    RRCF timer, 1, 1    //divide timer by 2 to get half a bit's timing
    //set up bit counter
    MOVLW 8
    MOVWF bitnr, 1
```

```
    // wait for start bit to go low
  waitP3start7:
    BTFSC rc0
    goto waitP3start7
    //wait to middle of start bit
  MiddleStartBit7:
    DECFSZ timer, 1, 1
    goto MiddleStartBit7

  ReadByte7:
  //wait till middle of bit 0
    MOVFF wait2, timer //setup timer
  MidBit7:
    NOP
    DECFSZ timer, 1, 1
    goto MidBit7
    RLNCF byte7, 1, 1   //rotate to next bit position
    BTFSC rc0           //check bit, skip next step if low
    BSF byte7, 0, 1     //set bit_0 status if high
    DECFSZ bitnr, 1, 1
    goto ReadByte7      //if not last bit, then go to next bit
    //wait until middle of stop bit
    MOVFF counter, timer
  MiddleStopBit7:
    DECFSZ timer, 1, 1
    goto MiddleStopBit7

_endasm

  Delay_200us(0x00,0x02);

  *(buf+0) = 0xAA;
  *(buf+1) = byte0;
  *(buf+2) = byte1;
  *(buf+3) = byte2;
  *(buf+4) = byte3;
  *(buf+5) = byte4;
  *(buf+6) = byte5;
  *(buf+7) = byte6;
```

```
  *(buf+8) = byte7;
  *(buf+9) = 0xDB;  //Indicates that Debug Data will follow
  *(buf+10) = counter;

  return 0x00;

}/* Stream127 end*/
```

3 Alternative Assembly implementation for fast MCU



1) Wait for falling edge
2) Wait for 8 edges and then stop edge (rising edge is 9th edge)
3) Do Calculations (calculate bit period = "wait")
4) Byte0
    a. set up bit timer (wait)
    b. Wait for start bit to go low
    c. Wait until "wait" counts out
    d. Read status of pin and record bit
    e. Repeat until 8 bits of Byte0 are recorded
5) Record Byte1 to Byte7 the same way as (4)
6) Move bytes to buffer ("counter" is period of synchronisation byte, "wait" is period of a bit)

```
/*****************************************************************************/
void azq127_Stream(char *buf)
/*****************************************************************************/
{
  byte  lees1, lees2, lees3, lees4, lees5, lees6;
  byte  err;
  byte  timeout;

  err = 0;
  *(buf+0) = 0x00;
  counter = 0;
  byte0 = 0x00;
  byte1 = 0x00;
  byte2 = 0x00;
  byte3 = 0x00;
  byte4 = 0x00;
  byte5 = 0x00;
  byte6 = 0x00;
  byte7 = 0x00;

//  gnl_WriteOutput(BE, LOW);

_asm

  #define rc0 0x0F82,0x00,0x0  //TOUT/OUT is on Port RC0 of PIC

  //(1)
  /* Find Start Bit of synchronisation byte ***************************************/
  // wait for first falling edge on TOUT/OUT
```

```
 waitP3low0:
   BTFSC rc0
   goto waitP3low0

//(2)
 /* wait for 8 edges on TOUT/OUT to determine period of 8 bits (one byte) data */
 waitP3high0:
   INCF  counter, 1, 1
   BTFSS rc0
   goto waitP3high0
 waitP3low1:
   INCF  counter, 1, 1
   BTFSC rc0
   goto waitP3low1
 waitP3high1:
   INCF  counter, 1, 1
   BTFSS rc0
   goto waitP3high1
 waitP3low2:
   INCF  counter, 1, 1
   BTFSC rc0
   goto waitP3low2
 waitP3high2:
   INCF  counter, 1, 1
   BTFSS rc0
   goto waitP3high2
 waitP3low3:
   INCF  counter, 1, 1
   BTFSC rc0
   goto waitP3low3
 waitP3high3:
   INCF  counter, 1, 1
   BTFSS rc0
   goto waitP3high3
 waitP3low4:
   INCF  counter, 1, 1
   BTFSC rc0
   goto waitP3low4
```

```
waitP3high4:
  BTFSS rc0       //wait for stop bit, no increment of counter anymore
  goto waitP3high4

