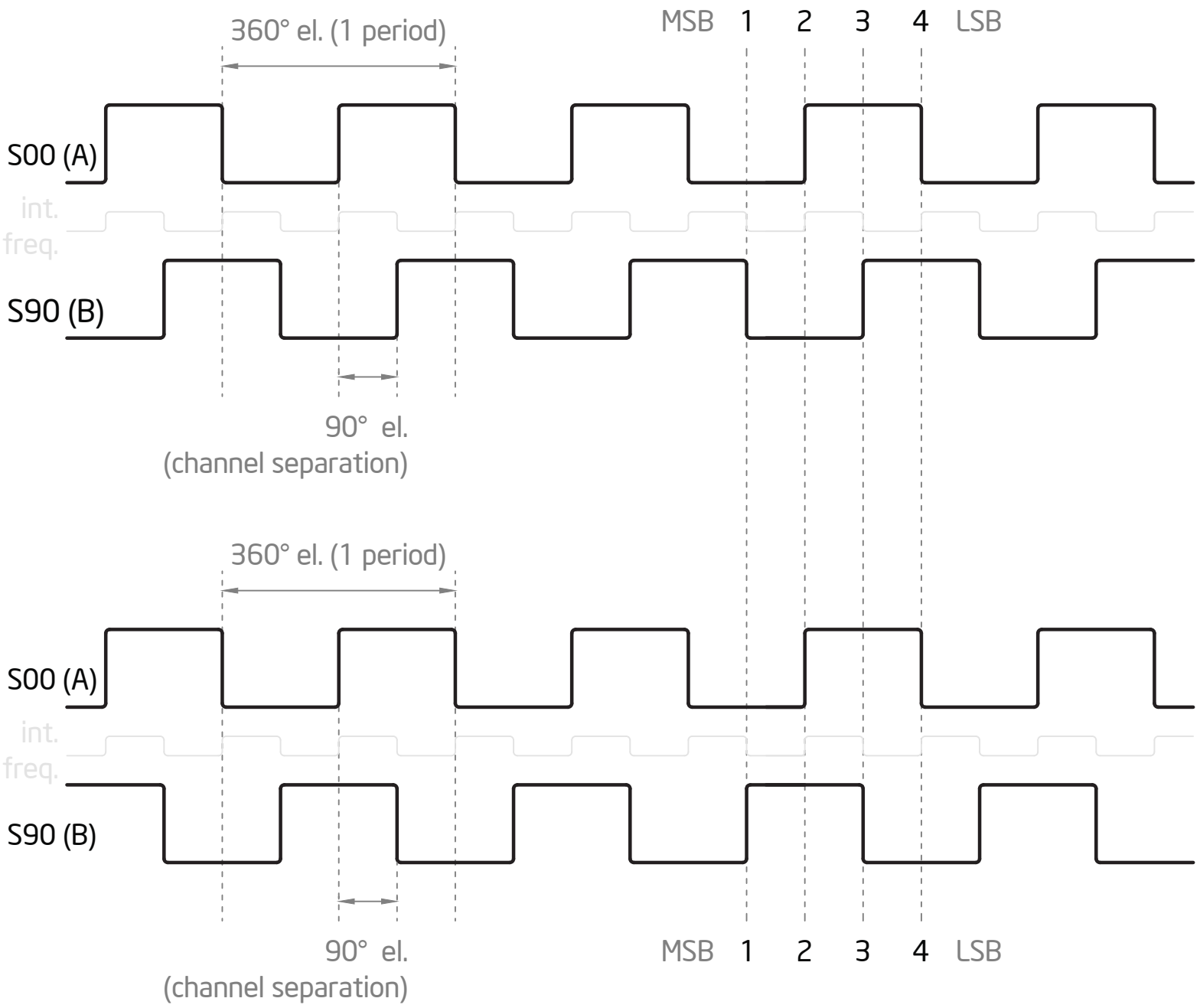
version 1.2

Set of pulse decoding algorithms for quadrature rotary and linear encoders*

(*) Algorithms are likely platform nonindependent in performance comparison. However results are based to the Atmel AVR 8-bit core architecture running at 20 MHz.

