



## Servo Controller 3.00

### **Introduction**

The UHU Servocontroller 3.00 was implemented on an Atmel ATtiny2313, with the goal to provide an affordable alternative for commercial servo controllers. The focus lies on easy logical integrability and a simple user interface, which provides a maximum of flexibility in parametrizing the control loop.

The algorithm is an extended PID controller, where the usual proportional, integral and differential feedbacks were complemented by higher order feedbacks. This empowers the controller to stabilize even difficult setups at high speeds while not having significant tracking errors.

The control of the CNC system is done by a step/dir interface which is the common interface to PC-CNC controllers.

Parametrizing is done by plain text dialogue over a serial communication interface.

All operating parameters can be stored in an EEPROM area and are therefore easily reproducible

Multiple sets of parameters allow the convenient change between different scenarios.

By utilizing a special terminal program which is provided with the processor, the tracking error can be visualized in real time over a moving timeline. This ensures best possible conditions for optimizing the control loop.

A tracking error shutdown function turns off the output stage if a configurable tracking error is reached. An error input allows to loop in error conditions and to ensure that all axis stop in case of an emergency.

An internal watchdog turns off the output stage, if the recalculation routine was not jumped in for more than 125ms due to misconfiguration or whatever error situation.

The controller does a 4-fold interpretation of the encoder lines and is able to process over 100.000 steps/s. All Processing in real-time and driven by hard- and counter interrupts.

***Attention: Machines, particularly those with automated controls and devices with high voltages are potentially dangerous.***

***Due to legal reasons, the UHU-Servocontroller is intended only for innocuous demonstrating setups.***

***The control of machines with the UHU-Servocontroller underlies the personal responsibility of the user as long as it is not explicitly approved.***

***Any responsibility or warranties are expressly excluded.***

# Requirements

## PCB

The controller needs to be placed on a proper designed PCB which has to comply to common EMC rules. The reason lies in very high currents in combination with high frequencies and sensitive logic components. The best solution to date is the PCB posted by ‚Stefan8051‘ in "Peters CNC-Ecke" [www.5128.rapidforum.com/](http://www.5128.rapidforum.com/). An own attempt pays only if specific needs must be taken into account.

Nevertheless some information about hardware implementation:

The controller needs an output stage which has to be compatible to the 'Selbstbausteuering' published in "Peters CNC-Ecke" in 2004. The processor needs a 24 MHz crystal (1<sup>st</sup>. order).

A 20 kHz PWM signal is generated on portB/3 (Pin15). portB/4 (Pin 16) provides a power enable signal. The PWM signal control range is limited to 13% to 87% to ensure proper operation of the charge pumps in the MOS-Fet-drivers IR2184 or IR2104. Hence voltage of the output stage has to be 15% higher than the voltage specified on the motor to provide maximum performance.

There is an option for connecting a 'Ready-LED' on portB/5 and a 'Error-LED' on portB/6 (680 Ohm to Vcc).

Signals:

Step:	PortD/2 Pin 6
Dir:	PortD/3 Pin 7
Encoder A	PortB/0 Pin 12
Encoder B	PortB/1 Pin 13

It is highly recommended to have an electrical isolation between PC and servo controller by the use of opto-couplers. It is essential to use high quality types with a bandwidth over 1 Mbps and low input current. We had best results with the types HCPL7601 and HCPL22xx made by HP/Agilent.

Level shifting of the serial lines can be done in a simple way as shown in the circuit below. A true level shifter like MAX232 however is always the better choice.