        //(3)
        //Do Calculations
  //Calculate bit period
  MOVFF counter, wait  //wait is timer for delays with bit check, byte shift, etc. overheads
  ADDLW 0x00           //simple way to clear carry bit
  RRCF wait, 1, 1    //divide by 2
  ADDLW 0x00           //simple way to clear carry bit
  RRCF wait, 1, 1    //divide by 2
  ADDLW 0x00           //simple way to clear carry bit
  RRCF wait, 1, 1    //divide by 2
  MOVLW 1
  SUBWF wait, 1, 1     //subtract 1 to make provision for bit check, byte shift, etc. overheads

//(4)
/* BYTE 0 ***************************************************************/
  //set up bit counter
  MOVLW 8
  MOVWF bitnr, 1
// wait for start bit to go low
waitP3start0:
  BTFSC rc0
  goto waitP3start0

// Get Byte0
ReadByte0:
  MOVFF wait, timer //setup timer
  //wait untill bit is valid
  StartBit0:
    NOP
    DECFSZ timer, 1, 1
    goto StartBit0
    RLNCF byte0, 1, 1   //rotate to next bit position
    BTFSC rc0          //check bit, skip next step if low
    BSF byte0, 0, 1     //set bit_0 status if high
```

```
    DECFSZ bitnr, 1, 1
    goto ReadByte0      //if not last bit, then go to next bit
//wait for stop bit (if last bit was low)
WaitStopBit0:
  BTFSS rc0
  goto WaitStopBit0

//(5)
/* BYTE 1 *****************************************************************/
  //set up bit counter
  MOVLW 8
  MOVWF bitnr, 1
// wait for start bit to go low
waitP3start1:
  BTFSC rc0
  goto waitP3start1

// Get Byte1
ReadByte1:
  MOVFF wait, timer //setup timer
  //wait untill bit is valid
  StartBit1:
    NOP
    DECFSZ timer, 1, 1
    goto StartBit1
    RLNCF byte1, 1, 1   //rotate to next bit position
    BTFSC rc0           //check bit, skip next step if low
    BSF byte1, 0, 1     //set bit_0 status if high
    DECFSZ bitnr, 1, 1
    goto ReadByte1      //if not last bit, then go to next bit
//wait for stop bit (if last bit was low)
WaitStopBit1:
  BTFSS rc0
  goto WaitStopBit1

/* BYTE 2 *****************************************************************/
  //set up bit counter
  MOVLW 8
```

```
  MOVWF bitnr, 1
// wait for start bit to go low
waitP3start2:
  BTFSC rc0
  goto waitP3start2

// Get Byte2
ReadByte2:
  MOVFF wait, timer //setup timer
  //wait untill bit is valid
  StartBit2:
    NOP
    DECFSZ timer, 1, 1
    goto StartBit2
    RLNCF byte2, 1, 1   //rotate to next bit position
    BTFSC rc0           //check bit, skip next step if low
    BSF byte2, 0, 1     //set bit_0 status if high
    DECFSZ bitnr, 1, 1
    goto ReadByte2      //if not last bit, then go to next bit
//wait for stop bit (if last bit was low)
WaitStopBit2:
  BTFSS rc0
  goto WaitStopBit2

/* BYTE 3 ***************************************************************/
  //set up bit counter
  MOVLW 8
  MOVWF bitnr, 1
// wait for start bit to go low
waitP3start3:
  BTFSC rc0
  goto waitP3start3