Ideal signal output of the quadrature encoder



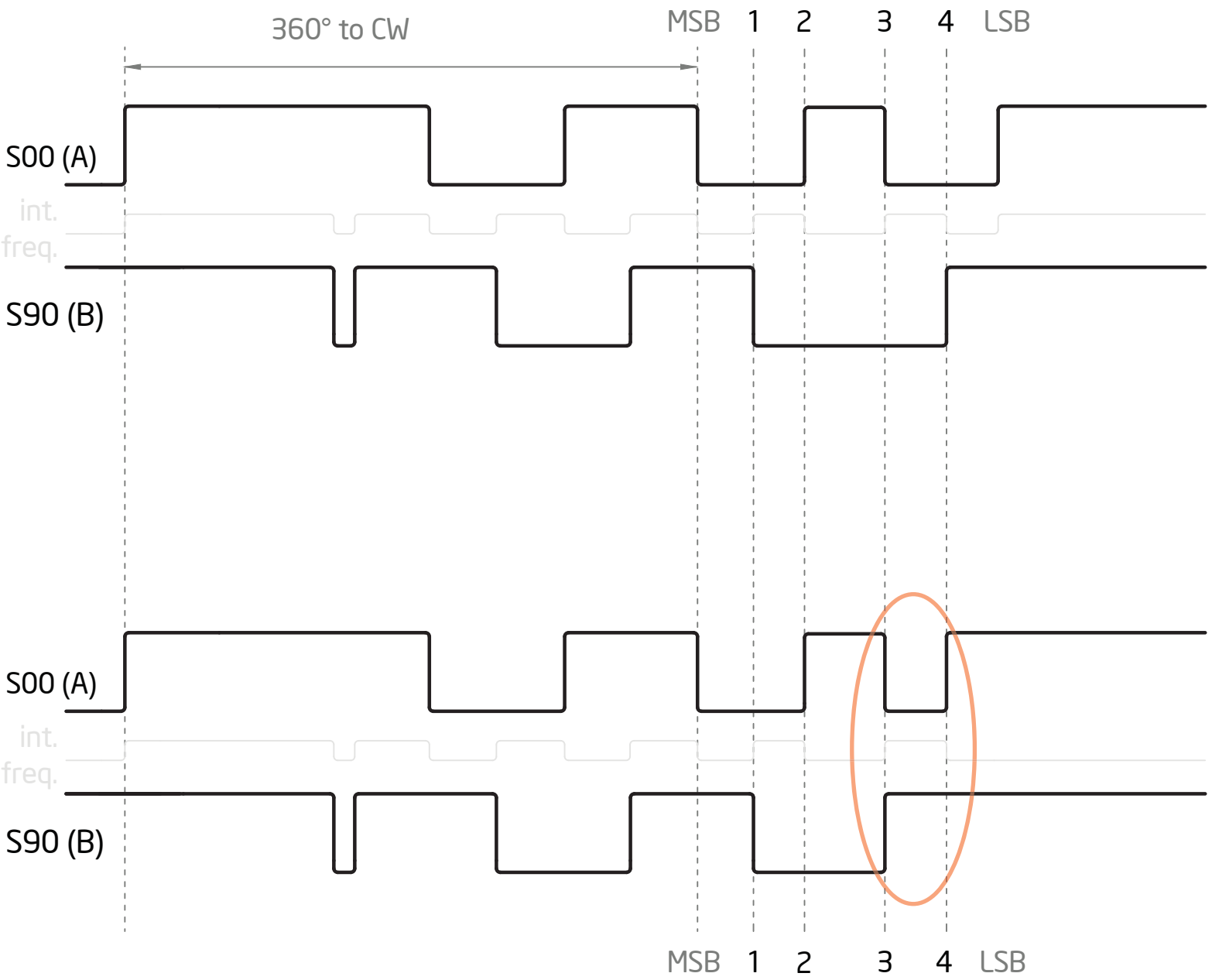
Clockwise,
seen from shaft side

Phase	S00 (0x06)	S90 (0x03)	Output
1	0	0	0
2	1	0	2
3	1	1	3
4	0	1	1
	0110	0011	

Counterclockwise,
seen from shaft side

Phase	S00 (0x06)	S90 (0x0C)	Output
1	0	1	1
2	1	1	3
3	1	0	2
4	0	0	0
	0110	1100	

Non ideal signal output of the quadrature encoder



Random direction without errors

Phase	S00 (0x06)	S90 (0x03)	Output
1	0	0	0
2	1	0	2
3	0	0	0
4	0	1	1
	0100	0001	

Random direction with errors

Phase	S00 (0x05)	S90 (0x03)	Output
1	0	0	1
2	1	0	2
3	0	1	1
4	1	1	3
	0101	0011	

Lookup table values

Lookup table of 256 bytes will contain 16 valid pulse sequences as shown below.
No further information available at this moment how the data generation of the tables has been made.

Coding for clockwise rotation (right)

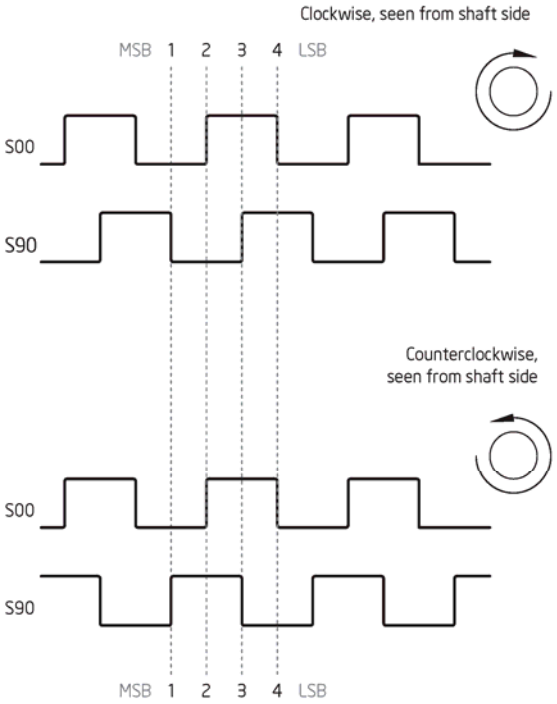
Phase	S00 (06h)	S90 (03h)	Output (dec)
1	0	0	0
2	1	0	2
3	1	1	3
4	0	1	1
	0110	0011	

Lookup index for 0x63 is 0x2D or 00101101. Pulses +1

Coding for counter-clockwise rotation (left)

Phase	S00 (06h)	S90 (0Ch)	Output (dec)
1	0	1	1
2	1	1	3
3	1	0	2
4	0	0	0
	0110	1100	

Lookup index for 0x6C is 0x78 or 01111000. Pulses -1




```
input = (input << 1) | S00;
input = (input << 1) | S90;
Pulses += LookUp[input]
if (output == 0) pulse_errors++;
```

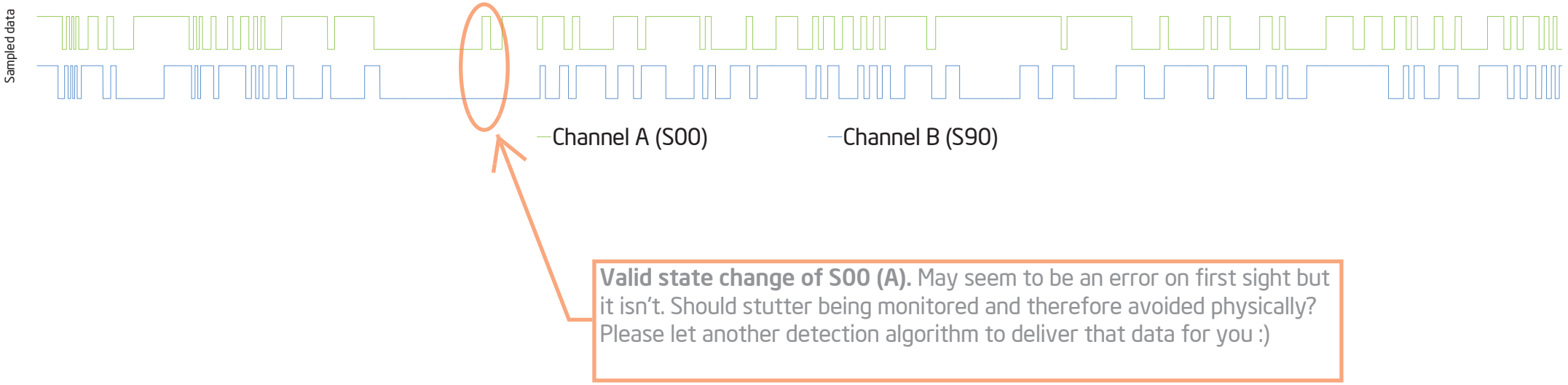
	CCW (left) -1	CW (right) +1	
1	0x88	0x22	
2	0x21	0x8B	
3	0x2E	0x84	
4	**0x87**	**0x2D**	← *4. seq.*
5	0xB8	0x12	
6	0x11	0xBB	
7	**0x1E**	**0xB4**	← *1. seq.*
8	0xB7	0x1D	
9	0x48	0xE2	
10	**0xE1**	**0x4B**	← *3. seq.*
11	0xEE	0x44	
12	0x47	0xED	
13	**0x78**	**0xD2**	← *2. seq.*
14	0xD1	0x7B	
15	0xDE	0x74	
16	0x77	0xDD	