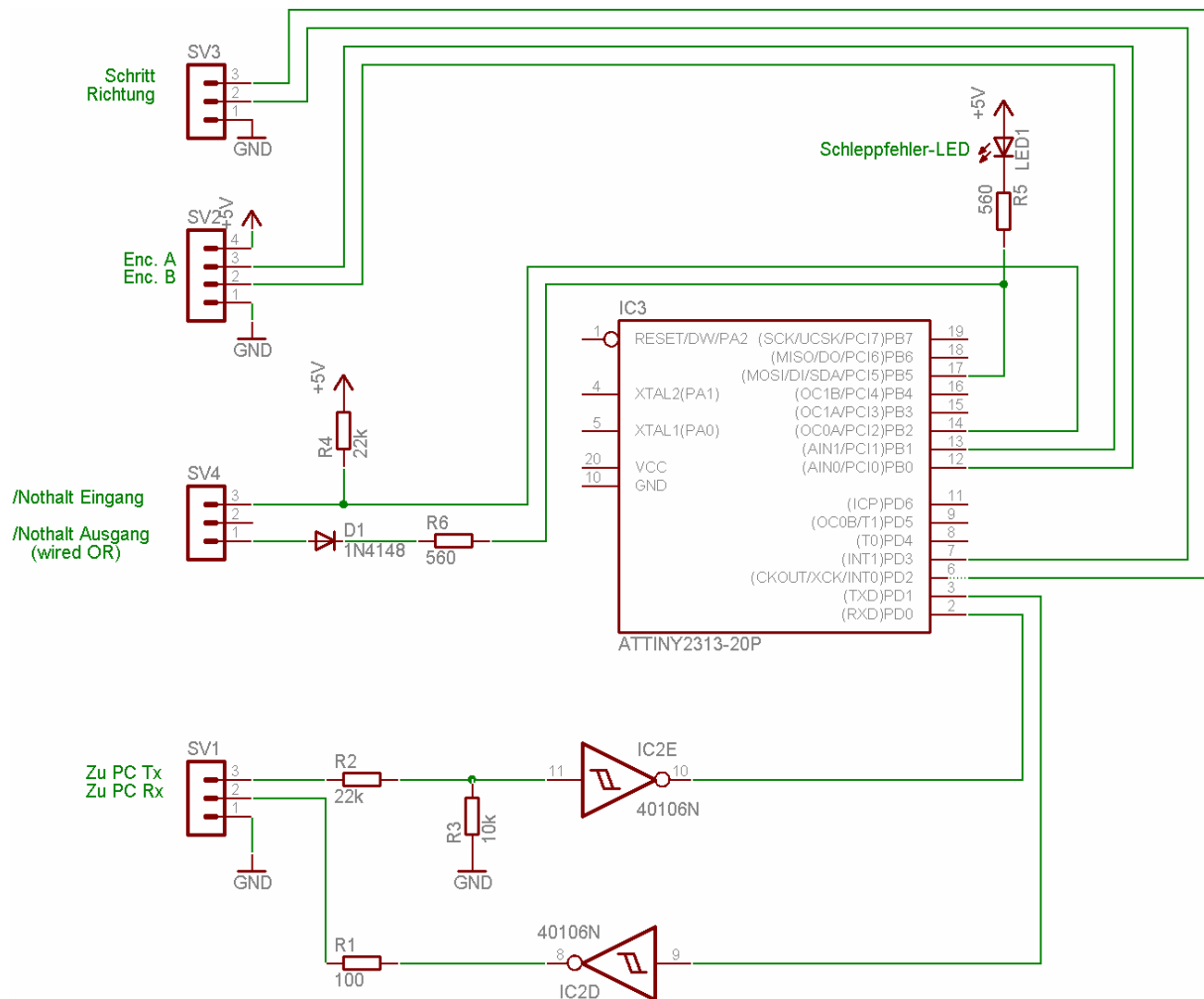
If needed, there is an error input available on portB/2 Pin 14 and an error output on portB/5 Pin 17.

Error input has to be at Vcc level (+5V) for normal operation. If it is pulled to ground, the controller does an emergency shutdown within the next recalculation interval (i.e.600us). The output stage stays deactivated till the next soft- or hard-reset.

Multiple servo controllers can be connected in parallel if the suggested diode is used (wired-or). If not used, the error input portB2 pin 14 should be tied to Vcc/+5V with a 4k7 resistor.

AVR controllers in general are pretty sensible to electrical interferences. The use of 100nF blocking capacitors over all IC power pins and parallel to all ELKOs is a must. In general, the close neighborhood of high power switching and low level logic needs special consideration of EMC compliant design and professional setup. Almost all known problems in the past were caused by inappropriate PCBs or frowzy cabling.

In spite of the built in reset algorithm and brown-out-detectors of the Tiny2313, the common way to connect Reset to a R-C-combination seems not to be ideal. It is highly recommended to use a reset controller like TL7757CLP, MC34064P-5 or MC33064P-5. Do not use a reset controller with a switching level under 4.5V. Never exceed an operating voltage of 5.5V.



