onderous Adventures of

echanical

WC

# Mechanical QWC - Making Of...

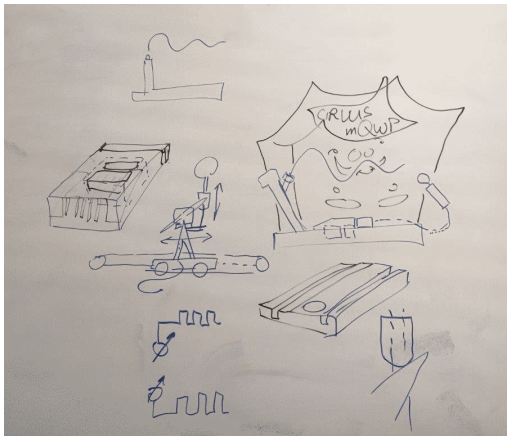
Thomas Becker, Stephan Topp 2019

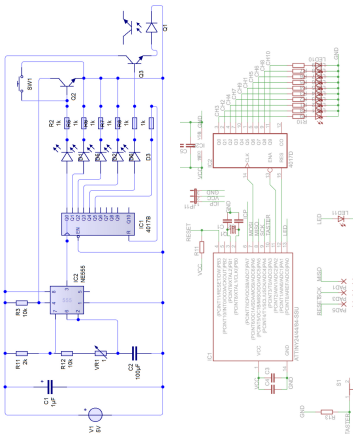
June 30, 2019

## Contents

# 1 Hardware/Software SOD

## 1.1 Idee / Rohentwurfsphase / Alternativen





Aufgabenpakete zum Abstreichen ...

entwurfsskizze.pdf Thomas Becker, Stephan Topp, 27.5.2019

i. EDC Imitat
   - Attiny Platine mit
   - LM263 und CD4017,
   - 4 Strang Motor-Servo-Kabelbaum,
   - 1 Strang zum Ring, 1 Strang zum Injektor
   - sowie 11 Strang Kabelbaum zum Bild

ii. Drehknopf-Armatur (Eingabegert), minimal 2 Potis

iii. Summer: Stromkreis und Summer

iv. Hintergrundbild Zelt und Clown , Schild ENS

v. LED Kette den Jongleur- Ballbildern hinterlegt

vi. Fhrleine fr Motor-Servo-Ring Kabelbaum als Mast-Deko

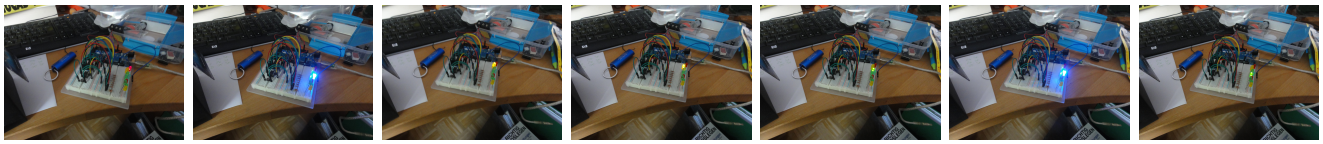
vii. Leitung, Injektor-Wellenverband im Holz

viii. Ringkonstruktion federnd (Servo kann nicht extern gedreht werden...)

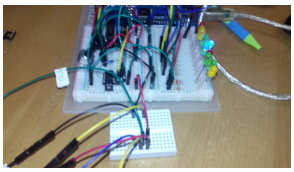
ix. Doppelschwinge, um Servo in Hub zu verwandeln

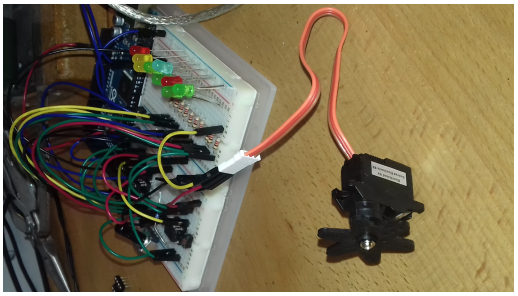
x.
   - a) Wagen mit Motorhalterung, Reibrad zum Antriebsrad
   - a) Halterung Servo und EDoppelschwinge
   - a) Stecker: Ring-, Motor-Servo-Kabelbaum...
   - b) Schiene zur Fhrung des Wagens (in Grundbrett inegriert oder aufgesetzt)