	CCW (pulses -1)	CW (pulses +1)	
1	10001000	00100010	
2	00100001	10001011	
3	00101110	10000100	
4	**10000111**	**00101101**	← *4. seq.*
5	10111000	00010010	
6	00010001	10111011	
7	**00011110**	**10110100**	← *1. seq.*
8	10110111	00011101	
9	01001000	11100010	
10	**11100001**	**01001011**	← *3. seq.*
11	11101110	01000100	
12	01000111	11101101	
13	**01111000**	**11010010**	← *2. seq.*
14	11010001	01111011	
15	11011110	01110100	
16	01110111	11011101	

⚠ Other values will output a zero value for error counter.

Lookup table is filled only with three different value types. 0x01 will increase while 0xFF (-1) will decrease the pulse counter. 0x00 is optimally reserved for error detection because it doesn't affect to the pulse counting. This featrure might be usefull in some applications. For correct output value, proper indexing must be done with above sequencies as in further example codes.

Sample of quadrature sensor data from real life. Persistent stutter (minor direction changes) with variable rotation time will cause significant deformations to the ideal signal. Debug is far more complicated.



Poor man's method with resolution of 1X. Decoding frequency is most efficient, but it does not handle stuttering well as it can skip rapid rotation or displacement changes very easily in real life applications. Minimum response time is 1,32 µs or 758 kHz with maximum CPS (counts per second) of 189 k.

```
// External Interrupt 0 service routine for S00 (rising edge, resolution 1X)
interrupt [EXT_INT0] void ext_int0_isr(void)
{
   if (S90 == 0) pulses++;
   else pulses--;
}
```

More sophisticated method with resolution of 2X. It's theoretically affected by varying execution time being still faster than rest of the algorithms, but with reduced accuracy. Routine is anyhow unable to provide any error detection. Minimum response time 2,33 µs or 429 kHz with maximum CPS of 215 k.

```
// External Interrupt 0 service routine for S00 (both edges, resolution 2X)
interrupt [EXT_INT0] void ext_int0_isr(void)
{
   if (S00 == 1)
   {
      if (S90 == 0) pulses++;
      else pulses--;
   }
   else
   {
      if (S90 == 0) pulses--;
      else pulses++;
   }
}
```

Algorithm with constant execution time. Slightly slower than the routine code above, but it's capable of decent error detection and performance. Minimum response time 3,00 µs or 333 kHz with maximum CPS of 167 k.

```
const char LookupTable[256] = {
                0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0xFF,
                0x00, 0x00, 0x01, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                0x00, 0x00, 0x01, 0xFF, 0x00, 0x00, 0x01, 0xFF, 0x00, 0x00, 0x00, 0x00,
                0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                0x00, 0x00, 0x01, 0xFF, 0x00, 0x00, 0x01, 0xFF, 0x00, 0x00, 0x00, 0x00,
                0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0xFF, 0x00, 0x00, 0x01, 0xFF,
                0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                0x00, 0x00, 0x00, 0x00, 0xFF, 0x01, 0x00, 0x00, 0xFF, 0x01, 0x00, 0x00,
                0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0x01, 0x00, 0x00,
                0xFF, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0x01, 0x00, 0x00,
                0xFF, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                0xFF, 0x01, 0x00, 0x00, 0xFF, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                0x00, 0x00, 0x00, 0x00
};

// External Interrupt 0 service routine on both edges with 2X resolution
interrupt [EXT_INT0] void ext_int0_isr(void)
{
   input = (input << 1) | S00;
   input = (input << 1) | S90;
   buffer = LookupTable[input];
   pulses += buffer;
   if (buffer == 0) pulse_errors++;
}

void reset_pulses()
{
   // Pulse sensor initialization
   if ((S00 == 0) && (S90 == 0)) input = 0xCC;
   if ((S00 == 1) && (S90 == 0)) input = 0x33;
   if ((S00 == 1) && (S90 == 1)) input = 0x33;
   if ((S00 == 0) && (S90 == 1)) input = 0xCC;
   pulse_errors = 0;
   pulses = 0;
}
```

Quadrature decoding algorithm with resolution of 4X. Capable of detecting false pulse sequences efficiently with improved accuracy. Precision and error detection modifications causes no significant responsiveness loss. Two interrupts required! Minimum response time 5,88 μs or 170 kHz with maximum CPS of 170 k.

```
const char LookupTable[256] = {
                0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00,
                0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0xFF, 0x00, 0x00, 0xFF, 0x01, 0x00,
                0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0xFF, 0x00,
                0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0xFF,
                0xFF, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0xFF,
                0xFF, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                0x01, 0x00, 0x00, 0xFF, 0xFF, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x00,
                0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                0x01, 0x00, 0x00, 0xFF, 0xFF, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x00,
                0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00,
                0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0xFF, 0x00, 0x00, 0xFF, 0x01, 0x00,
                0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0xFF, 0x00,
                0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                0x00, 0x00, 0x00, 0x00
};

// External Interrupt 0 service routine on both edges with 4X resolution
interrupt [EXT_INT0] void ext_int0_isr(void)
{
   input = (input << 1) | S00;
   input = (input << 1) | S90;
   output = LookupTable[input];
   pulses += output;
   if (output == 0) pulse_errors++;
}

// External Interrupt 1 service routine on both edges with 4X resolution
interrupt [EXT_INT1] void ext_int1_isr(void)
{
   input = (input << 1) | S00;
   input = (input << 1) | S90;
   output = LookupTable[input];
   pulses += output;
   if (output == 0) pulse_errors++;
}

void reset_pulses()
{
   // Pulse sensor initialization
   if ((S00 == 0) && (S90 == 0)) input = 0x78;
   if ((S00 == 1) && (S90 == 0)) input = 0x1E;
   if ((S00 == 1) && (S90 == 1)) input = 0x87;
   if ((S00 == 0) && (S90 == 1)) input = 0xE1;
   pulse_errors = 0;
   pulses = 0;
}
```