// Get Byte3
ReadByte3:
  MOVFF wait, timer //setup timer
  //wait untill bit is valid
  StartBit3:
```

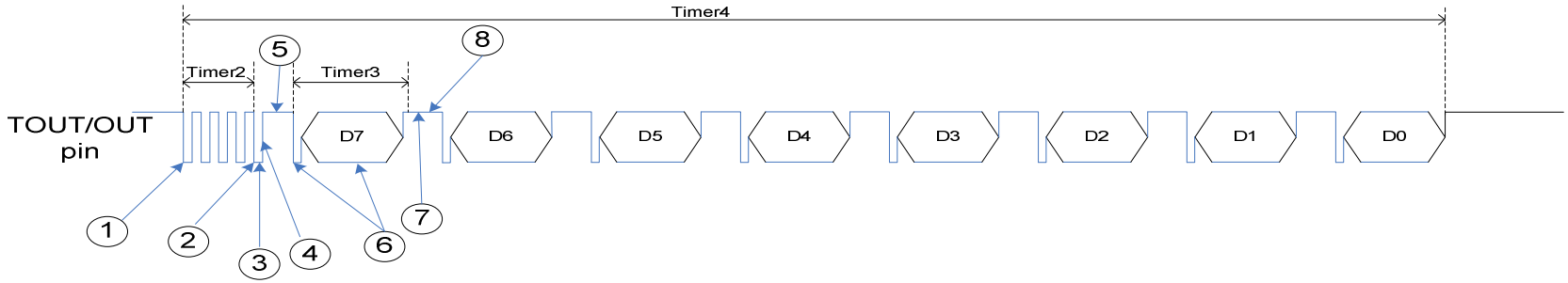
```
    NOP
    DECFSZ timer, 1, 1
    goto StartBit3
    RLNCF byte3, 1, 1   //rotate to next bit position
    BTFSC rc0           //check bit, skip next step if low
    BSF byte3, 0, 1     //set bit_0 status if high
    DECFSZ bitnr, 1, 1
    goto ReadByte3      //if not last bit, then go to next bit
//wait for stop bit (if last bit was low)
WaitStopBit3:
  BTFSS rc0
  goto WaitStopBit3

/* BYTE 4 *************************************************************/
  //set up bit counter
  MOVLW 8
  MOVWF bitnr, 1
// wait for start bit to go low
waitP3start4:
  BTFSC rc0
  goto waitP3start4

// Get Byte4
ReadByte4:
  MOVFF wait, timer //setup timer
  //wait untill bit is valid
  StartBit4:
    NOP
    DECFSZ timer, 1, 1
    goto StartBit4
    RLNCF byte4, 1, 1   //rotate to next bit position
    BTFSC rc0           //check bit, skip next step if low
    BSF byte4, 0, 1     //set bit_0 status if high
    DECFSZ bitnr, 1, 1
    goto ReadByte4      //if not last bit, then go to next bit
//wait for stop bit (if last bit was low)
WaitStopBit4:
  BTFSS rc0
```

```
  goto WaitStopBit4

/* BYTE 5 ***************************************************************/
  //set up bit counter
  MOVLW 8
  MOVWF bitnr, 1
// wait for start bit to go low
waitP3start5:
  BTFSC rc0
  goto waitP3start5

// Get Byte5
ReadByte5:
  MOVFF wait, timer //setup timer
  //wait untill bit is valid
  StartBit5:
    NOP
    DECFSZ timer, 1, 1
    goto StartBit5
    RLNCF byte5, 1, 1   //rotate to next bit position
    BTFSC rc0           //check bit, skip next step if low
    BSF byte5, 0, 1     //set bit_0 status if high
    DECFSZ bitnr, 1, 1
    goto ReadByte5      //if not last bit, then go to next bit
//wait for stop bit (if last bit was low)
WaitStopBit5:
  BTFSS rc0
  goto WaitStopBit5

/* BYTE 6 ***************************************************************/
  //set up bit counter
  MOVLW 8
  MOVWF bitnr, 1
// wait for start bit to go low
waitP3start6:
  BTFSC rc0
  goto waitP3start6
```

```
// Get Byte6
ReadByte6:
  MOVFF wait, timer //setup timer
  //wait untill bit is valid
  StartBit6:
    NOP
    DECFSZ timer, 1, 1
    goto StartBit6
    RLNCF byte6, 1, 1   //rotate to next bit position
    BTFSC rc0           //check bit, skip next step if low
    BSF byte6, 0, 1     //set bit_0 status if high
    DECFSZ bitnr, 1, 1
    goto ReadByte6      //if not last bit, then go to next bit
//wait for stop bit (if last bit was low)
WaitStopBit6:
  BTFSS rc0
  goto WaitStopBit6

/* BYTE 7 ************************************************************/
  //set up bit counter
  MOVLW 8
  MOVWF bitnr, 1
// wait for start bit to go low
waitP3start7:
  BTFSC rc0
  goto waitP3start7

// Get Byte7
ReadByte7:
  MOVFF wait, timer //setup timer
  //wait untill bit is valid
  StartBit7:
    NOP
    DECFSZ timer, 1, 1
    goto StartBit7
    RLNCF byte7, 1, 1   //rotate to next bit position
    BTFSC rc0           //check bit, skip next step if low
    BSF byte7, 0, 1     //set bit_0 status if high
```