This partial circuitry shows essential changes to the old PCB of the 'Selbstbausteuering'. The full documentation of the circuit was posted to "Peters CNC Ecke" by "Stefan8051"

## Terminal program

The use of the supplied terminal program Serv200.exe requires a DOS-PC with a serial interface COM1. It is not possible to run the program under 32 Bit Windows like W2k or XP. A Win98 Dos Box should be OK.

## Method of Operation

The servo controller processes position commands coming from the step/dir interface and position feedback coming from the motor encoder to a PWM signal that reflects a current command which makes the motor running to the commanded position.

The control loop utilizes an extended PID algorithm and ensures fast response to position commands along with high stability of the control loop and minimal tracking errors.

The recalculation interval is software configurable and constant over all conditions to guarantee a reliable operation.

All input signals are treated as hardware interrupts and executed consecutively with regard to their priority..

## Settings

Parametrizing is done solely over the serial interface. As it provides additional functions like a graphical display of the step function response, the DOS-Program Serv2000.exe is the preferred choice.

Any other terminal program can be used, if it allows to configure the PC RS232 port to the parameters 38400,N,8,1 without handshake and with echo. But usage is limited to setting and reading of parameters.

The serial connection is only used for setup and control functions. It is not needed for normal operation.

**Automated machines are potentially dangerous. It is highly recommended to read the entire documentation before activating the servo controller.**

In terminal operations all inputs are done on a command line after the prompt '>' All input lines are submitted by the press of the enter key, displayed below as '␣'

```
UHU Servo Controller 3.00 (c) Snr: 134
EEPROM: OK
Help by (?)
>
```

In principle a letter has to be pressed which represents the function. A help function is available by the press of the '#' key.

### Example

```
># ␣
(P)rop.      900
(I)nt.       24
(D)iff.      260
(H)iOrd.     400
Ma(X)        300
(T)orque     255
I(N)ertance  150
(E)rror      2000
(M)ult.      0
(O)verrun    0
(C)ycle      50
(L)oad       [0,1,2]
(S)ave       [0,1,2]
(G)o         [Wert]
(C)ounter
(R)eset
>
```

Parameters can be changed by the press of a letter, followed by the target value of the desired parameter. It will be confirmed with 'OK'.

```
> P600 ␣
OK
>
```

If no value is supplied, the actual value is displayed with the possibility to enter a new value.

```
> P ␣
(P)rop.      900: 600␣
OK
>
```

No input value leads to an unchanged parameter.  
All input values integers, no decimals.

*Parameters are internally handled as 16 bit fixed point number. Therefore the first 8 bits represent the value before the decimal point, the second 8 bits the value behind the decimal point as multiples of 1/256.*

Factual there is no need for those calculations if the controller is parametrized by intuition.  
Just treat values as normal integers !

## **Proportional constant(P)**

As the name is suggesting, this value generates a control variable which is proportional to the actual position difference. The higher the value is, the more aggressively is the reaction on new position commands.

If the value is to high, it causes instabilities and low frequency oscillations with slowly declining or inclining amplitude.

Possible values are in between 50 and 5000 dependent on the encoder resolution and motor dynamics.

## **Integral constant (I)**

This value generates a control variable which increases over time with all unbalanced positions. This ensures to compensate even smallest errors (i.e. 1 encoder line) by increasing the control variable by 1/256 of the given integral constant in every recalculation interval until it is sufficient to make the motor start.

If the value is to high, it causes high frequency oscillations with inclining amplitude.

Possible values are in between 1 and 120 dependent on the encoder resolution and motor dynamics.

Attention: In Rel 3.00 the effect of the I constant was reduced by a factor 4 compared to the old Versions. This makes it easier to handle the mostly small values.

## **Differential constant (D)**

This value generates a degenerative feedback which is proportional to the actual speed to stabilize the control loop. The drawback is a tracking error, increasing with the speed. This makes the H parameter the better choice in most cases. If not needed, enter 255.

If the value is to low, it causes high frequency oscillations with inclining amplitude.

Possible values are in between 255 and 20.000 dependent on the encoder resolution and motor dynamics.

## **Higher derivative constant (H)**

This value generates a degenerative feedback which is proportional to the actual speed difference in the last recalculation interval. It stabilizes the control loop without negative influence on the tracking error. Hence it is the better choice over increasing the 'D' constant.

If the value is to low, it causes high frequency oscillations with inclining amplitude.