## 1.2 Blinkenlights

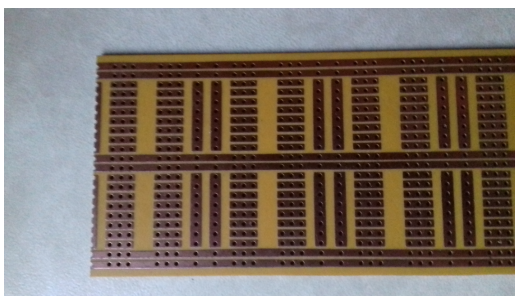


## 1.3 Flash-Station, Servo-Montage

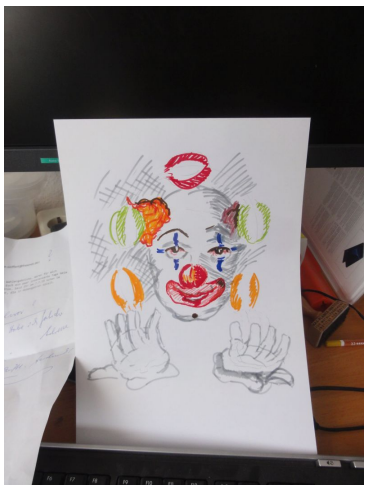
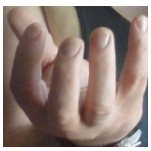
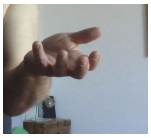
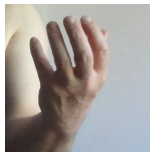




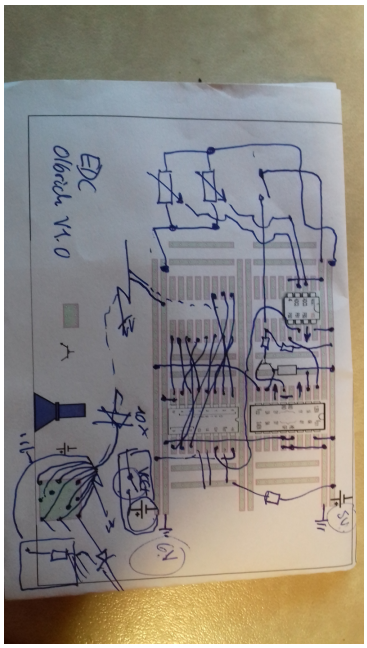
## 1.4 PCB Auswahl



### 1.5 Entwurf, Rohmaterial und Machbarkeitsstudien

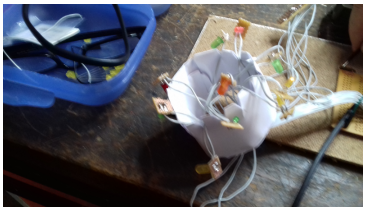





### 1.6 Professionelles Layout im CAD System, Vergabe an Designbüro, Fassung montiert



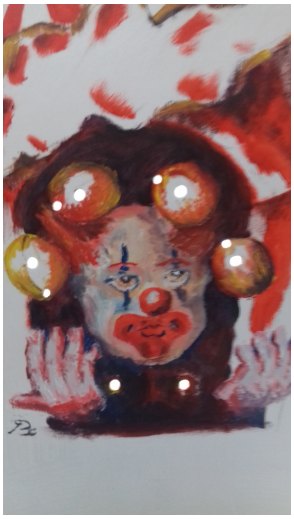
### 1.7 nach SEB Musterfertigung Blinkenlights





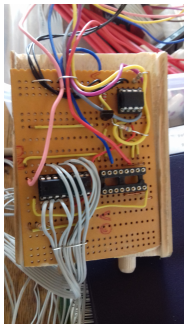
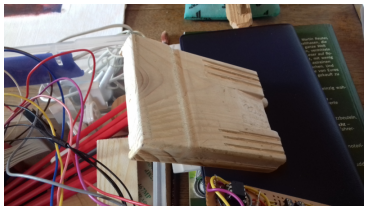
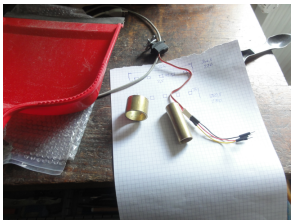
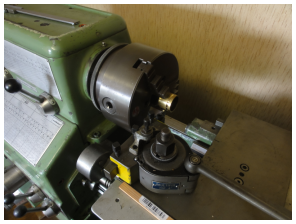