Quadrature decoding algorithm with resolution of 4X. Optimized AVR (MCU 8-bit) assembly code can process error and non-erroneous signals at near symmetric execution time. This with ASM and lookup table programming will lead to the minimal code overhead of error detection. This among other optimizations can be achieved by low-level programming which will increase the total efficiency of algorithm near 30% compared to the C-code. Minimum response time is 4,54 µs or 220 kHz with maximum CPS of 220 k.

```
#pragma keep+
flash const char LookupTable[256] = {
                0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00,
                0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0xFF, 0x00, 0x00, 0xFF, 0x01, 0x00,
                0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0xFF, 0x00,
                0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0xFF,
                0xFF, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0xFF,
                0xFF, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                0x01, 0x00, 0x00, 0xFF, 0xFF, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x00,
                0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                0x01, 0x00, 0x00, 0xFF, 0xFF, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x00,
                0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00,
                0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0xFF, 0x00, 0x00, 0xFF, 0x01, 0x00,
                0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0xFF, 0x00,
                0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                0x00, 0x00, 0x00, 0x00
};
#pragma keep-

// Continues on the next page...
```

```
// Code below is same for both interrupts (it may be necessary to change the jump address names independently)
#asm
    PUSH  R16
    PUSH  R17
    PUSH  R18
    PUSH  R19
    PUSH  R22
    PUSH  R24
    PUSH  R30
    IN    R30, SREG
    PUSH  R30
    PUSH  R31
    IN    R31, 0x10
    LSR   R31
    LSR   R31
    LSR   R31
    ROL   _input
    LSR   R31
    ROL   _input
    MOV   R30, _input
    EOR   R31, R31
    SUBI  R30, LOW(-_LookupTable << 1)
    SBCI  R31, HIGH(-_LookupTable << 1)
    LPM   R22, Z
    CPI   R22, 0
    BREQ  Error
    LDS   R16, _pulses
    LDS   R17, _pulses+1
    LDS   R18, _pulses+2
    LDS   R19, _pulses+3
    MOV   R24, R22
    ADD   R24, R24
    SBC   R24, R24
    ADD   R16, R22
    ADC   R17, R24
    ADC   R18, R24
    ADC   R19, R24
    STS   _pulses, R16
    STS   _pulses+1, R17
    STS   _pulses+2, R18
    STS   _pulses+3, R19
    JMP   Exit
Error:
    LDS   R16, _pulse_errors
    LDS   R17, _pulse_errors+1
    LDS   R18, _pulse_errors+2
    LDS   R19, _pulse_errors+3
    LDI   R24, 1
    ADD   R16, R24
    LDI   R24, 0
    ADC   R17, R24
    ADC   R18, R24
    ADC   R19, R24
    STS   _pulse_errors, R16
    STS   _pulse_errors+1, R17
    STS   _pulse_errors+2, R18
    STS   _pulse_errors+3, R19
Exit:
    POP   R31
    POP   R30
    OUT   SREG, R30
    POP   R30
    POP   R24
    POP   R22
    POP   R19
    POP   R18
    POP   R17
    POP   R16
#endasm
```

Method efficiency comparison

Routine/algorithm	Minimum response time (µs) @ 20 MHz	Max. quadrature frequency (kHz)	Accuracy multiplier	Theoretical max. CPS
1X IF with no error detection	1,32	758	1	189394
2X IF with no error detection	2,33	429	2	214592
2X lookup with mediocre error detection	3,00	333	2	166667
2X ASM lookup with decent error detection	2,33	429	2	214592
4X IF with no error detection	n/a	n/a	4	n/a
4X lookup with enhanced error detection	5,88	170	4	170068
4X ASM lookup with enhanced error detection	4,54	220	4	**220264**

All code without ASM keyword is written with embedded C-language. Source codes were compiled by CodeVisionAVR 2.60. Overall results may vary, especially when there is no ASM optimized competitor for the time critacal C-code segment for routines above (1X IF, 2X IF & 4X IF).

Sample code below from proto device.
Software seems to be quite stable.

```
/*****************************************************
This program was produced by the
CodeWizardAVR V2.60 Standard
Automatic Program Generator
© Copyright 1998-2012 Pavel Haiduc, HP InfoTech s.r.l.
http://www.hpinfotech.com

Project                 : Meter counter/Leine Linde RS501 (incremental rotary encoder)
Version                 : 120916
Date                    : 19.2.2012
Author                  : Klaus Varis
Company                 : SMOY
Chip type               : ATmega32 (8-bit)
Program type            : Proto application
AVR clock frequency     : 20,000 MHz (Overclocked +4 MHz)
Memory model            : Small
External RAM size       : 0
Data Stack size         : 512
Maximum detection speed : 220/4 kHz (4,54 µs)
Maximun RPM             : 6600
Maximum Propagation     : 33,35 m/s or 120 km/h
*****************************************************/

#include <mega32.h>
#include <alcd.h>
#include <stdio.h>
#include <delay.h>
#define S00 PIND.2
#define S90 PIND.3
#define RESET PINB.2

// Global routine variables
long int pulses = 0;
register unsigned char input;
unsigned long int pulse_errors = 0;
unsigned char clear_lcd_once = 1;
const unsigned long int ERROR_THRESHOLD = 50; // Displays error count after threshold
const float calibration_pulses = 659636;
const float calibrated_meters = 100.0;

#pragma keep+
flash char LookupTable[256] =
{
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0x01, 0x00, 0x00, 0x01, 0xFF, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0x01, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0x00, 0x00, 0x01,
        0x01, 0x00, 0x00, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0x00, 0x00, 0x01,
        0x01, 0x00, 0x00, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0xFF, 0x00, 0x00, 0x01, 0x01, 0x00, 0x00, 0xFF, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0xFF, 0x00, 0x00, 0x01, 0x01, 0x00, 0x00, 0xFF, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0x01, 0x00, 0x00, 0x01, 0xFF, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0x01, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00
}; // Inverted lookup table for positive counting meters (0xFF = 0x01 and 0x01 = 0xFF)
#pragma keep-

void reset_pulses();
void init();

// Declare your global variables here
char lcd_buf[80];
```

```
// External Interrupt 0 service routine on both edges
interrupt [EXT_INT0] void ext_int0_isr(void)
{
#asm
        PUSH    R16
        PUSH    R17
        PUSH    R18
        PUSH    R19
        PUSH    R22
        PUSH    R24
        PUSH    R30
        IN      R30, SREG
        PUSH    R30
        PUSH    R31
        IN      R31, 0x10
        LSR     R31
        LSR     R31
        LSR     R31
        ROL     _input
        LSR     R31
        ROL     _input
        MOV     R30, _input
        EOR     R31, R31
        SUBI    R30, LOW(-_LookupTable << 1)
        SBCI    R31, HIGH(-_LookupTable << 1)
        LPM     R22, Z
        CPI     R22, 0
        BREQ    Error_A
        LDS     R16, _pulses
        LDS     R17, _pulses+1
        LDS     R18, _pulses+2
        LDS     R19, _pulses+3
        MOV     R24, R22
        ADD     R24, R24
        SBC     R24, R24
        ADD     R16, R22
        ADC     R17, R24
        ADC     R18, R24
        ADC     R19, R24
        STS     _pulses, R16
        STS     _pulses+1, R17
        STS     _pulses+2, R18
        STS     _pulses+3, R19
        JMP     Exit_A
Error_A:
        LDS     R16, _pulse_errors
        LDS     R17, _pulse_errors+1
        LDS     R18, _pulse_errors+2
        LDS     R19, _pulse_errors+3
        LDI     R24, 1
        ADD     R16, R24
        LDI     R24, 0
        ADC     R17, R24
        ADC     R18, R24
        ADC     R19, R24
        STS     _pulse_errors, R16
        STS     _pulse_errors+1, R17
        STS     _pulse_errors+2, R18
        STS     _pulse_errors+3, R19
Exit_A:
        POP     R31
        POP     R30
        OUT     SREG, R30
        POP     R30
        POP     R24
        POP     R22
        POP     R19
        POP     R18
        POP     R17
        POP     R16
#endasm
}
```