```
    DECFSZ bitnr, 1, 1
    goto ReadByte7      //if not last bit, then go to next bit
  //wait for stop bit (if last bit was low)
  WaitStopBit7:
    BTFSS rc0
    goto WaitStopBit7

_endasm

  Delay1KTCYx(2);

  *(buf+0) = err;
          *(buf+1) = 0xAA;
          *(buf+2) = byte0;
          *(buf+3) = byte1;
          *(buf+4) = byte2;
          *(buf+5) = byte3;
          *(buf+6) = byte4;
          *(buf+7) = byte5;
          *(buf+8) = byte6;
          *(buf+9) = byte7;
  *(buf+10) = 0xDD; //Indicates that Debug Data will follow
  *(buf+11) = counter;
  *(buf+12) = wait;

}/* azq127_Stream end*/
```

4 C Implementation (STM32F103RBT6, 72MHz, 1x32bit instruction every clock cycle)



TIMER2: Used to do timings related with synchronisation byte. 250ns resolution. Set up in step (5).
TIMER3: Used to do timings related with the data bytes. Set up done before step (1).
TIMER4: Used as WDT for every data serial word. Set up done before step (1).

1) Wait for falling edge (TOUT/OUT non-active state is HIGH) and start TIMER2 with first falling edge (START edge)
2) Wait for 8 edges (excluding START falling edge). The 8th edge change is a falling edge and also the STOP falling edge
3) Do calculations in this time:
    - Stop TIMER2
    - Get TIMER2 counter value = X (TIMER2 counted in 250ns steps between START edge and STOP edge)
    - Calculate bit period: Bit period = X / 8 + 1 (take the bit time and 1 count more to compensate for rounding errors)
4) Wait for rising edge on pin
5) Set up TIMER3 during this time
6) Start getting the data
    - Wait for falling edge on pin

- Start TIMER3
- TIMER3 will time-out (TIMER3 will not time-out while the pin is in its transition stage)
- Read the status of the pin, and record the bit
- Reset TIMER3 and repeat until all 8 bits are read for a byte

7) Store Byte data
   - Reset TIMER3
   - Store byte in buffer
   - Read the status of the pin, it should be HIGH, continue if HIGH

8) Repeat from step 6 until all 8 bytes are read and stored

```
////////////// Timer Interrupt handlers //////////////
void IRQ_Init(void)
{
  NVIC_InitTypeDef NVIC_InitStruct;

  NVIC_InitStruct.NVIC_IRQChannel=TIM2_IRQn;
  NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority=2;
  NVIC_InitStruct.NVIC_IRQChannelSubPriority=0;
  NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
  NVIC_Init(&NVIC_InitStruct);

  NVIC_InitStruct.NVIC_IRQChannel=TIM3_IRQn;
  NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority=2;
  NVIC_InitStruct.NVIC_IRQChannelSubPriority=0;
  NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
  NVIC_Init(&NVIC_InitStruct);

  NVIC_InitStruct.NVIC_IRQChannel=TIM4_IRQn;
  NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority=2;
  NVIC_InitStruct.NVIC_IRQChannelSubPriority=0;
  NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
  NVIC_Init(&NVIC_InitStruct);
}

/*******************************************************************************
* Function Name  : TIM2_IRQHandler
* Description    : This function handles TIM2 global interrupt request.
* Input          : None
* Output         : None
```

```
* Return         : None
*******************************************************************************/
void TIM2_IRQHandler(void)
{
  //TIM2->CR1 = TIM2->CR1 & 0xFE;   //do not stop counter2 - it must keep on running for its intended purpose during 127 streaming
  Timer2Expired = 1;
  TIM_ClearITPendingBit(TIM2, TIM_IT_Update);
}

/*******************************************************************************
* Function Name  : TIM3_IRQHandler
* Description    : This function handles TIM3 global interrupt request.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void TIM3_IRQHandler(void)
{
  //TIM3->CR1 = TIM3->CR1 & 0xFE;   //do not stop counter2 - it must keep on running for its intended purpose during 127 streaming
  Timer3Expired = 1;
  TIM_ClearITPendingBit(TIM3, TIM_IT_Update);
}