### 1.8 Packaging und Kabelbaum



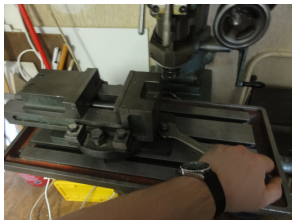
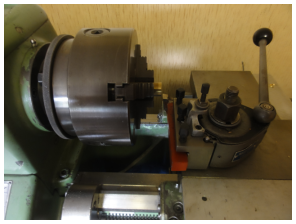
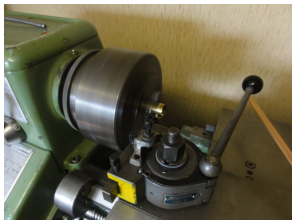
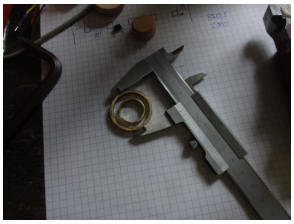


Loch an Loch und hält doch.....

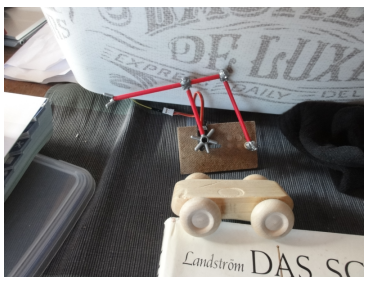
### 1.9 HW DRBFM und AE Gehäusekonzept, Fertigung Toleranzmaß

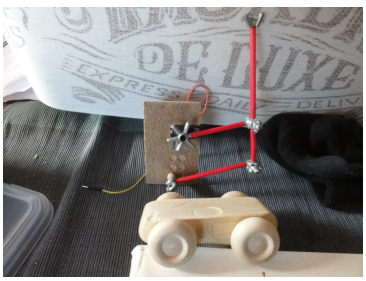






### 1.10 QWC SMD Vias setzen: QWC...grob...fein...tolerieren

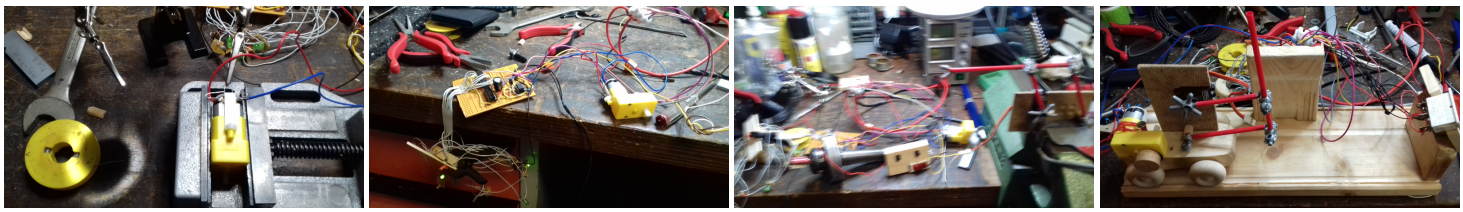





### 1.11 Kalibrieren Amplituden und Eingriffsgrenzen im A-Muster...



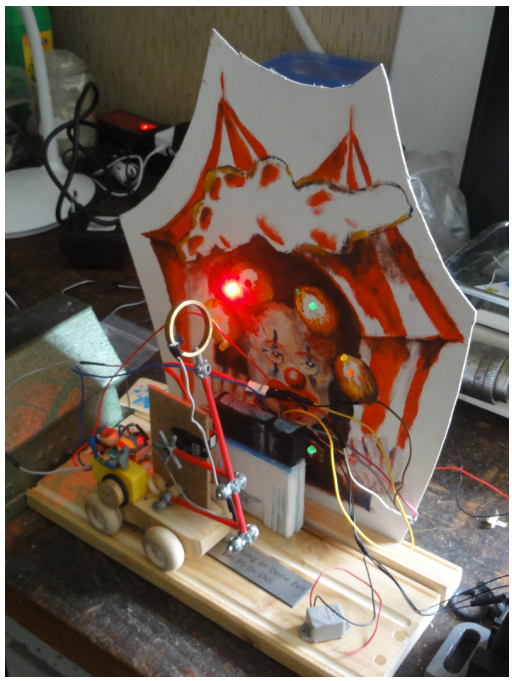


### 1.12 TDIFF Setpoint und minTDIFF...und User Interface



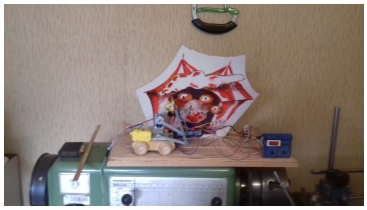
### 1.13 first Firing A Muster

### 1.14 Das A-Muster

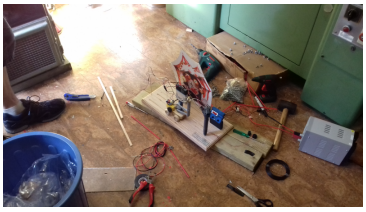
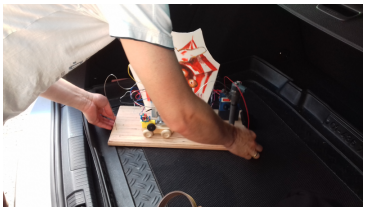


