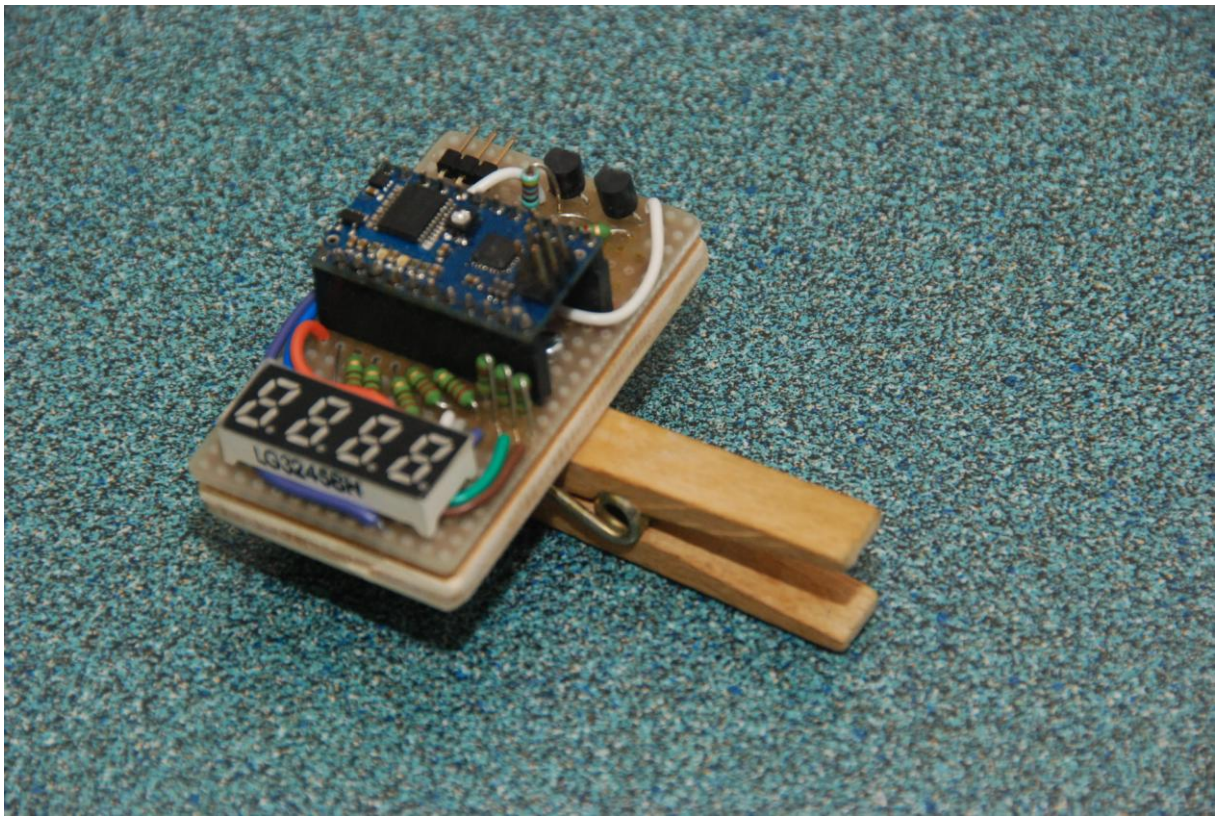
My Frsky Baby-O Dashboard



I wanted a dashboard to monitor power output on my electric RC plane. For this purpose I acquired a Frsky telemetry unit.

I used an Attopilot 50A sensor for analogue input of both U and I. This sensor gives a full scale at 3,3V which matches the specification of the analog inputs of the Frsky receiver.

