C-Control/C-Cross-Compiler Handbuch

Oliver Haag

13. Juli 2006

Zusammenfassung

Dieses Handbuch soll als Einführung in die Programmierung mit dem C-Control/C-Cross-Compiler dienen. Vorausgesetzt werden Kenntnisse in der C-Programmierung, da hier nur auf die Besonderheiten gegenüber ISO-C eingegangen wird.

Inhaltsverzeichnis

1 Grundlagen

Im Compiler-Packet sind die folgenden Werkzeuge enthalten.

1.1 Compiler

Der Compiler wird über die Kommandozeile mit Hilfe von Kommandozeilenparametern gesteuert. Die Syntax schaut folgendermaßen aus:

```
bin/ccccc [Optionen] Datei
```

Optionen:

-a *Datei*	Assemblerdatei einbinden (Nicht zusammen mit -h und -s verwendbar)
-i *Pfad*	Pfad zu den Include-Dateien
-l	Logbuchdatei erstellen
-o *Datei*	Name der Ausgabedatei
-c	Programm und Assemblerprogramm im C-Control dat-Format
-h	Ausgabe im Intel Hex-Format
-s	Ausgabe im Motorola S-Records Format

Datei: Name der Quelltextdatei

1.2 Data Loader

Mit dem Data Loader kann die vom Compiler ausgegebene Datei oder eine Assemblerausgabe in die C-Control geladen werden. Es ist auch möglich Daten auf der C-Control auszulesen und als Datei zu speichern.
Bis jetzt ist der Data Loader jedoch erst für Linux verfügbar.
Die Syntax des Data Loaders ist wie folgt:

```
bin/ccdl [Optionen] Datei
```

Optionen:

-d *Datei*	Genutzer Port (z.B. /dev/ttyS0)
-b *Zahl*	Baudrate (Standard: 9600, für 12 MHz Quarz auf 28800 stellen)
-g	Daten aus der C-Control auslesen
-s	Daten in die C-Control schreiben
-c	Programm und Assemblerprogramm im C-Control dat-Format
-a	Assemblerprogramm im internen EEPROM
-p	Programm im externen EEPROM
-f	Inhalt der Datei auf dem externen EEPROM
-i	Intel Hex-Format
-m	Motorola S-Records Format

Datei: Name der Ein-/Ausgabedatei

2 Programmiersprache

2.1 Präprozessor

Es gibt bereits die meisten Präprozessorbefehle die in ISO-C festgelegt sind, die fehlenden werden in späteren Versionen des Compilers verfügbar sein.

2.1.1 #define

Definiert Konstanten grundsätzlich wie in ISO-C. Der Text vom Ende des Definitionsnames bis zum Zeilenende wird beim compilieren an alles Stellen im Quelltext eingefügt wo der Definitionsname vorkommt. Falls ein Kommentar folgt wird nur der Text bis zum Anfang den Kommentars eingefügt.

Es sind bereits folgende Konstanten standardmäßig definiert:

false		0
true		-1
__CCCCC__	[Version]	Version: Compilerversion, Hexadezimal (z.B. 1.0.13.2 = 0x10d2)
__C_CONTROL__	[Version]	Version: C-Control-Betriebsystemversion (z.B. 1.1 = 0x1100)

2.1.2 #error

Gibt eine Fehlermeldung zurück wenn die Zeile erreicht wird.

2.1.3 #ifdef

Übersetzt die folgenden Zeilen bis zu #endif oder #else (bzw. #elif usw.) nur wenn die folgende Definition definiert wurde.

2.1.4 #ifndef

Übersetzt die folgenden Zeilen bis zu #endif oder #else (bzw. #elif usw.) nur wenn die folgende Definition nicht definiert ist.

2.1.5 #else

Übersetzt die folgenden Zeilen bis zu #endif nur wenn die vorige Bedingung nicht zutraf.

2.1.6 #elifdef

Übersetzt die folgenden Zeilen bis zu #endif oder #else (bzw. #elif usw.) nur wenn die vorige Bedingung nicht zutraf und die folgende Definition definiert wurde.

2.1.7 #elifndef

Übersetzt die folgenden Zeilen bis zu #endif oder #else (bzw. #elif usw.) nur wenn die vorige Bedingung nicht zutraf und die folgende Definition nicht definiert ist.

2.1.8 #include

Bindet Header-Dateien ein wie in ISO-C.

2.1.9 #undef

Macht die Definition der folgenden Definition rückgängig.

2.2 Variablen

Bei den Variablen gibt es einige Unterschiede zum ISO-C.

2.2.1 Typen

Auf der C-Control gibt es folgende Variablentypen:

- bool: Nimmt die Werte Falsch = 0 (0x0000) und Wahr = -1 (0xFFFF) an und belegt nur 1 Bit im RAM der C-Control.

- char: Wertebereich von 0 bis 255, belegt 1 Byte im RAM der C-Control.
- int: Wertebereich von -32768 bis 32767, belegt 2 Byte im RAM der C-Control.

2.2.2 Deklaration

Bei der Deklaration muss, anderst als im ISO-C, nach dem Variablentyp die Speicheradresse in eckigen Klammern angegeben werden.
Dabei ist die Einheit der Speicheradresse immer abhänig vom Variablentyp. Also bei bool 1 Bit, bei char 1 Byte und bei int 2 Byte. Werden mehrere Variablen mit einem Deklarationsbefehl deklariert wird für die folgenden immer der Speicherplatz nach der vorigen Variable genommen.

