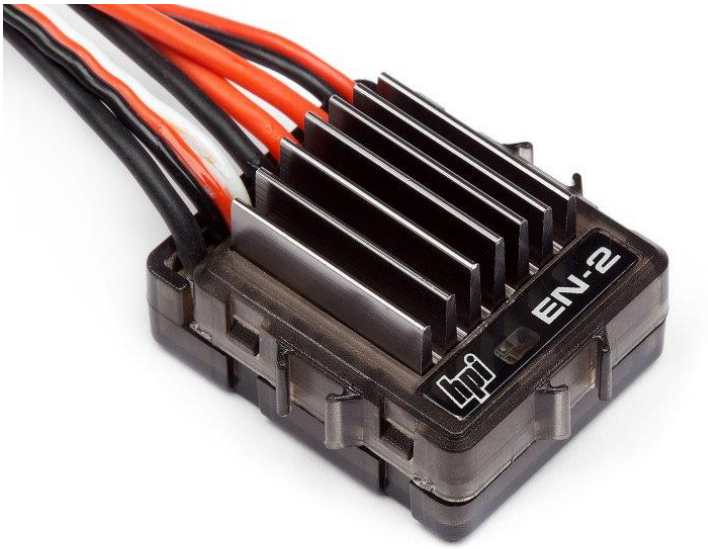
FH | JOANNEUM
Elektronik und
Computer Engineering

Crazy Car Speed Controller
HPI ESC EN-2 Modified
Manual



DI Karl Engelbogen
DI (FH) Markus Krenn, MSc

SW Vers.: 1.12 (the StarWars Edition)
Doc. Vers.: 1.0
Stand: 30.09.2016

Änderungsverzeichnis

Vers. Nr.	Datum	Änderung	Ersteller
1.0	05.08.2016	Erstellen des Dokumentes	Markus Krenn
1.0	30.09.2016	Prüfung auf Korrektheit der Beschreibung	Karl Engelbogen

Inhalt

Abbildungsverzeichnis

1. Allgemein

1.1. Zweck

Der mitgelieferte Speed Controller HPI EN-2 weist ein sehr ungenaues Regelverhalten im unteren Drehzahlbereich auf, der für die Crazy Cars nicht brauchbar ist. Der Speed Controller wurde mit einem neuen Mikrokontroller und einer neuen Software ausgestattet um das Regelverhalten zu verbessern und einige zusätzliche Features einzubauen:

- Verbesserung des Regelverhaltens im unteren Drehzahlbereich
- Ansteuerung des Speed Controllers mit PWM und UART Signal (Standard: PWM)
- Konfiguration einiger Parameter z.B. Max Speed über UART
- Anpassung der Vorwärts/Bremsen/Rückwärts Kurve und Funktionalität