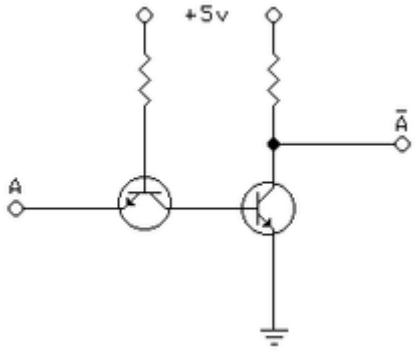
The processor used is an ATmega168 mounted on a Baby-O from www.pololu.com The processor reads the Tx output with an interrupt routine. This allows me to use the main program to send the data to a 7 segment common anode display. The digits get their power from the 1A motor driver available on this board, the consumption is 14 mA per segment with an 220 Ohm resistor installed well below the 40 mA allowed per port.

This is what the application does:

- When the power is off it displays the battery voltage
- As soon as power is required it switches over to displaying the power in Amps

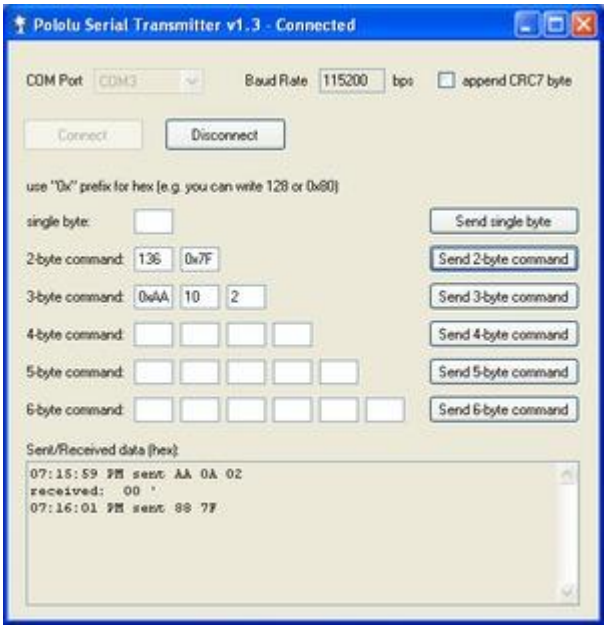
Pitfalls:

1. The Tx output is rs232 and inversed to ttl, so you cannot read the Tx message unless you invert the signal between the Tx and the ATmega168. I used the following schematic:



2. The uart routine may cause problems, I found some very useful tutorials:
   http://www.avrfreaks.net/index.php?name=PNphpBB2&file=viewtopic&t=45341
   http://www.avrfreaks.net/index.php?name=PNphpBB2&file=viewtopic&t=48188

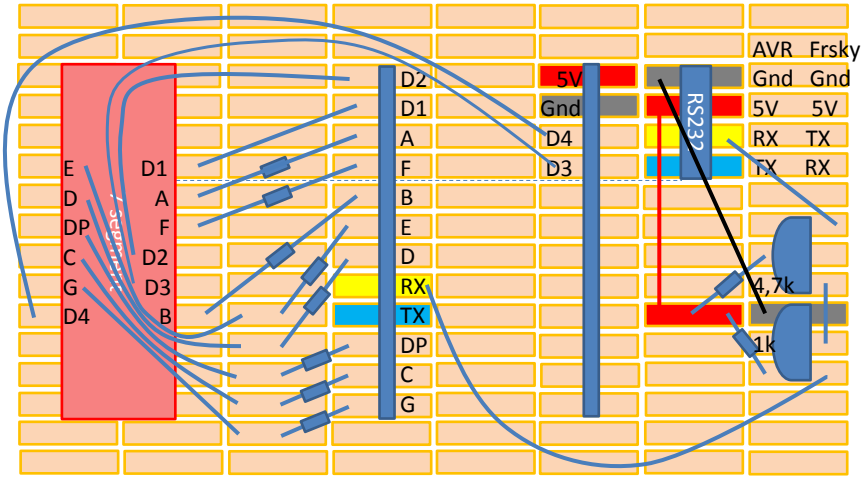
3. Testing an example software is made a lot easier with the Pololu Serial Transmitter for Windows:



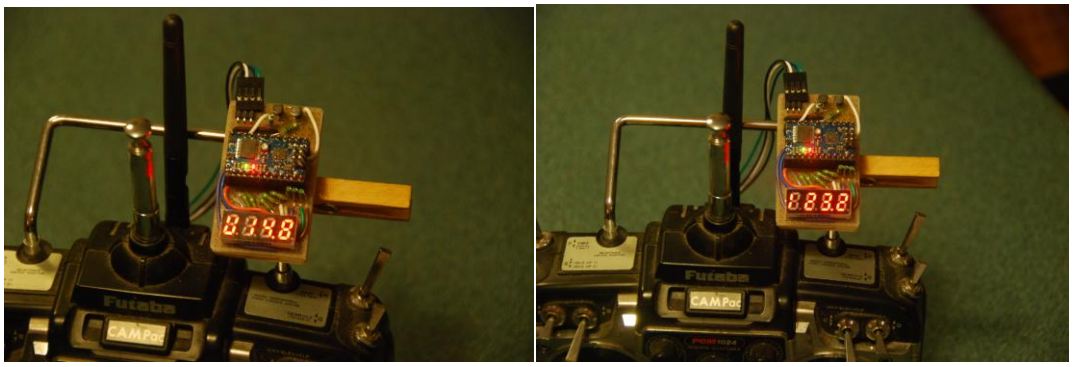
4. Resolution of the sensor is 0..255 which means that in 3300 mV the maximum value is reached.
   3,3V = 3300 mV at a resolution of 255 means 3300/255 = 12,94 mV per point
   Let us take the 30A sensor from Pololu as an example. This sensor has a sensitivity of 133 mV/Amps
   Number of points * 12,94 / sensitivity = output
   Number of points * (12,94/sensitivity) = output
   Amps = AD2 * (12,94/133) = 0,09729 or AD1 * (97,3/1000)
   As we prefer to work with integers we can adjust either the 97 or the 1000 to come as close as we can. AD2 * (97/997) is less than 0,001 off and accurate enough for the resolution of 255. So if we modify `amps = ad2;` into `amps = (ad2 * (97/997));` the reading will

be accurate.

5. Assembly
   I prefer to use a piece of development board with connectors, the baby-O moves on from project to project.



6. The device in action on my old and trusty FF8.



Code device.h

```
// Taken from Tom Benedict's Orangutan-lib

// Modified by Ben Schmidel to work with the Orangutan LV-168

// Modified by Erik Janssen to work with the Baby-O 168


#ifndef _DEVICE_

#define _DEVICE_
```

```
#ifndef F_CPU
#define F_CPU 20000000UL
#endif                  //!F_CPU
////////////////////////////////////////////////////////////////////
// Baby-O-168 pin assignments                                     //
////////////////////////////////////////////////////////////////////
// 7 Segent
// On the Baby-O LV-168, the 7 segments control lines are split between
// ports B and D:

#define SEG_DP_C_G_DDR       DDRD
#define SEG_DP_C_G_PORT      PORTD
#define SEG_A_F_B_E_D_DDR          DDRB
#define SEG_A_F_B_E_D_PORT         PORTB
#define A                    PB0
#define F                    PB1
#define B                    PB2
#define E                    PB4
#define D                    PB5
#define C                    PD4
#define G                    PD7

// ***** Character digit PINS *****
#define D2                          PD5         // used to control digit 2
#define D1                          PD6         // used to control digit 1
#define D4                          PD3         // used to control digit 3
#define D3                          PB3         // not used for the 3
character example, may be used for 4 character *** NOTE: M2A is on port B,
not port D
```

```
#endif //_DEVICE_
```