### 1.15 Marketing und Zuliefererwechsel durch Einkauf

### 1.16 Sichtprüfung neues Design B Muster




### 1.17 Modernes 'Ground Floor' Fertigungskonzept,Verpackung und Auslieferung


### 1.18 Das B-Muster


### 1.19 Serienteil hier einkleben :-)

## 2 Software

```
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#define N_1     (_BV(CS00))
#define N_8     (_BV(CS01))
#define N_64    (_BV(CS01)|_BV(CS00))
#define N_256   (_BV(CS02))
#define N_1024  (_BV(CS02)|_BV(CS00))
//Prototypes
void Delay(unsigned int deli);
static void pwm_init(void);
static void pwm_set_frequency(uint32_t N);
static void pwm_set_duty(uint8_t duty);
static void pwm_stop(void);
void initADC();
uint16_t read_ADC10(uint8_t channel);
uint16_t meinErgebnis = 0;
volatile int state =0;
volatile int s =0;
volatile unsigned char ton1flag =0;
volatile unsigned char dutyservo =60;
// 13 repetitions of timer OVF create 20 ms timebase for Servo
#define NSKIP 13

ISR(TIM0_OVF_vect) /* earlier: in gcc: SIGNAL(SIG_OVERFLOW0) */
{
    switch(state)
        {
        case 0:
                PORTB|=(1<<PB1);    // engine fast pwm creation: pin on, off is done by HW capture of Tim
                if (!ton1flag)                // requested SERVO on is shorter then OVF period
                {
                    TIFR0 = (1<< OCF0B);        // clear pending IR (thanx, B.Becker!)
                    TIMSK0 |= (1<<OCIE0B);    // allow second capture IR
                    s = TCNT0;     //debug
                }

         break;
        case 1:
                if (ton1flag)       // requested SERVO on is longer than one but shorter then 2 OVF perio
                {
                    TIFR0 = (1<< OCF0B);
                    TIMSK0 |= (1<<OCIE0B);
                        s = TCNT0;   //debug
                }
         break;
        case NSKIP:
            state =-1;
         break;
        default:
        break;
        }
    state++;
}

ISR(TIM0_COMPB_vect) /* outdated version was : SIGNAL(SIG_OVERFLOW0) */
{
        PORTB &=~(1<<PB1);              // reset servo pin and disable interrupt
        TIMSK0 &= ~(1<<OCIE0B);
}

void setServoDuties(uint8_t dutyServoAD, uint8_t * c, int8_t * flag)
{
```

```
                        // now handle Servo: linear transmission ad to duty
                        *flag = (dutyServoAD > 156)? 1: 0; //signal longer then 1 TOV of TI0
                        *c = 60 + dutyServoAD + (dutyServoAD >> 2);
                        // 1.2 times AD = approx 1 AD + 1/4 AD ...
                        //partial or complete remainder for Compare Interrupt OCR0B by uint8_t overflow
                        //safety limits.... min 0.4 ms
                        if (flag ) {if (*c < 10 ) *c =10;} else {if  (*c<60) *c=60; }
}

int main(void)
{
        unsigned char c;
        int8_t flag;
        uint8_t firstdutyServoAD;
      uint8_t firstdutyEngineAD;
    /* setup */
      pwm_init();
        initADC();
      pwm_set_frequency(N_8);
        firstdutyEngineAD = (uint8_t)read_ADC10(3);
        firstdutyServoAD = (uint8_t)read_ADC10(2);
        pwm_set_duty(firstdutyEngineAD);  //enters fast PWM duty directly
        setServoDuties(firstdutyServoAD,&c, &flag);
        ton1flag = flag;
       OCR0B=dutyservo = c;

    /* loop */
    while (1) {
                uint8_t dutyEngineAD = (uint8_t)read_ADC10(3);
                uint8_t dutyServoAD = (uint8_t)read_ADC10(2);
              pwm_set_duty(dutyEngineAD);  //enters fast PWM duty directly
                // now handle Servo: linear transmission ad to duty
                setServoDuties(dutyServoAD,&c, &flag);
                // --> start interrupt protected, consistent transfer of remainder and timer flag
                cli();