1.2. Überblick

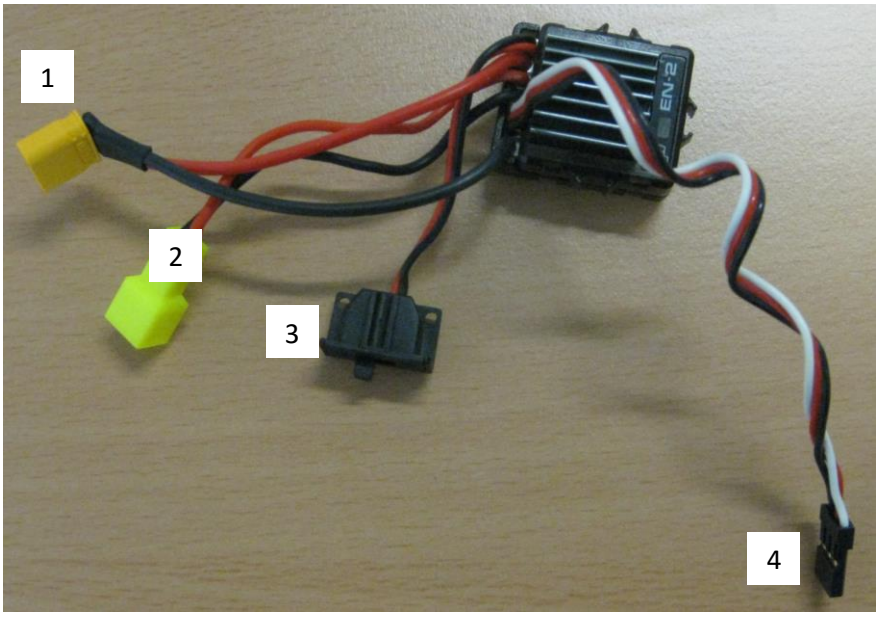


Abbildung 1: HPI EN-2 Speed Controller Beschreibung.

1. Versorgungsstecker
2. Motoranschlusskabeln
3. Ein/Aus Schalter
4. Signal-BEC Stecker

2. Anschluss

2.1. Versorgung

Der Speed Controller wird mittels 2 pol. XT30 Stecker an den Crazy Car Controller oder direkt am Fahrakku angeschlossen.

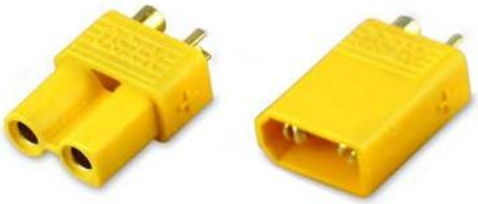


Abbildung 2: 2pol. XT30 Stecker am Speed Controller.

2.2. Signal

Die Ansteuerung des Speed Controllers funktioniert gleich wie beim Original. Über die Signalleitung, vgl. *Abbildung 3: Standard 3pol. Servo/BEC Stecker.*, wird der Speed Controller angesteuert. Das Signal kann bei der modifizierten Version ein Standard PWM sowie ein UART Signal sein.

weiß…	Servo Signal
rot…	BEC Spannung vom ESC
schwarz…	Masse (Akku Minus)

Abbildung 3: Standard 3pol. Servo/BEC Stecker.

3. Betriebsmodi

3.1. Konfiguration/Initialisierung

Der Speed Controller muss zuerst mittels PWM Signal in den jeweils gewünschten Betriebsmodus gebracht werden. Dies geschieht durch ein Signal mit etwa 50Hz und unterschiedlichen Pulsbreiten. Die Frequenz des Signals spielt dabei nur eine untergeordnete Rolle. Wichtig ist, dass die Pulsbreiten eingehalten werden. Dabei werden Impulse kleiner 50µs vom Speed Controller ignoriert. Die Konfigurationspulse müssen mindestens 3 Perioden lang andauern.

PWM Modus
In diesem Modus wird der Speed Controller konventionell mittels PWM Signal gesteuert.
400µs > Konfigurationspuls < 5000µs

UART/Serieller Modus
In diesem Modus wird der Speed Controller mittels UART Signal gesteuert. Nur in diesem Modus können die Konfigurationsparameter des Speed Controllers verändert werden.
100µs > Konfigurationspuls < 300µs

Die Konfiguration des Speed Controllers muss bei jedem Neustart/Reset vollzogen werden. Eine Neukonfiguration während des Betriebs ist nicht möglich.

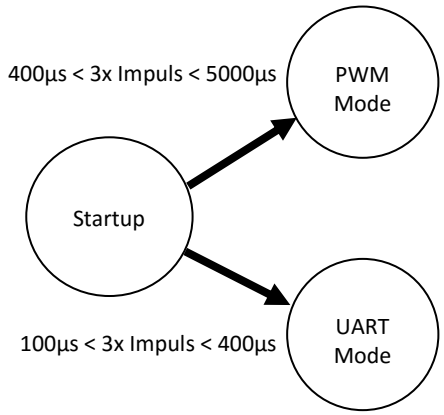


Abbildung 4: Speed Controller – Initialisierung Betriebsmodi State Flow.