Possible values are in between 200 and 20.000 dependent on the encoder resolution and motor dynamics.

The influence of the H constant was significantly increased in Rel 3.00.

## Maximum speed(X)

This value limits the maximum speed of the system. It is entirely useless for step/dir operations and should be set to 5000. If 'G' is below the stepping frequency, it causes a tracking error shutdown.

The speed limitation can be used in special applications, where position commands are given by the 'G' command. The tracking error value 'E' must be adjusted to a value bigger than any intended single move.

Reasonable values are 1 to 5000.

## Maximum torque (T)

This values limits the maximum current by limiting the PWM duty cycle.

It can be used if a motor with low impedance has to be attached to a controller with higher voltages. Attention: If the T-value is changed in these setups, it can destroy the output stage or the motor.

Possible values are 50 to 255 (max. duty cycle 87%).

## System inertance (N)

This value affects the dynamic the control variable. It is intended only for fine tuning and is mostly not needed.

Higher values lead to 'sharper' control reactions.  
Useful values are from 50 to 1000

## Tracking error shutoff (E)

This value is the maximum allowed tracking error in encoder steps in 4-fold interpretation before the output stage is shut off and the error-LED is activated.

The output stage stays deactivated till the next soft- or hard-reset

If connected, the error output on portB/5 pin 17 can be used to shutoff surrounding systems.

Possible values are 10 to 5000. Lower values are better/safer.

## Step multiplier (M)

The provided integer value is added supplementary with every step pulse. I.e. if 'M' is set to '2', the motor moves exactly three 4-fold interpreted encoder lines with every step pulse.

This makes it possible to drive setups with high encoder resolutions to high speeds even if the commanding PC is limited in Terms of stepping rate. As it also reduces overhead operations, it is always desirable to increase this parameter if the resolution of the encoders is not needed.

Attention: Setting 'M' to a high value leads to higher tracking errors with every step. The tracking error shutoff 'E' has to be increased accordingly.

Useful values are 0 to 20.

## Recalculation Cycle interval (C)

The provided integer value changes the recalculation interval of the control loop.

Basic setup is 63 (interval 666µs or 1.5 kHz).

The smaller the value is, the shorter is the interval and the smaller are the position differences in each interval. This can be used to tune critical setups with high encoder resolutions as the influence of the 'T' parameter is reduced.

Higher values reduces the needed calculation power and makes it easier to tune setups with lower dynamics.

As all speeds are calculated over position deltas, the change of 'C' also influences the P-, I-, D-, H-, X- and N-constants.

Possible values are 30 to max. 200 (equals from 0.3 ms to 2 ms).

Attention: Values smaller than 30 lead to sequential processing and the loss of exact timing. This can cause unreliable results.

## 'Overrun-counter' (O)

This value has no influence to the control process. It is just a counter for false transitions on the encoder lines. These can be detected because of the fact that transitions always happen sequentially and the two phases must not change both in a time interval (Gray Code).

If the do, it is always due to an error:

- speed/frequency is too high (unlikely in normal setups)
- uneven or disturbed transitions of the encoder lines
- EMC problems
- defective encoder or defective input pin on the controller

The displayed value should always be '0'. Any higher value indicates a severe problem which will for sure lead to step losses and inaccurate operations.

The counter can be zeroed or preset.

## ***Saving configuration parameters***

All parameters can be saved in the EEPROM area of the controller. There are 3 independent sets of parameters available, numbered from 0 to 2.

The values are stored with a checksum and forward error correction to give maximum reliance and to ensure that the controller never operates with untuned or random control loop parameters.

Any hard reset loads and activates parameter set '0'

## **Saving (S)**

Allowed commands are S0, S1 and S2 for the three sets of parameters.

While saving the values to EEPROM, a checksum is generated and saved in addition. There is no logical check of the values.

**Attention: Remind that any values stored to set '0' will be activated at the next startup, even if they make no sense or lead to improper operation.**

It is a good idea to store good configurations in S2 and test configurations in S1.

## **Loading (L)**

Allowed commands are L0, L1 and L2 for the three sets of parameters.

If the configuration cannot be reconstructed despite of forward error correction, the output stage is shut off, an error message is displayed and the error LED is activated.