Code main.c

```
#define BaudRate 9600UL
#include <avr/io.h>
#include <avr/interrupt.h>
#include "device.h"     // pinout definitions 7 segment display
#include <util/delay.h>

volatile uint16_t status;
volatile uint16_t ad1;
volatile uint16_t ad2;
uint16_t volt;
uint16_t amps;

void delay_ms( unsigned int time_ms )     // delay for time_ms milliseconds
by looping
{
      unsigned int i;
      for ( i = 0; i < time_ms; i++ )
            _delay_ms( 1 );   // _delay_ms() comes from <util/delay.h>
}

void VoltOn()     // truth table for segments U
{
      { PORTB |= 1 << A; PORTD |= 1 << G; PORTD |= 1 << DP;}
}

void AmpsOn()     // truth table for segments U
{
      { PORTB |= 1 << A; PORTB |= 1 << B; PORTD |= 1 << C;  PORTB |= 1 <<
D; PORTD |= 1 << G; PORTD |= 1 << DP;}
}

void SegmentOn(uint16_t n)    // truth table for segments
{
      if ( n == 0 ) { PORTD |= 1 << G; PORTD |= 1 << DP; }
      if ( n == 1 ) { PORTB |= 1 << A; PORTB |= 1 << D; PORTB |= 1 << E;
PORTB |= 1 << F; PORTD |= 1 << G; PORTD |= 1 << DP; }
      if ( n == 2 ) { PORTD |= 1 << C; PORTB |= 1 << F; PORTD |= 1 << DP; }
      if ( n == 3 ) { PORTB |= 1 << E; PORTB |= 1 << F; PORTD |= 1 << DP; }
      if ( n == 4 ) { PORTB |= 1 << A; PORTB |= 1 << E; PORTB |= 1 << D;
PORTD |= 1 << DP; }
      if ( n == 5 ) { PORTB |= 1 << B; PORTB |= 1 << E; PORTD |= 1 << DP; }
      if ( n == 6 ) { PORTB |= 1 << B; PORTD |= 1 << DP; }
      if ( n == 7 ) { PORTB |= 1 << D; PORTB |= 1 << E; PORTB |= 1 << F;
PORTD |= 1 << G; PORTD |= 1 << DP; }
      if ( n == 8 ) { PORTD |= 1 << DP; }
      if ( n == 9 ) { PORTB |= 1 << E;PORTD |= 1 << DP;  }
}

void DigitOn(digit)                                   // truth table for
switching on M1A, M1B, M2A, M2B
{
      if ( digit == 3 ) { PORTD |=  ( 1 << D1 ); PORTD &= ~( 1 << D2 );
PORTB &= ~( 1 << D3 ); PORTD &= ~( 1 << D4 ); }
      if ( digit == 1 ) { PORTD &= ~( 1 << D1 ); PORTD |=  ( 1 << D2 );
PORTB &= ~( 1 << D3 ); PORTD &= ~( 1 << D4 ); }
      if ( digit == 4 ) { PORTD &= ~( 1 << D1 ); PORTD &= ~( 1 << D2 );
PORTB |=  ( 1 << D3 ); PORTD &= ~( 1 << D4 ); }
```

```
      if ( digit == 2 ) { PORTD &= ~( 1 << D1 ); PORTD &= ~( 1 << D2 );
PORTB &= ~( 1 << D3 ); PORTD |=  ( 1 << D4 ); }
}

void Off()
{
      PORTB &= ~( 1 << A ); PORTB &= ~( 1 << B ); PORTB &= ~( 1 << F );
      PORTB &= ~( 1 << D ); PORTB &= ~( 1 << E ); PORTD &= ~( 1 << C );
PORTD &= ~( 1 << G ); PORTD &= ~( 1 << DP );
      PORTD &= ~( 1 << D1 ); PORTD &= ~( 1 << D2 ); PORTB &= ~( 1 << D3 );
PORTD &= ~( 1 << D4 );
}

void Seg_Off()
{
      PORTB &= ~( 1 << A ); PORTB &= ~( 1 << B ); PORTB &= ~( 1 << F );
      PORTB &= ~( 1 << D ); PORTB &= ~( 1 << E ); PORTD &= ~( 1 << C );
PORTD &= ~( 1 << G ); PORTD &= ~( 1 << DP );
}


void Show_U(para)             // Shows parameter value
{
      uint16_t number; number = 0;
      unsigned int i;
      for (i = 0; i < 100; i++)     // counts showtime
      {
            number = (para)%10;     // Convert number to segments
            SegmentOn(number);      // Send segments to LED
            DigitOn(4);             // Turn on fourth digit
            delay_ms(1);            // Leave it on 1 ms
            Off();                  // Turn off digit to prevent ghosting
            number = (para/10)%10;  // Convert number to segments
            SegmentOn(number);      // Send segments to LED
            if ( para >   9 ) { DigitOn(3); PORTD &= ~( 1 << DP );}
                                    // Turn on third digit
            delay_ms(1);            // Leave it on 1 ms
            Off();                  // Turn off digit to prevent ghosting
            number = (para/100)%10;// Convert number to segments