3.2. PWM Modus

Wird der Regler im PWM Modus initialisiert, ist der Konfigurationsimpuls gleichzeitig der erste Kalibrierungspunkt für die Rückwärts/Bremsen/Vorwärts Kurvencharakteristik, entspricht also dem *„MaxRPW“* Punkt. Um die Kalibrierung des Speed Controllers abzuschließen, müssen weitere Kalibrierungspunkte eingestellt werden um die Charakteristik - fettgedruckte Linie - zu kalibrieren, vgl. *Abbildung 5: Speed Controller Ansteuercharakteristik.*

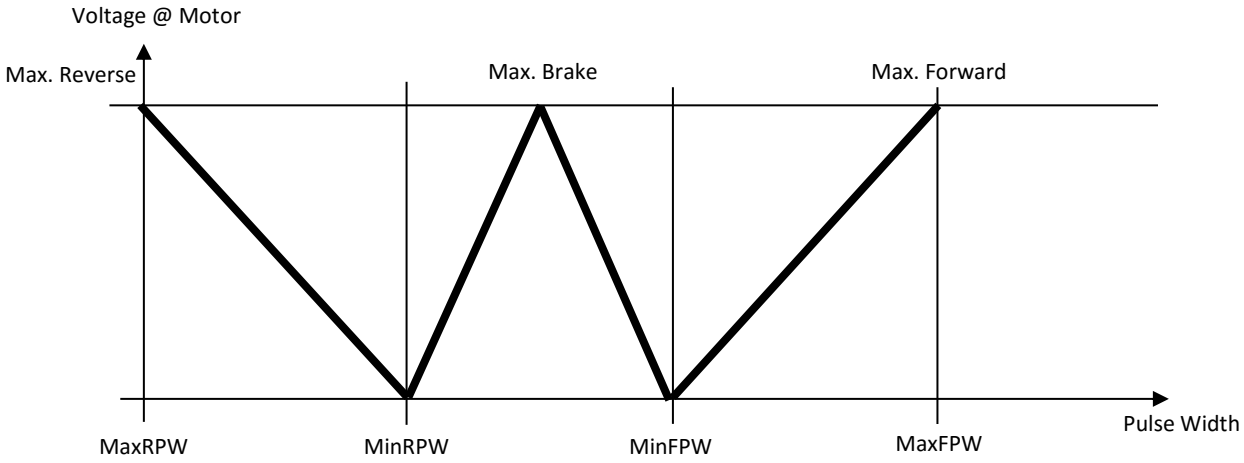


Abbildung 5: Speed Controller Ansteuercharakteristik.

Jeder dieser Punkte entsprechen einer Pulsbreite und müssen eingestellt sowie mindestens 128 Perioden lang gehalten werden. Um bei angeschlossenem Motor die akustische Bestätigung zu erhalten, muss die Pulsbreite mindestens 140 Perioden lang gehalten werden. Wurden diese 4 Punkte kalibriert, muss die Pulsbreite zwischen MinRPW und MinFPW (Bremsbereich) eingestellt werden, um den Kalibrierungsvorgang zu beenden.
Bei vollständig durchgeführter Kalibrierung gibt der Regler eine Reihe an Signaltönen als Bestätigung aus. Die Punkte Max. Reverse, Max. Brake und Max. Forward können im UART Modus konfiguriert werden. Im Auslieferungszustand besitzen diese folgende Werte:

- Max. Reverse: 50% der maximalen Motordrehzahl
- Max. Brake: 100% der maximalen Bremswirkung
- Max. Forward: 50% der maximalen Motordrehzahl

Die Auflösung der Motorsteuerung besitzt 16 Bit. Diese kann nach Belieben für jeden Bereich über den Unterschied der Pulsbreiten und die maximalen Motorwerte eingestellt werden. Die geringste Pulsbreite bzw. den geringsten Pulsbreitenunterschied, die der Speed Controller erfassen kann, ist 1µs. Der Abstand zwischen den einzelnen Kalibrierungspunkten muss mindestens 200µs groß sein.