## ***Additional functions***

### **Go (G)**

This function allows to send an incremental position update to the controller. The controller moves then to the new position with a speed limited by the 'X' parameter. As this is done by setting the position error to the desired value, the tracking error value 'E' has to be always higher than any difference of 'G' to the actual position value to prevent an undesired shutoff.

The function was implemented to support a PC based auto tuning functionality in future.

Possible values are -32.000 to +32000

### **Soft-reset (R)**

The input of 'R' (without any parameter) resets the controller with the actual (unsaved) set of parameters.

The output stage is activated and the error LED is cleared.

The analyze mode is deactivated.

The command is confirmed with 'OK'.

### **Hard-reset (+)**

The input of '+' (without any parameter) restarts the controller. It is identical to pulling pin1 (reset) to zero.

Parameter set '0' is loaded and the output stage is activated. Error LED and analyze mode are cleared.

### **Start analyze mode (A)**

This function is intended to be used only in conjunction with the terminal program Serv2000.exe or any other specialized terminal with the capability to display the tracking error in a scope-like way.

Do not start this command from a normal ASCII terminal !

If there is any interest to write a 32 Bit windows application as a substitute for Serv2000 or integration in any other software, please don't hesitate to contact me. I will happily give you any needed information.

At this point just a rough information about the technical principle.

After the command 'A' (without any parameter) the controller starts to output tracking error values in fixed time intervals which are five times longer than the recalculation interval.

The individual values are transmitted as 8 bit characters with the high bit set as an indication for not transferring normal ASCII characters.

Therefore the transmitted value is a 7 bit value (0..127) with a zero line at 63 and shifted upwards by adding 128.

The command is not confirmed.

### **End analyze mode (Z)**

Just switches off the output of tracking error values. Useful if the analyze mode was activated accidentally on an ASCII terminal.



## Getting started

### Terminal program

The terminal program Serv2000.exe is a pure DOS application and will not run under Windows, due to several needs to access PC hardware directly (serial interface, graphics).

It has to be copied in a folder together with the supporting file egavga.bgi and is called directly from the command line of your DOS PC.

Interface setting is fixed to com1, 38400,n,8 and hence there is no need (or possibility) to change settings.

The program forwards all key presses to the controller, where they are echoed and then displayed on the screen. If you don't see your key presses on the PC, check connection and/or proper operation of your controller board.

By the press of the space bar, the analyze mode is activated by sending an 'A' to the controller. All characters coming from the controller with a high bit set, are treated as tracking error values and plotted as a curve over the time in the graphics window.

Observing this curve gives best possible feedback for optimizing the control loop.

The program execution is stopped by the press of the ESC key.

### Controller

**Attention: Prior to the initial startup of a controller board, always disconnect the motor power supply ! Doing so, you prevent your machine from hazardous movements which could be caused by faulty setup values or hardware defects.**

**If possible, start with a lower motor voltage and a current limitation by employing a regulated power supply. Rise voltage and current limit only if there is no indication for any malfunctions.**

**Never operate a machine without a emergency stop button which directly disrupts the motor power supply !**

Directly after power up, the controller executes a reset procedure, where the operating parameters are read from EEPROM and the control process is initialized.

After the initialization procedure the controller sends a startup message to the terminal.:

```
UHU Servo Controller 3.00 (c) Snr: 134
EEPROM: OK
Help by (?)
>
```

In the condition upon delivery, the loop control parameters are pre-set to more or less save values. Nevertheless a false checksum was generated to force an error condition upon startup. This lead to the following message:

```
UHU Servo Controller 3.00 (c) Snr: 134
EEPROM: Error!
Help by (?)
>
```

After a short check of the values you can activate the controller by sending a soft reset command.

If you want to make the controller start without error condition next time, save your active parameter set with the command 'S0' to EEPROM.

## Setup strategy

Luckily the rough setup of a PID controller is not rocket science (if the bandwidth of the parameters is wide enough).

The extended algorithm which takes care of acceleration effects makes it even easier as it provides a degenerative feedback with less impact on the tracking error.

But there will still be setups where the controller fails - for example those where backlash between motor and encoder is to high and stabilizing the loop comes close to impossible.

Fine tuning on the other hand is something you need an intuition for. Playing with the effects of changing parameters will bring you this proficiency slow by slow.

If the system is not too powerful, the best way to find the best basic setup is to move the motor jerkily against its resistance and to observe the reactions on the SERV2000 terminal.