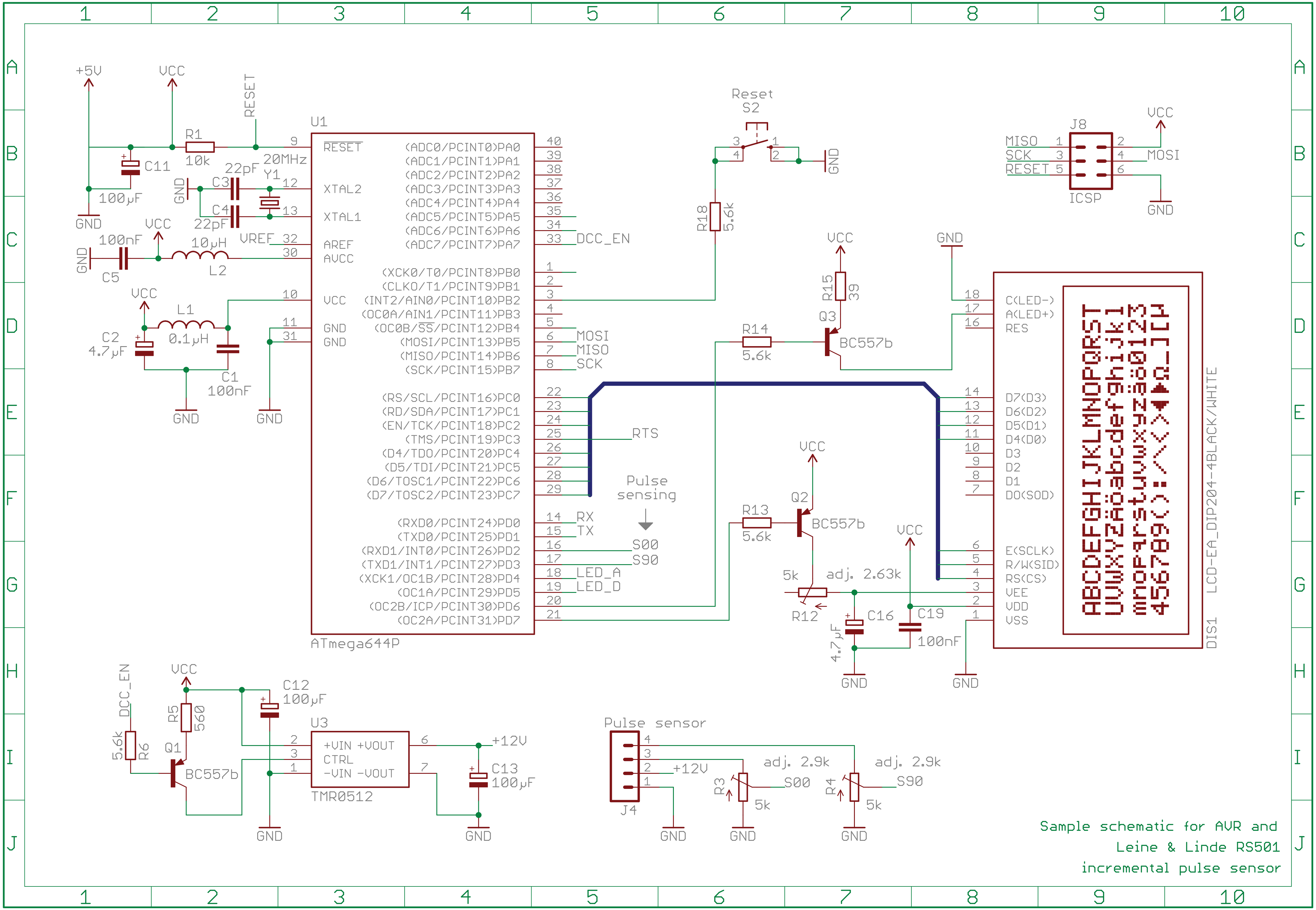
```
// External Interrupt 1 service routine on both edges
interrupt [EXT_INT1] void ext_int1_isr(void)
{
#asm
        PUSH    R16
        PUSH    R17
        PUSH    R18
        PUSH    R19
        PUSH    R22
        PUSH    R24
        PUSH    R30
        IN      R30, SREG
        PUSH    R30
        PUSH    R31
        IN      R31, 0x10
        LSR     R31
        LSR     R31
        LSR     R31
        ROL     _input
        LSR     R31
        ROL     _input
        MOV     R30, _input
        EOR     R31, R31
        SUBI    R30, LOW(-_LookupTable << 1)
        SBCI    R31, HIGH(-_LookupTable << 1)
        LPM     R22, Z
        CPI     R22, 0
        BREQ    Error_B
        LDS     R16, _pulses
        LDS     R17, _pulses+1
        LDS     R18, _pulses+2
        LDS     R19, _pulses+3
        MOV     R24, R22
        ADD     R24, R24
        SBC     R24, R24
        ADD     R16, R22
        ADC     R17, R24
        ADC     R18, R24
        ADC     R19, R24
        STS     _pulses, R16
        STS     _pulses+1, R17
        STS     _pulses+2, R18
        STS     _pulses+3, R19
        JMP     Exit_B
Error_B:
        LDS     R16, _pulse_errors
        LDS     R17, _pulse_errors+1
        LDS     R18, _pulse_errors+2
        LDS     R19, _pulse_errors+3
        LDI     R24, 1
        ADD     R16, R24
        LDI     R24, 0
        ADC     R17, R24
        ADC     R18, R24
        ADC     R19, R24
        STS     _pulse_errors, R16
        STS     _pulse_errors+1, R17
        STS     _pulse_errors+2, R18
        STS     _pulse_errors+3, R19
Exit_B:
        POP     R31
        POP     R30
        OUT     SREG, R30
        POP     R30
        POP     R24
        POP     R22
        POP     R19
        POP     R18
        POP     R17
        POP     R16
#endasm
}
```

```
// External Interrupt 2 service routine on both edges
interrupt [EXT_INT2] void ext_int2_isr(void)
{
	unsigned int cnt = 0;
	GICR &= 0b11011111; // Disable EXT_INT2
	lcd_init(20);
	lcd_putsf("Resetting...\n\nDo not move\nthe encoder!");
	while (RESET == 0)
	{
		delay_ms(1);
		cnt++;
		if (cnt > 2000)
		{
			lcd_clear();
			lcd_putsf("\nHardware reset!\nPlease wait...");
			while (RESET == 0) delay_ms(1);
			lcd_clear();
			delay_ms(1000);
			GIFR |= 0b00100000; // Clear pending EXT_INT2 interrupts caused by key debounce
			WDTCR = 0x18;  // Sophisticated reset with Watch Dog Timer (WDT).
			WDTCR = 0x08;  // In some situations an ordinary JMP 0x0000 may not be enough.
			while(1);  // Wait until WDT resets the chip.
		}
	}
	#asm("sei");
	do
	{
		reset_pulses();
		for (cnt = 0; cnt < 3500; cnt++)
		{
			lcd_gotoxy(19,3);
			if (pulses != 0)
			{
				lcd_putchar(255);
				delay_us(50);
			}
			else
			{
				lcd_putchar(' ');
				delay_us(50);
			}
		}
	}
	while(pulses != 0);
	reset_pulses();
	lcd_init(20);
	lcd_clear();
	for (cnt = 0; cnt < 80; cnt++) lcd_buf[cnt] = 0; // Fixes random lcd_clear issues with undesired character
	//messy.
	GIFR |= 0b00100000; // Clear pending EXT_INT2 interrupts caused by key de-bounce
	GICR |= 0b00100000; // Re-enable EXT_INT2
}

void main(void)
{
	unsigned int cnt = 0;
	init();
	while ((RESET == 0) & (cnt != 2000))
	{
		cnt++;
		delay_ms(1);
	}
	if (cnt >= 2000)
	{
		sprintf(lcd_buf, " Calibration data\nMeters : %10.3f\nPulses : %10.0f\nMaximum : %2.2f
		m/s",calibrated_meters, calibration_pulses, (calibrated_meters/calibration_pulses)*220000);
		lcd_puts(lcd_buf);
		while (1);
	}
```