/*******************************************************************************
* Function Name  : TIM4_IRQHandler
* Description    : This function handles TIM4 global interrupt request.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void TIM4_IRQHandler(void)
{
  TIM4->CR1 = TIM4->CR1 & 0xFE;   //stop counter
  Timer4Expired = 1;
  TIM_ClearITPendingBit(TIM4, TIM_IT_Update);
}

////////////// Timer setup //////////////////
```

```
/*******************************************************************************
* Function Name  : Timer2Init.
* Description    : Configures the Timer2.
* Input          : TimingValue (u16), PreScaleValue (u16)
* Output         : None.
* Return         : None.
*******************************************************************************/
void Timer2Init(u16 TimingValue, u16 PreScaleValue)
{
  TIM2->CR1 = 0x80;  //ControlRegister 1: Auto-Reload Preload enabled, Counter used as upcounter, Stop counter.
  TIM2->CR2 = 0x00;  //ControlRegister 2
  TIM2->SMCR = 0x00; //Slave Mode Control Register
  TIM2->DIER = 0x01; //Interrupt Enable Register: Update interrupt enabled
  TIM2->PSC = PreScaleValue;    //Prescaler
  TIM2->ARR = TimingValue; //Auto-reload register
  TIM2->EGR = TIM2->EGR | 0x01;   // set UG bit to force an event to load the new ARR value - will also reset counter value to zero (for up-counter)
}

/*******************************************************************************
* Function Name  : TIMER3Init.
* Description    : Configures the TIMER3.
* Input          : TimingValue (u16), PreScaleValue (u16)
* Output         : None.
* Return         : None.
*******************************************************************************/
void Timer3Init(u16 TimingValue, u16 PreScaleValue)
{
  //TIM3->CR1 = 0x8D;  //ControlRegister 1: Auto-Reload Preload enabled, Counter used as upcounter, One Pulse mode, Only interrupt on overflow,  Start counter.
  TIM3->CR1 = 0x80;  //ControlRegister 1: Auto-Reload Preload enabled, Counter used as upcounter, Stop counter.
  TIM3->CR2 = 0x00;  //ControlRegister 2
  TIM3->SMCR = 0x00; //Slave Mode Control Register
  TIM3->DIER = 0x01; //Interrupt Enable Register: Update interrupt enabled
  TIM3->PSC = PreScaleValue;    //Prescaler
  TIM3->ARR = TimingValue; //Auto-reload register
  TIM3->EGR = TIM3->EGR | 0x01;   // set UG bit to force an event to load the new ARR value - will also reset counter value to zero (for up-counter)
}

void Timer4Init(u16 TimingValue)
```

```
{
  //TIM4->CR1 = 0x8D;  //ControlRegister 1: Auto-Reload Preload enabled, Counter used as upcounter, One Pulse mode, Only interrupt on overflow,  Start counter.
  TIM4->CR1 = 0x80;  //ControlRegister 1: Auto-Reload Preload enabled, Counter used as upcounter, Stop counter.
  TIM4->CR2 = 0x00;  //ControlRegister 2
  TIM4->SMCR = 0x00; //Slave Mode Control Register
  TIM4->DIER = 0x01; //Interrupt Enable Register: Update interrupt enabled
  TIM4->PSC = 7200;    //Prescaler
  TIM4->ARR = TimingValue; //Auto-reload register
  TIM4->EGR = TIM4->EGR | 0x01;   // set UG bit to force an event to load the new ARR value - will also reset counter value to zero (for up-counter)
}

//////// Streaming function ////////////
/*****************************************************************************/
uint8_t Stream127(uint8_t *buf)
/*****************************************************************************/
{
    uint8_t   CurrentState;
    uint8_t   PreviousState;
    uint8_t   ReadByte;
    uint8_t   BitCount;
    uint8_t   ByteCount;
    uint8_t   EdgeCount;

    uint16_t  BitPeriod;
    uint16_t  Timer2CountResult;

  *(buf++) = 0x00; //first byte is reserved for error message

  //Set up TIMER2 for measuring sync byte 0xAA (TIMER2 preconfigured for 250ns resolution up-counter)
  TIM2->ARR = (40000);           // reload value * 250ns (=10ms)
  TIM2->EGR = TIM2->EGR | 0x01;   // set UG bit to force an event to load the new ARR value and reset counter to zero
  TIM2->SR = TIM2->SR & 0xFE;     // reset update interrupt flag
  Timer2Expired = 0;