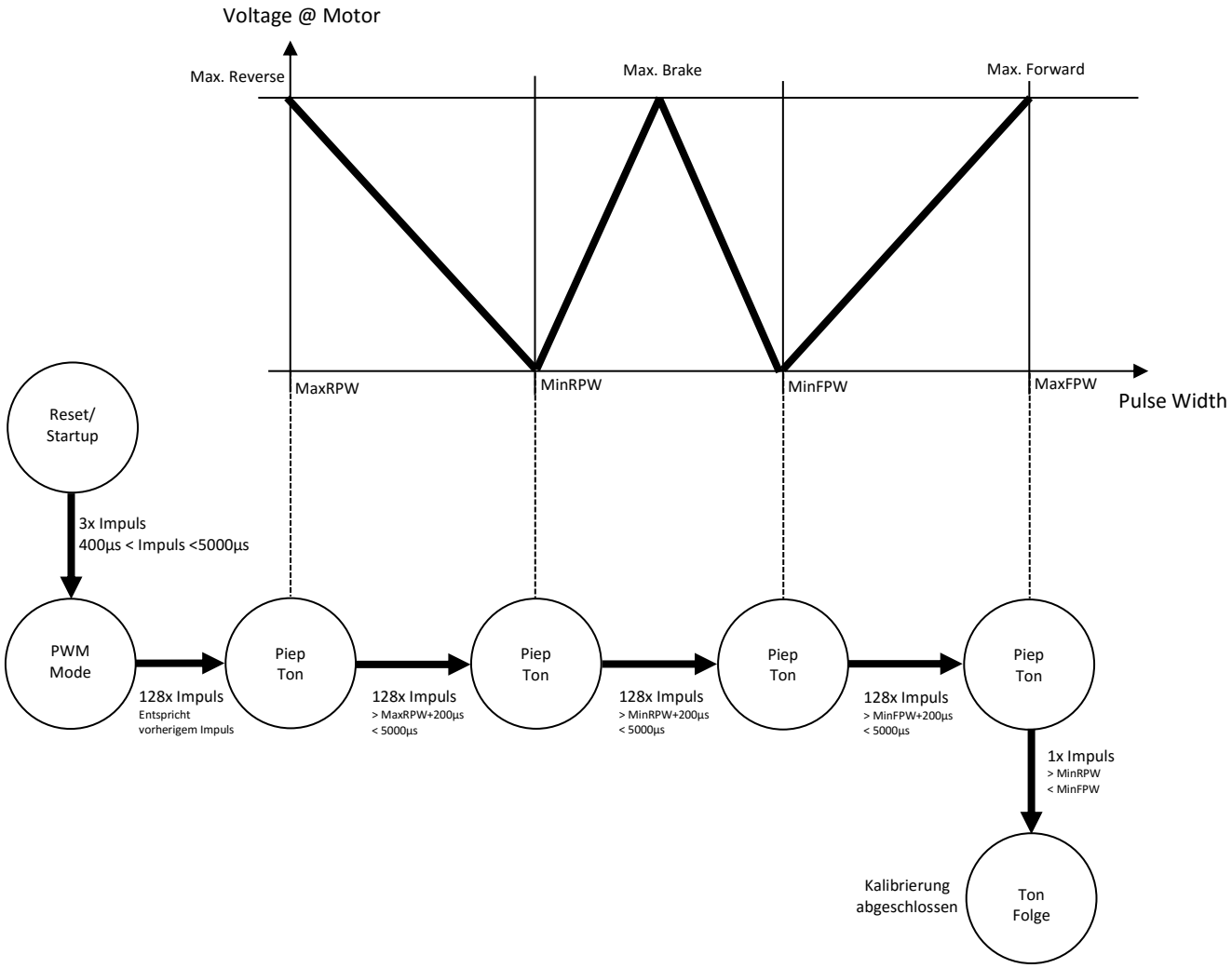


Abbildung 6: Speed Controller – PWM Signal Kalibrierungsvorgang State Flow.