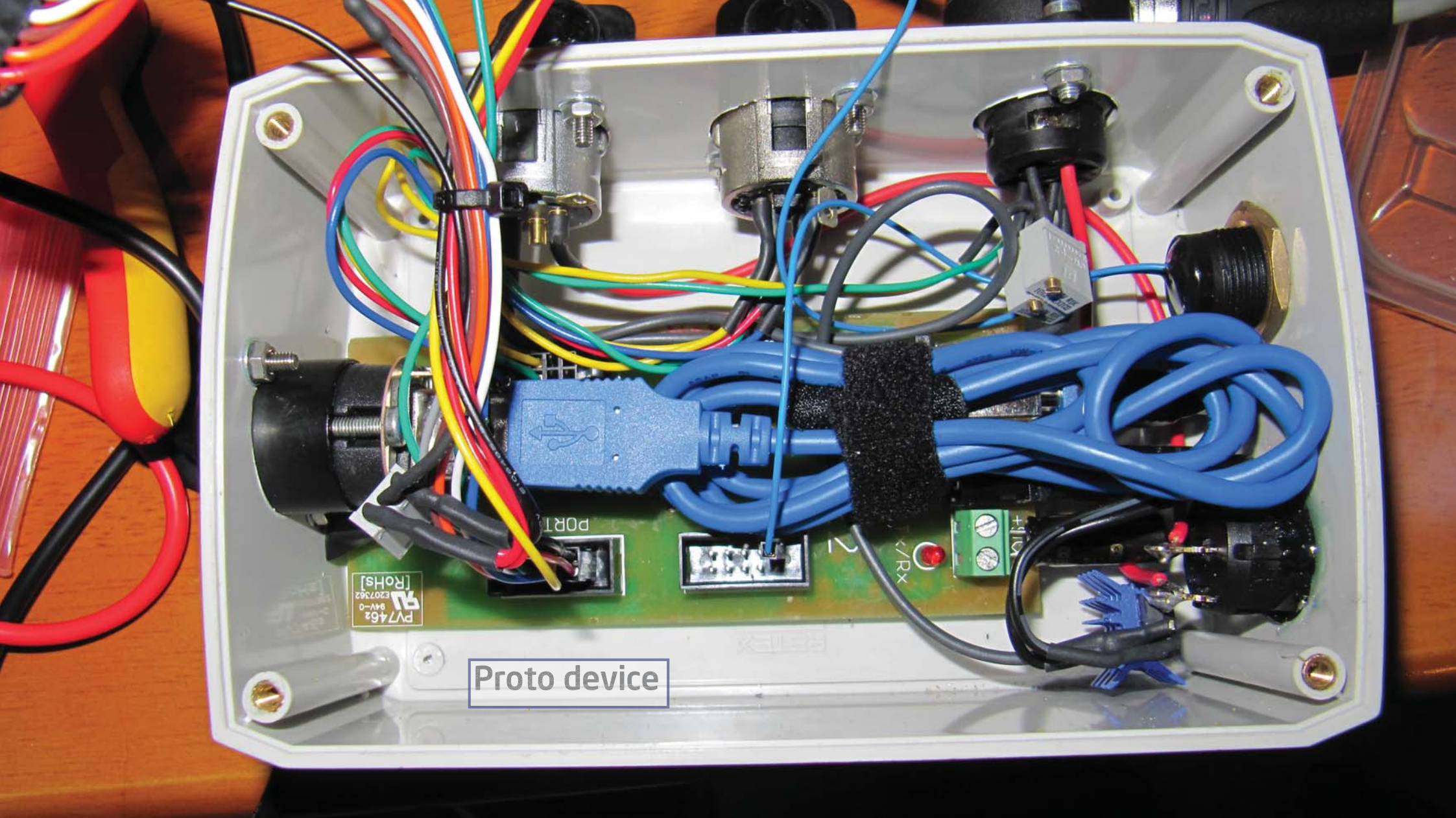
```
	if (RESET == 1)
	{

		lcd_putsf("\n  DrillCon\n SMOY");
		do
		{
			delay_ms(1);
			cnt++;
			if (RESET == 0) goto SKIP;
		}
		while (cnt != 3750);
		cnt = 0;
		lcd_clear();
		lcd_putsf("\n Software ver.\n  120916");
		do
		{
			delay_ms(1);
			cnt++;
			if (RESET == 0) goto SKIP;
		}
		while (cnt != 3750);
		cnt = 0;
		lcd_clear();
		lcd_putsf("Precise reading of\nQuadrature incremen.\nrotary encoder like\nLeine & Linde RS-501");
		do
		{
			delay_ms(1);
			cnt++;
			if (RESET == 0) goto SKIP;
		}
		while (cnt != 3750);
		cnt = 0;
	}
SKIP:
	while(RESET == 0) delay_ms(1);
	lcd_clear();
	GIFR |= 0b00100000; // Clear pending EXT_INT2 interrupts
	#asm("sei") // Global enable interrupts
	reset_pulses();
	while (1)
	{
		if (pulse_errors > ERROR_TRESHOLD)
		{
			if (clear_lcd_once == 1)
			{
				clear_lcd_once = 0;
				GICR &= 0b11011111; // Disable EXT_INT2
				lcd_clear();
				GICR |= 0b00100000; // Re-enable EXT_INT2
			}
			sprintf(lcd_buf, " Depth in meters\n [%8.3f]\n\n Errors: %lu",
			(calibrated_meters/calibration_pulses)*pulses, pulse_errors);
		}
		else sprintf(lcd_buf, "\n Depth in meters\n [%8.3f]", (calibrated_meters/calibration_pulses)*pulses);
		GICR &= 0b11011111; // Disable EXT_INT2, fixes random lcd_clear issues with undesired
		character messy.
		lcd_gotoxy(0, 0);
		lcd_puts(lcd_buf);
		GICR |= 0b00100000; // Re-enable EXT_INT2
	}
}

void reset_pulses()
{
	// Pulse sensor initialization
	if ((S00 == 0) && (S90 == 0)) input = 0x78;
	if ((S00 == 1) && (S90 == 0)) input = 0x1E;
	if ((S00 == 1) && (S90 == 1)) input = 0x87;
	if ((S00 == 0) && (S90 == 1)) input = 0xE1;
	pulse_errors = 0;
	pulses = 0;
	clear_lcd_once = 1;
}
```