  //Set up timer4 for time-out operation (Timer4 preconfigured for 100us resolution up-counter)
  TIM4->ARR = (1000);            // reload value * 100us (=100ms)
  TIM4->EGR = TIM4->EGR | 0x01;   // set UG bit to force an event to load the new ARR value and reset counter to zero
  TIM4->SR = TIM4->SR & 0xFE;     // reset update interrupt flag
```

```
Timer4Expired = 0;

//(1)
//Wait for falling edge of start condition on TOUT/OUT
TIM4->CR1 = TIM4->CR1 | 0x01;   // start counter4
while (GPIO_ReadInputDataBit(P3_127_PORT, P3_127_PIN) && !Timer4Expired);
//start counter2 as soon as falling edge is detected
TIM2->CR1 = TIM2->CR1 | 0x01;   // start counter2

//(2)
//Wait for 8 edges
PreviousState = GPIO_ReadInputDataBit(P3_127_PORT, P3_127_PIN);
for(EdgeCount = 0 ; EdgeCount < 8 ; EdgeCount++)
{
  //check what the current state is
  while((CurrentState == PreviousState) && !Timer4Expired)
  {
    CurrentState = GPIO_ReadInputDataBit(P3_127_PORT, P3_127_PIN); //TOUT/OUT
  }
  PreviousState = CurrentState;
}

//(3)
//Do calculations
TIM2->CR1 = TIM2->CR1 & 0xFE;   // stop TIMER2 after last edge

Timer2CountResult = TIM2->CNT;  // get TIMER2 value
*(buf++) = 0xAA; //store sync byte in buffer
BitPeriod = (Timer2CountResult / 8) + 1; // add one to compensate for round-down errors

//(4)
while (!GPIO_ReadInputDataBit(P3_127_PORT, P3_127_PIN) && !Timer4Expired); //wait for rising edge of stop condition

//(5)
//Set up TIMER3 for BitPeriod delay (TIMER3 preconfigured for 250ns resolution up-counter)
TIM3->ARR = (BitPeriod);        // reload value * 250ns
TIM3->EGR = TIM3->EGR | 0x01;   // set UG bit to force an event to load the new ARR value and reset TIMER3 to zero
TIM3->SR = TIM3->SR & 0xFE;     // reset update interrupt flag
```

```
Timer3Expired = 0;
//(6)
//Start getting the data
for(ByteCount = 0 ; ByteCount < 8;ByteCount++)
{
  ReadByte = 0x00;

  while (GPIO_ReadInputDataBit(P3_127_PORT, P3_127_PIN) && !Timer4Expired); //wait for falling edge of start bit on TOUT/OUT
  TIM3->CR1 = TIM3->CR1 | 0x01;   // start TIMER3

  for(BitCount = 0 ; BitCount < 8; BitCount++)
  {
    Timer3Expired = 0;
    ReadByte <<= 1;
    while (!Timer3Expired); //wait for a full bit period to get to data bit
    if (GPIO_ReadInputDataBit(P3_127_PORT, P3_127_PIN)) //TOUT/OUT
    {
      ReadByte |= 0x01;
    }
  }

  //(7)
  //Store Byte data
  TIM3->CR1 = TIM3->CR1 & 0xFE;   // stop TIMER3
  TIM3->EGR = TIM3->EGR | 0x01;   // set UG bit to force an event to load the new ARR value and reset TIMER3 to zero
  TIM3->SR = TIM3->SR & 0xFE;     // reset update interrupt flag

  *(buf++) = ReadByte;    //store byte in buffer
  while (!GPIO_ReadInputDataBit(P3_127_PORT, P3_127_PIN) && !Timer4Expired); //wait for rising edge of stop condition (if last bit was low)
  //(8)
}

TIM2->CR1 = TIM2->CR1 & 0xFE;   // stop TIMER2
TIM3->CR1 = TIM3->CR1 & 0xFE;   // stop TIMER3
TIM4->CR1 = TIM4->CR1 & 0xFE;   // stop TIMER24

if (Timer2Expired || Timer4Expired)
{
```

```
  return ERR_DEVICE_NOT_READY;
}

*(buf++) = 0xDB;   //for DeBug
*(buf++) = Timer2CountResult >> 8;  //MSB timer value for debugging
*(buf++) = Timer2CountResult;       //LSB timer value for debugging

return 0x00;
}/* Stream127 end*/
```