Beispiel:

MaxRPW…	500µs
MinRPW…	1000µs
MinFPW…	1500µs
MaxFPW…	2000µs
Frequenz…	60Hz

3.3. UART/Serieller Modus

In diesen Modus muss nach dem Konfigurationsvorgang das Signal von PWM auf UART umgestellt werden. Zum Abschluss der Initialisierung des seriellen Modus, muss dem Speed Controller eine „hello“ Nachricht gesendet werden. Wurde diese erfolgreich übertragen, nimmt der Speed Controller serielle Befehle entgegen.

Befehlsaufbau:

<STX><Parameter><4 Bytes Werte><CRC8><ETX>

STX, ETX… Framerkennung
Parameter… 1 Byte (siehe Parameterliste)
Werte… 4 Bytes
CRC… 1 Byte, 8 Bit CRC Checksumme (Parameter + 4 Bytes Werte)

Bsp. “hello” Nachricht (1 Byte/Zeichen)

STX h e l l o CRC ETX

Ist der Konfigurationsvorgang beendet, nimmt der Speed Controller Nachrichten entgegen. Folgende Tabellen beschreiben die Steuerungs- und Konfigurationsparameter. Konfigurationsparameter werden im Speed Controller dauerhaft gespeichert.

Steuerungsparameter

Kennung	Default	Max.	Beschreibung
f	0x00	0xFFFF	PWM Wert für Forward … im Serial Mode
b	0x00	0xFFFF	PWM Wert für Brake … im Serial Mode
r	0x00	0xFFFF	PWM Wert für Reverse … im Serial Mode

Bsp. Maximal Vorwärts - 1 Byte/Zeichen

STX f F F F F CRC ETX

Konfigurationsparameter

Kennung	Default	Max.	Beschreibung
F	0x0FFF	0xFFFF	Max. PWM Wert Forward
B	0x01FF	0xFFFF	Max. PWM Wert Brake
R	0x01FF	0xFFFF	Max. PWM Wert Reverse
A	200	0xFFFF	Minimale Zeit in µs die sich zwei PWM Kalibrierungspunkte unterscheiden müssen.
C*	0	0xFFFF	Overtemperature 0 … deaktiviert overtemperatur protection 560 … empfohlener overtemperatur Erkennungswert
D*	650	0xFFFF	Overtemperature Release Temperatur 0 … deaktiviert overheat release, ESC bleibt im Error Mode bis zum Reset 650… empfohlener Overtemperature Release Erkennungswert
E *	1	0x001	Current Limit Pin Handling 0 … Current limit pin wird ignoriert 1 … Error bei Current Limit
G*	1	0x001	Current Limit Pin Release 0 … no Release, bleibt im Current Limit Pin Fehlermode solange bis ESC resetet wird 1 … ESC geht wieder in normalen Modus über wenn Current Limit Pin nicht mehr aktiv
H*	0	0xFFFF	Current Limit Pin activation Time Zeit in ms die der Current Limit Pin aktiv sein muss, damit dies als Current Limit Pin Fehler gewertet wird.
J*	1000	0xFFFF	Timeout im Serial Mode in ms 0 … disables timeout
I*	200	0xFFFF	Timeout im PWM Mode in ms 0 … disables timeout Achtung: Bei Deaktivierung des Timeouts, wird bei Signalverlust der ESC Vollgas geben!
K*	100	0xFFFF	Time of Confirmation peep Zeit in ms für den Confirmation peep bei Parameter-änderungen
L	100	0xFFFF	Serial Time Min. Minimale Grenze in µs zur Erkennung des Seriellen Modus
M	300	0xFFFF	Serial Time Max. Maximale Grenze in µs zur Erkennung des Seriellen Modus
N	400	0xFFFF	PWM Time Min. Minimale Grenze in µs zur Erkennung des PWM Modus
O	5000	0xFFFF	PWM Time Max. Maximale Grenze in µs zur Erkennung des PWM Modus
P	50	0xFFFF	Minimale Impulszeit in µs im Reset State. Alle Impulse kleiner als dieser Parameter werden ignoriert.
Q*	0x800C	0xFFFF	Baudrate für den Seriellen Modus höchste Bit des Parameters entspricht dem U2X (Double the USART Transmission Speed) Flag. Die niedrigeren Bits entsprechen dem UBRR Register des µC. Die passenden Einstellungen für eine gewünschte Baudrate bei 8MHz kann hier gefunden werden: http://wormfood.net/avrbaudcalc.php Anmerkung: Defaultwert =76.8k. Bei Wahl einer anderen Baudrate den maximalen Fehler berücksichtigen.
S*	0	0x001	Timeout peep Hier kann man einstellen ob im Fehlerfall „Timeout“ ein Fehler akustisch signalisiert werden soll. 0 … kein Piep 1 … Piep ist aktiviert
T*	128	0x00FF	Motor PWM Max Duty Cycle Forward Maximaler Duty Cycle im PWM Forward Mode. Wobei der Wert 255 einem Duty Cycle von 100% entspricht.