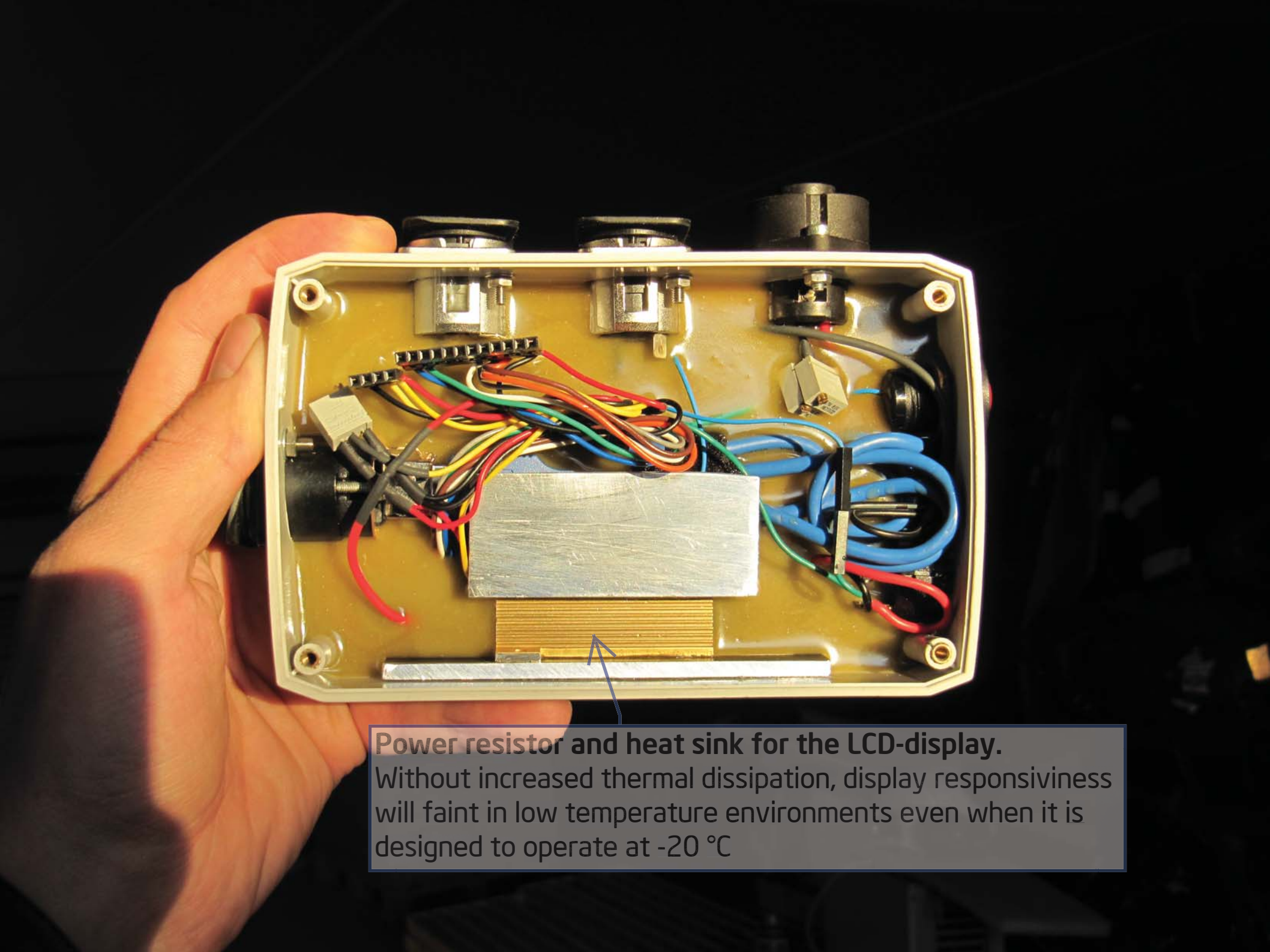
```
void init()
{
        unsigned char temp;
        // Input/Output Ports initialization
        // Port A initialization
        // Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
        // State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
        PORTA=0x00;
        DDRA=0x00;

        // Port B initialization
        // Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
        // State7=T State6=T State5=T State4=T State3=T State2=P State1=T State0=T
        PORTB=0x04;
        DDRB=0x00;

        // Port C initialization
        // Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
        // State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
        PORTC=0x00;
        DDRC=0x00;

        // Port D initialization
        // Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
        // State7=T State6=T State5=T State4=T State3=P State2=P State1=T State0=T
        PORTD=0x0C;
        DDRD=0x00;

        // Timer/Counter 0 initialization
        // Clock source: System Clock
        // Clock value: Timer 0 Stopped
        // Mode: Normal top=0xFF
        // OC0 output: Disconnected
        TCCR0=0x00;
        TCNT0=0x00;
        OCR0=0x00;

        // Timer/Counter 1 initialization
        // Clock source: System Clock
        // Clock value: Timer1 Stopped
        // Mode: Normal top=0xFFFF
        // OC1A output: Discon.
        // OC1B output: Discon.
        // Noise Canceler: Off
        // Input Capture on Falling Edge
        // Timer1 Overflow Interrupt: Off
        // Input Capture Interrupt: Off
        // Compare A Match Interrupt: Off
        // Compare B Match Interrupt: Off
        TCCR1A=0x00;
        TCCR1B=0x00;
        TCNT1H=0x00;
        TCNT1L=0x00;
        ICR1H=0x00;
        ICR1L=0x00;
        OCR1AH=0x00;
        OCR1AL=0x00;
        OCR1BH=0x00;
        OCR1BL=0x00;

        // Timer/Counter 2 initialization
        // Clock source: System Clock
        // Clock value: Timer2 Stopped
        // Mode: Normal top=0xFF
        // OC2 output: Disconnected
        ASSR=0x00;
        TCCR2=0x00;
        TCNT2=0x00;
        OCR2=0x00;

        // Timer(s)/Counter(s) Interrupt(s) initialization
        TIMSK=0x00;
```

```
    // USART initialization
    // USART disabled
    UCSRB=0x00;

    // Analog Comparator initialization
    // Analog Comparator: Off
    // Analog Comparator Input Capture by Timer/Counter 1: Off
    ACSR=0x80;
    SFIOR=0x00;

    // ADC initialization
    // ADC disabled
    ADCSRA=0x00;

    // SPI initialization
    // SPI disabled
    SPCR=0x00;

    // TWI initialization
    // TWI disabled
    TWCR=0x00;

    // Alphanumeric LCD initialization
    // Connections are specified in the
    // Project|Configure|C Compiler|Libraries|Alphanumeric LCD menu:
    // RS - PORTA Bit 0
    // RD - PORTA Bit 1
    // EN - PORTA Bit 2
    // D4 - PORTA Bit 4
    // D5 - PORTA Bit 5
    // D6 - PORTA Bit 6
    // D7 - PORTA Bit 7
    // Characters/line: 20
    lcd_init(20);
    for (temp = 0; temp < 80; temp++) lcd_buf[temp] = 0;

    // External Interrupt(s) initialization
    // INT0: On
    // INT0 Mode: Any change
    // INT1: On
    // INT1 Mode: Any change
    // INT2: On
    // INT2 Mode: Falling Edge
    GICR|=0xE0;
    MCUCR=0x05;
    MCUCSR=0x00;
    GIFR=0xE0;
}
```