            SegmentOn(number);      // Send segments to LED
            if ( para >  99 ) { DigitOn(2); }
                                    // Turn on second digit
            delay_ms(1);            // Leave it on 1 ms
            Off();                  // Turn off digit to prevent ghosting
            number = (para/1000)%10;      // Convert number to segments
            VoltOn();               // Send segments to LED
            DigitOn(1);             // Turn on first digit
            delay_ms(1);            // Leave it on 1 ms
            Off();                  // Turn off digit to prevent ghosting
      }
}

void Show_I(para)             // Shows parameter value
{
      uint16_t number; number = 0;
      unsigned int i;
      for (i = 0; i < 100; i++)     // counts showtime
      {
            number = (para)%10;     // Convert number to segments
            SegmentOn(number);      // Send segments to LED
            DigitOn(4);             // Turn on fourth digit
```

```
            delay_ms(1);            // Leave it on 1 ms
            Off();                  // Turn off digit to prevent ghosting
            number = (para/10)%10;  // Convert number to segments
            SegmentOn(number);      // Send segments to LED
            if ( para >   9 ) { DigitOn(3); PORTD &= ~( 1 << DP );}
                                    // Turn on third digit
            delay_ms(1);            // Leave it on 1 ms
            Off();                  // Turn off digit to prevent ghosting
            number = (para/100)%10; // Convert number to segments
            SegmentOn(number);            // Send segments to LED
            if ( para >  99 ) { DigitOn(2); }
                                          // Turn on second digit
            delay_ms(1);      // Leave it on 1 ms
            Off();            // Turn off digit to prevent ghosting
            number = (para/1000)%10;      // Convert number to segments
            AmpsOn();         // Send segments to LED
            DigitOn(1);       // Turn on first digit
            delay_ms(1);      // Leave it on 1 ms
            Off();            // Turn off digit to prevent ghosting
      }
}

void uart_init (void)
{
      uint16_t ubrr = (uint16_t) ((uint32_t)F_CPU/(16*BaudRate)-1);
      UBRR0H = (uint8_t)(ubrr >> 8);
      UBRR0L = (uint8_t)(ubrr);

      UCSR0B = (1<<RXEN0)|(1<<TXEN0);
      UCSR0C = (1<<UCSZ01)|(1<<UCSZ00);

      UCSR0B |= (1 << RXCIE0);
      UCSR0A |= (1 << RXC0);

      sei();
}



int main (void)
{
      uart_init();

      DDRB |= (1 << A) | (1 << B) | (1 << F) | (1 << D) | (1 << E);
      // set LED pin to output
      DDRD |= (1 << C) | (1 << G) | (1 << DP);                          //
set LED pin to output

      for (;;)
      {
            cli ();
            volt = ad1; amps = ad2;
            sei ();

            if ( amps <= 0 )
            {
                  Show_U(volt);     // Shows value of the parameter volt
            }
            else
            {
                  Show_I(amps);     // Shows value of the parameter amps
```

```
            }
      }

return 0;
}


ISR(USART_RX_vect) // RXC modified to RX, this is WRONG in datasheet
{
      unsigned char buffer;
      while ( !(UCSR0A & (1<<RXC0)) );
      buffer = UDR0;

      if (status==2)
      {
            ad2 = buffer; status = 0;
            // after amps logged input waits for next input cycle
      }

      if (status==1)
      {
            ad1 = buffer; status = 2;
            // next value will be logged as amps
      }

      if (buffer == 0xFE)
      {
            status = 1;
            // next value will be logged as volts
      }
}
```