U*	255	0x00FF	Motor PWM Max Duty Cycle Brake Maximaler Duty Cycle im PWM Brake Mode. Wobei der Wert 255 einem Duty Cycle von 100% entspricht.
V*	128	0x00FF	Motor PWM Max Duty Cycle Reverse Maximaler Duty Cycle im PWM Reverse Mode. Wobei der Wert 255 einem Duty Cycle von 100% entspricht.
W*	1	5	Prescaler für Motor PWM Forward: 1 … $f_{CPU}/1$ 2 … $f_{CPU}/8$ 3 … $f_{CPU}/64$ 4 … $f_{CPU}/256$ 5 … $f_{CPU}/1024$
X*	1	5	Prescaler für Motor PWM Brake: 1 … $f_{CPU}/1$ 2 … $f_{CPU}/8$ 3 … $f_{CPU}/64$ 4 … $f_{CPU}/256$ 5 … $f_{CPU}/1024$
Y*	1	5	Prescaler für Motor PWM Reverse: 1 … $f_{CPU}/1$ 2 … $f_{CPU}/8$ 3 … $f_{CPU}/64$ 4 … $f_{CPU}/256$ 5 … $f_{CPU}/1024$

Die PWM Frequenz des Motors kann wie folgt berechnet werden:

$$f_{PWM} = \frac{255 \cdot f_{CPU}}{2 \cdot MaxDutyCycle \cdot (G_{High} - G_{Low}) \cdot Prescaler}$$

f_{PWM} …	PWM Frequenz (bestimmt auch die PWM Auflösung)
f_{CPU} …	CPU Frequenz (8000000Hz)
Prescaler …	konfigurierbarer Parameter (1, 8, 64, 256, 1024)
G_{HIGH} …	obere Impulsdauer Grenze in µs
G_{LOW} …	untere Impulsdauer Grenze in µs
MaxDutyCycle …	maximaler duty cyle (Konfigurierbarer Parameter: 0 bis 255 entspricht 0 – 100%)

Beispiel:

1. MinRPW 500 µs
2. MaxRPW 800µs
3. MinFPW 1100 µs
4. MaxFPW 2100 µs
5. Prescaler = 8
6. MaxDutyCycle = 128

$$f_{PWM} = \frac{255 \cdot f_{CPU}}{2 \cdot MaxDutyCycle \cdot (G_{High} - G_{Low}) \cdot Prescaler} = \frac{255 \cdot 8000000}{2 \cdot 127 \cdot (2100 - 1100) \cdot 8} = 1004$$

4. Fehler

Der Speed Controller besitzt verschiedene Fehlerfälle die bei angeschlossenem Motor als Piep Töne ausgegeben werden. Je nach Fehlerart wechselt der Speed Controller nach Wegfall des Fehlers wieder in den konfigurierten Betriebsmodi zurück. Folgende Tabelle beschreibt die Fehlercodes:

Fehlerbeschreibung

Fehler		# Peeps	Reversibel
TIMEOUT	Kein Inputsignal über einen bestimmten Zeitraum Das Errorsignal kann deaktiviert werden	1	JA
OVERHEAT	Der Temperaturwert ist unter den Schwellwert (→ Parameter) gesunken. Dieser Fehlerfall lässt sich deaktivieren.	2	JA/NEIN
CURRENTLIMITPIN	Der CurrentLimitPin ist auf LOW gegangen. Dieser Fehlerfall lässt sich deaktivieren.	3	JA/NEIN
SERIAL_HELLO	ESC hat kein „hello“ Empfangen. Es werden die ersten 5 gültigen Nachrichten geprüft, erst dann wird der Fehler ausgelöst.	4	NEIN
UNKNOWN_MODE	Es war nicht möglich über die Initialisierungsimpulse im Detection state zu erkennen welcher Betriebsmodus gewünscht ist	5	NEIN
FLASH	Der Flashspeicher konnte trotz mehrmaligen Versuch nicht fehlerfrei beschrieben werden	6	NEIN
UNKNOWN	Unbekannt Fehler	7	NEIN

5. Appendix A – CRC8 Calculation

```
/* CRC-8 Calculation
=================
erstellt von: Kurt Moraw (Juli 2009) www.helitron.de
Grundlagen zu diesen Funktionen wurden der Webseite: http://www.cs.waikato.ac.nz/~312/crc.txt
entnommen (A PAINLESS GUIDE TO CRC ERROR DETECTION ALGORITHMS)
Das Ergebnis wurde geprueft mit dem CRC-Calculator: ttp://www.zorc.breitbandkatze.de/crc.html
Das Generator-Polynom wurde wie folgt gewaehlt:
Polynom = x^8+x^7+x^6+x^4+x^2+1 = x^8+x^7+x^6+x^4+x^2+X^0
das entspricht: (1) 1101 0101 = Hex d5 bis zu einer maximalen Bitlaenge von 93 entspricht die
Datensicherungsfunktion der Hamming-Distanz 4
*/

/* crc8_bytecalc
diese Funktion enthaelt die CRC8 Schleife fuer 8 Bit einer Nachricht.
Sie wird solange aufgerufen bis die komplette Nachricht verarbeitet wurde,
danach muss sie nochmals mit dem Wert 0 aufgerufen werden um die CRC-8 Berechnung
abzuschliessen

Parameter: byte ... ein Byte der Nachricht
Return: aktueller Wert des CRC-8
*/

unsigned char reg = 0;// Rechen-Register fuer den CRC Wert mit Initial Value 0

unsigned char crc8_bytecalc(unsigned char byte)
{
	int i;                          // Schleifenzaehler
	char flag;                      // flag um das oberste Bit zu merken
	unsigned char polynom = 0xd5; // Generatorpolynom

	// gehe fuer jedes Bit der Nachricht durch
	for (i = 0; i < 8; i++)
	{
		if (reg & 0x80)
			flag = 1;
		else
			flag = 0;      // Teste MSB des Registers

		reg <<= 1;             // Schiebe Register 1 Bit nach Links und
		if (byte & 0x80)
			reg |= 1;// FÃ¼lle das LSB mit dem naechsten Bit der Nachricht auf
		byte <<= 1;      // nÃ¤chstes Bit der Nachricht
		if (flag)
			reg ^= polynom;       // falls flag==1, dann XOR mit Polynom
	}
	return reg;
}

/* crc8_messagecalc

 Ruft crc8_bytecalc solange auf bis die Nachricht vollstaendig abgearbeitet wurde, danach
 wird crc8_bytecalc noch einmal mit Wert 0 aufgerufen um die CRC-8 Berechnung abzuschliessen

 Parameter:
 msg ... Bytearray welches die Nachricht enthaelt
 len ... Laenge der Nachricht
 Return: CRC-8 der kompletten Nachricht
 */

unsigned char crc8_messagecalc(unsigned char *msg, int len)
{
	reg = 0;
	int i;
	for (i = 0; i < len; i++)
		crc8_bytecalc(msg[i]);        // Berechne fuer jeweils 8 Bit der Nachricht
	return crc8_bytecalc(0);          // die Berechnung muss um die Bitlaenge des
                                      // Polynoms mit 0-Wert fortgefuehrt werden
}
```