Using the 'E' command is an alternative for moving the motor by hand for more powerful systems.

### Step 1

Connect the controller board to your CNC PC and a Dos PC with SERV2000.exe running.

### Step 2

Set the control parameters to 'pretty safe' starting values.

P = 1000

I = 25

D = 255

H = 1000

X = 3000

T=255

N=100

Check the System for stability. If it starts oscillating, rise the H and/or D value or reduce M.

### Step 3

Increase P step by step until the system starts to oscillate and compensate with H until oscillating cannot be stopped any more. D should be raised only if H does not lead to satisfying stability in very nervous setups. Remember that D leads to higher tracking errors with high operating speeds.

There is an optimum where a high enough P-value ensures fast system response and damping caused by H does not reduce the dynamic significantly. You have to find this point by trying different combinations and watching the results at the analyzer. The plotted curve should be as smooth as possible, without any oscillations and should show only small amplitudes in acceleration phases.

Ensure, that a ramp is defined in your CNC program which meets the possibilities of your motor and mechanics. Tuning can only be done in your final system setup !

Once you found a stable operating point, reduce P about 10 % to give a safety margin.

If the motor impedance does not match at all, try to decrease T. This stabilizes the loop but reduces the torque of the motor.

### Step 4

Increase I step by step until the system starts to oscillate. Lower I about 10%.

**Attention: Powerful systems make powerful oscillations ! Always stay in reachable distance of an emergency stop button !**

### Step 5

Don't forget to save your configuration with the S0 command. Save a second parameter set with S2 if needed.

# Change log

(in German - sorry;-)

**; Rev 3.00e**  
**; English version**

**;Rev 3.00**  
**; Freigabeversion**

;Rev 2.17  
; Konditionale Assemblierung Watchdog, Initialisierung auf 125ms  
; Konditionale Assemblierung Spacing , bei deaktiviertem 'N' Intervall auf 672ms fixiert  
; Timer Prescaler von 64 auf 256 umgestellt

;Rev 2.16  
; Umstellung der Initialisierungssequenz  
; Preset des N-Timers als Sicherheit für Soft-Reset  
; Korrektur Ausgabestring bei Soft-Reset  
; WD-int sichert Register  
; 'Optische' Verbesserungen am Code

;Rev 2.15  
;WD per Interrupt statt Reset aktiviert

;Rev 2.14  
;Fehler bei multiplem EEPROM-Read behoben.

;Rev 2.13  
; Diverse Aufräumarbeiten (LPM + MOVW)  
; Zähler und Analyzer entkoppelt  
; Umbau I-Wert auf DIV 4

;Rev 2.12  
; Problem bei der Überlappung Z und Analyzer beseitigt.

;Rev 2.11  
; Fehler bei Nutzung R0 in RS232 Input beseitigt  
; Anzeige Seriennummer

;Rev 2.10  
; Analyzer in Z wieder eingebaut  
; Fehler in Reihenfolge Berechnung Grundparameter beseitigt  
; Encodereingänge gleichlaufend zur 2.02t (Pin8-->13, Pin9-->12)

;Rev 2.09  
; Fehler in Z-Funktion beseitigt.

;Rev. 20.8  
; Umbau I-Wert auf DIV 256

;Rev. 2.07  
; Lin-Encoder ausgebaut (Platz...)  
; Counter-Pos auf Register geändert  
; unbenötigte Registerzuordnungen gelöscht  
; Fehlerbehandlung statt Interpolation in den Encoder-Zähler eingebaut  
; Einführung des (W)rap Zählers für verschlafene oder falsche Encoder-Flanken  
; Alle Interrupts für die Dauer der Berechnung der Grundwerte der Regelberechnung deaktiviert  
; Reihenfolge der Variablen verändert (und damit der EEPROM-Parameter) für fail safe  
; Fehler Startwert bei der Initialisierung 20/24MHZ behoben  
; Fehler ZL aus Zähler-Int behoben  
; Busy Flag in Berechnung eingeführt  
; Neuberechnungswert mit neg umgerechnet

;Rev. 2.06  
; Experimentell: Änderung Interrupt-Steuerung / verworfen

;Rev. 2.04, 2.05  
; Experimentell: Temporäre Änderung Zähler-Ausgabe / verworfen