                ton1flag = flag;
               OCR0B=dutyservo = c;
                sei();
                // <-- end of interrupt save consistent transfer of remainder and timer flag
               PORTB |= (1<<PB2);     //Blinkenlights: set LED counter trigger active
            Delay(400);             //wait for certain ..arbitrary length ...
              PORTB &= ~(1<<PB2);    //reset LED counter trigger and pass some time near sleeping
             Delay(2000);
    }
    pwm_stop();  // never to be reached :-)
}

// helper routines for comfort:
static void pwm_init(void)
{
    DDRB |= _BV(PB0) | _BV(PB1)| _BV(PB2);    // set PWM pin as OUTPUT
    TCCR0A |= _BV(WGM01)|_BV(WGM00);      // set timer mode to FAST PWM
    TCCR0A |= _BV(COM0A1);          // connect PWM signal to pin (AC0A => PB0)
    OCR0B=60;
}

/* When timer is set to Fast PWM Mode, the frequency can be
calculated using equation: F = F_CPU / (N * 256)
Possible frequencies (@1.2MHz):
 -> F(N_1) = 4.687kHz
 -> F(N_8) = 585Hz
 -> F(N_64) = 73Hz
 -> F(N_256) = 18Hz
 -> F(N_1024) = 4Hz */
static void pwm_set_frequency(uint32_t N)
```

```
{
    s=(TCCR0B & ~((1<<CS02)|(1<<CS01)|(1<<CS00))) | N;
    TCCR0B = (TCCR0B & ~((1<<CS02)|(1<<CS01)|(1<<CS00))) | N; // set prescaler
         // Overflow Interrupt erlauben
    TIMSK0 |= (1<<TOIE0);
        sei();
}
static void pwm_set_duty(uint8_t duty)
{
    OCR0A = duty; // register for automatic capture and pin toggle operation value
}
static void pwm_stop(void)
{
    TCCR0B &= ~((1<<CS02)|(1<<CS01)|(1<<CS00)); // stop the timer
        cli();
}

// only in idle-main task, as the cpu is busy doing actively nothing :-)
void Delay(unsigned int deli)
{
        volatile unsigned int del = deli;
        while(del--);
}

void initADC()
{
        // ADMUX |= (1 << REFS0);   //use internal reference voltage of 1.1V -- nope!
        DIDR0 |= (1<< ADC2D);
        DIDR0 |= (1<< ADC3D);
        //select ADC Channel 3;
        ADMUX |= (1 << MUX0);   //
        ADMUX |= (1 << MUX1);   //sets ADC3 as analog input channel
        //only 8 bit needed....
        ADMUX |= (1 << ADLAR);   //Left Adjust the ADCH and ADCL registers
        //enable adc
        ADCSRA |= (1<<ADEN);
        ADCSRA |= (1 << ADPS1);   //
        ADCSRA |= (1 << ADPS0);   //set division factor-8 for 125kHz ADC clock
        //ADCSRA |= (1 << ADSC);   //start conversion
}

uint16_t read_ADC10(uint8_t channel)
{
        uint8_t i;
        meinErgebnis = 0;

        ADCSRA |= (1<<ADEN);
        ADCSRA |= (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0);
        ADMUX  &= ~3;                    //erst ADC Kanal loeschen !!! da mir OR gewaehlt !!!
        ADMUX  |= (1 << ADLAR) | channel;                        //ADC Kanal waehlen
        //ADMUX |= REFS0 ;                       // Referenzspannung Intern nutzen! - nope
        ADCSRA |= (1<<ADSC);             // eine ADC-Wandlung starten
        while ( ADCSRA & (1<<ADSC) ) { ; }

        // Messen - Mittelwert aus 4 Wandlungen bilden
        meinErgebnis = 0;                                // Summen Inhalt von meinErgebnis loeschen
        for(i=0;i<4;i++)
        {
                ADCSRA |= (1<<ADSC);     // eine ADC-Wandlung starten "single conversion"
                while ( ADCSRA & (1<<ADSC) ) { ; }       // wait for conversion
                meinErgebnis += (uint16_t)ADCH;                  //sum up analog results
        }                                                        // only use low 8 bit AD resolution , ig
        ADCSRA &= ~(1<<ADEN);            // ADC ausschalten
        meinErgebnis >>= 2;                              // Mittelwert bilden / >>2 = teile durch 4
        return meinErgebnis;  // Analog Wandlung...
}
```