U1
ATmega644P
RESET
XTAL2
XTAL1
AREF
AVCC
VCC
GND
(ADC0/PCINT0)PA0
(ADC1/PCINT1)PA1
(ADC2/PCINT2)PA2
(ADC3/PCINT3)PA3
(ADC4/PCINT4)PA4
(ADC5/PCINT5)PA5
(ADC6/PCINT6)PA6
(ADC7/PCINT7)PA7
(XCK0/T0/PCINT8)PB0
(CLKO/T1/PCINT9)PB1
(INT2/AIN0/PCINT10)PB2
(OC0A/AIN1/PCINT11)PB3
(OC0B/SS/PCINT12)PB4
(MOSI/PCINT13)PB5
(MISO/PCINT14)PB6
(SCK/PCINT15)PB7
(RS/SCL/PCINT16)PC0
(RD/SDA/PCINT17)PC1
(EN/TCK/PCINT18)PC2
(TMS/PCINT19)PC3
(D4/TDO/PCINT20)PC4
(D5/TDI/PCINT21)PC5
(D6/TOSC1/PCINT22)PC6
(D7/TOSC2/PCINT23)PC7
(RXD0/PCINT24)PD0
(TXD0/PCINT25)PD1
(RXD1/INT0/PCINT26)PD2
(TXD1/INT1/PCINT27)PD3
(XCK1/OC1B/PCINT28)PD4
(OC1A/PCINT29)PD5
(OC2B/ICP/PCINT30)PD6
(OC2A/PCINT31)PD7
DCC_EN
MOSI
MISO
SCK
RTS
Pulse sensing
RX
TX
S00
S90
LED_A
LED_D
+5V
VCC
RESET
R1 10k
C11 100µF
22pF
20MHz Y1
C3
C4 22pF
VREF
100nF C5
10µH L2
L1 0.1µH
C2 4.7µF
C1 100nF
GND
Reset S2
R18 5.6k
J8 ICSP
MISO
SCK
RESET
MOSI
R15 39
Q3 BC557b
R14 5.6k
Q2 BC557b
R13 5.6k
5k adj. 2.63k
R12
C16 4.7µF
C19 100nF
C(LED-)
A(LED+)
RES
D7(D3)
D6(D2)
D5(D1)
D4(D0)
D3
D2
D1
D0(SOD)
E(SCLK)
R/W(SID)
RS(CS)
VEE
VDD
VSS
LCD-EA_DIP204-4BLACK/WHITE
DIS1
ABCDEFGHIJKLMNOPQRST
UVWXYZÄÖabcdefghijkl
mnopqrstuvwxyzäö0123
456789()!:<>
DCC_EN
R6 5.6k
R5 560
Q1 BC557b
C12 100µF
U3
+VIN +VOUT
CTRL
-VIN -VOUT
TMR0512
+12V
C13 100µF
Pulse sensor
J4
+12V
adj. 2.9k
R3 5k
S00
R4 5k
S90
Sample schematic for AVR and
Leine & Linde RS501
incremental pulse sensor


Proto device



Power resistor and heat sink for the LCD-display.
Without increased thermal dissipation, display responsiviness will faint in low temperature environments even when it is designed to operate at -20 °C

Standard blue/white LCD display for evaluating purposes.
Not recommended for anything else.


Depth in meters
[0.000]
PELI
STORM CASE
IM2050
PELI PRODUCTS S.L.U.