;Rev 2.03  
; Umbau auf Encoder Interruptbetrieb auf Port B0/1 (Pin 12 und 13)  
; Achtung: Parameter für Neuberechnung bezieht sich jetzt nicht mehr auf 1/192 Clk sondern auf 1/64 Clk

;Rev. 2.02t24  
; Umbau auf 24 MHz Quarz  
; Phase Correct PWM 25 kHz

;Rev. 2.02t  
; Watchdog 16ms eingebaut

;Rev. 2.01  
; Slave shutoff auf PINB2 / Pin 14 active low eingebaut.  
; Prüfsumme und Forward Error Correction für EEPROM eingebaut

**;Rev 2.0**  
**; Freigabeversion**

;Rev 1.9.0dual4  
; Schleifenabfolge / Timeslot für Neuberechnung programmierbar  
; Prüfsumme für EEPROM eingebaut  
; Nothalt Ein- und Ausgang in Software vorgefertigt.  
;

;Rev 1.9.0dual3  
; Timeslot auf 2ms verändert  
; Z setzt nun immer erst den Zähler durch Reset zurück  
; Lin. Encoder-Abtastung aktiviert (auf gleichem Port wie rot. / nur Perf. Test)  
; Version läuft mit V=1 sehr rund.

;Rev 1.9.0dual2  
; Fehler beim Limiter-Aufruf beseitigt  
; Eingangsports mit Pullup  
; Limiter auf Analysemodus  
; Ripplefilter ausgebaut. Bringt nichts.

;Rev 1.9.0dual1  
; Versionsnummer nachgezogen auf 1.9  
; Ripplefilter (24 Takte) in Step Input eingebaut  
; Register Zuordnungen verändert: \_analyzeDiv, \_mul, \_0

;Rev 1.8.9dual4  
; Aufrufe für Help und Selektion der Parametereingabe verkürzt  
; Trigger für Analysemodus umgebaut  
; Analysemodus auf 16 Bit  
; Flag gegen Reentry der Berechnung eingebaut

;Rev 1.8.9dual3  
; Massive Umbauten.  
;- Mul1616 auf indirekte Adressierung umgebaut  
;- AddMul1616 eingebaut  
;- 16 bit Limiter eingebaut

;Rev 1.8.9dual  
; Eingang für Linearencoder eingebaut (unfertig, Differenzierung der Encoder fehlt...)  
;Dazu:

; Variablen in R- und L- gedoppelt  
; Alle Counter auf Memory umgebaut bis auf Pos und Comm  
; Geschwindigkeitsbegrenzer ausgebaut

;Rev 1.8.9  
;Schleifendurchläufe gecheckt  
;Kommentare vereinheitlicht  
;xmit2 ausgebaut

;Rev 1.8.8.y  
; Einbau B-Kompensation  
; PC-Software neu skaliert: 16 Zeilen Text, Display schmaler.

;Rev 1.8.8.x  
; Reduzierung des Rom Bedarfs durch bessere Stringverwaltung

;Rev 1.8.8  
; seltsame Probleme beim Reset gelöst: nicht initialisierter V\_Par  
; Interrupt-Tabelle vervollständigt

;Rev 1.8.7  
; PC-Software auf reinen Grafik-Betrieb umgestellt, ID und (C) eingefügt  
; Störender Balken beim Scrollen beseitigt  
; Umschaltung Analysemodus mit <Leertaste>

; Rev 1.8.6  
; Hilfe für Analyse und Rückmeldung entfernt  
; Analyse a=on x=off  
; Eilgang-Steps mit Cursor (32) und Ctrl-Cursor(64) in PC-Software eingebaut

;Rev 1.8.5  
; Test neuer Schleppfehler mit Basis alte Soll Pos  
; SEI versetzt hinter Schleppfehlerausgabe  
; Schleppfehler-Led ausgebaut, zeigt jetzt nur mehr Fehlerabschaltung an  
; Umbenennung alle Variablen auf xxxxxL/M/H

;Rev 1.8.4  
;-Reset schaltet Analysemodus mit ab.  
;-falscher Text bei 'Vervielfacher' behoben

;Rev 1.8.3  
;-Texte Komprimiert  
;-(M)oment und (T)rägheit eingeführt  
;-(M)ultiplikator in (V)ervielfacher umbenannt  
;-senden der Fehlerwerte als Positiver Wert (addi64)