Beispiele für gültige Deklarationen:

```
char[2] Zahl;                          // Die Variable Zahl vom Typ char an der Adresse 2
                                       // (16. - 23. Bit) festlegen
int[4] Integer = 1000;                 // Die Variable Integer vom Typ int an der Adresse 4
                                       // (64. - 79. Bit) festlegen und ihr den Wert 1000
                                       // zuweisen
bool[12] Ein1, Ein2 = true, Ein3       // Die Variablen Ein1, Ein2 und Ein3 vom Typ bool an
                                       // den Adressen 12-14 (12. - 14. Bit) festlegen und
                                       // Ein2 den Wert Wahr zuweisen
```

2.3 Funktionen

Die Funktionsdeklaration funktioniert bis auf kleine Unterschiede gleich wie in ISO-C.

2.3.1 Argumente

Bei den Parametern müssen, wie bei der Variablendeklaration, nach den Variablentypen die Speicheradressen angegeben werden.

Beispiel für ein kleines Programm:

```
void Funktion(bool[191] Parameter);

void main()
{
  Funktion(true);
}

void Funktion(bool[191] Parameter)
{
  outport(1, Parameter);
}
```

2.3.2 Funktionstypen

Bis jetzt sind normale Funktionen und Interrupts implementiert, inline-Funktionen sind bereits geplant.

interrupt

Wenn eine Interruptfunktion mit setinterrupt festgelegt werden soll muss diese vom Typ interrupt sein.

Hier ein Beispiel für die Verwendung eines Interrupts:

```
interrupt Interruptroutine();

char[0] Wert = 0;

void main()
{
        setinterrupt(&Interruptroutine);

        while(true)
        {
                outportb(0, Wert);
        }
}

interrupt Interruptroutine()
{
        Wert = inportad(0);
}
```

2.4 Formeln

Formeln sind wie in ISO-C aufgebaut, es gibt jedoch ein paar Unterschiede.

2.4.1 Werte

Zahlen

Zahlen können folgendermaßen angegeben werden:

- Dezimal: 7416, -26385
- Hexadezimal: 0x1cf8, 0x98ef, -0x6711
- Binär: 0b1110011111000, 0b1001100011101111, -0b110011100010001

Variablen und Funktionen

Variablen und Funktionen werden genauso wie in ISO-C verwendet.

Adressen

Die Adresse einer Variablen oder Funktion wird durch ein kaufmännisches Und (&) und den Namen der Variablen bzw. Funktion ermittelt, z. B. &Variable, &Funktion.

2.4.2 Operatoren

Bei den Operatoren gibt es ein paar Unterschiede zu ISO-C, die Rangfolge ist jedoch sehr ähnlich.

12.	()	Gruppierung
11.	~	bitweise Negation
	!	logische Negation
	+ -	Vorzeichen
10.	* / %	Multiplikation, Division, Rest
9.	+ -	Summe, Differenz
8.	<< >>	Bitverschiebung
7.	< > <= >=	Relationen
6.	== !=	Gleichheitstest
5.	& !&	bitweises Und, bitweises Nicht-Und
4.	∧	bitweises exklusives Oder
3.	| !|	bitweises Oder, bitweises Nicht-Oder
2.	&&	logisches Und
1.	|| ∧∧	logisches Oder, logisches exklusives Oder

2.4.3 Formeloptimierung

Seit Version 0.2.1.0 ist auch ein Formeloptimierer integriert. Er rechnet konstante Formelteile (d. h. ohne Variablen oder Funktionen) bereits beim compilieren aus. Somit wird der Code beschleunigt und Speicherplatz gespart. Wenn die komplette Formel konstant ist kann sie auch in vordefinierten Funktionen, die konstante Parameter benötigen, und in allen anderen Fällen wo konstante Werte benötigt werden (z. B. Variablenadressen) verwendet werden. Adressen von Funktionen sind zwar auch konstant aber können aus technischen Gründen nicht optimiert werden.
Bei der Optimierung wird der Rechenstack der C-Control simuliert und bei Rechenoperationen überprüft ob die benötigten Stackwerte konstant sind.

Hier ein paar Beispiele mit den optimierten Versionen der Formeln als Kommentar:

```
a = 8 * 5 + 7 * (23 - 13) / 5 - 2       // a = 52
a = 5 * 3 + 17 % 5 / b;                 // a = 8 + 2 / b
a = f(27) * (5 + 2) / 2;                // a = f(27) * 5 / 2
a = 9 / 3 * c * 7 / 3;                  // a = 3 * c * 7 / 3;
a = 4 * &d + 3 * &f;                    // a = 20 + 3 * &f;
```

Die erste Formel ist komplett konstant und wird daher auch komplett aufgelöst.
Die zweite Formel hat die Variable ganz am Ende, daher wird sie so weit wie möglich optimiert.
Die dritte Formel hat die Funktion gleich an erster Stelle, somit kann nur die von der Priorität höher liegende Klammer aufgelöst werden. Dies hat jedoch Vorteile weil der Programmierer warscheinlich erreichen wollte, dass das Funktionsergebnis mal 3,5 genommen wird. Dies ist ohne diesen Trick nicht möglich, da die C-Control keine Kommazahlen kennt.
Die vierte Formel hat die Variable in der Mitte, wodurch die 9 / 3 davor noch optimiert werden.
Die letzte Formel zeigt dass bei Funktionsaddressen keine Optimierung möglich ist. Bei Variablen geht dies, die Adresse von d ist hier 5.

3 Vordefinierte Funktionen

Folgende Funktionen sind vordefiniert und werden direkt in C-Control-Tokens übersetzt:

3.1 Portoperationen

bool inport(const char port)
Gibt den Zustand des Ports *port* als Boolean zurück.

char inportb(const char port)
Gibt den Zustand von 8 Ports als Char zurück.

char inportw()
Gibt den Zustand von allen 16 Ports als Integer zurück.

void outport(const char port, bool status)
Setzt den Status des Ports *port* auf *status*.

void outportb(const char port, bool status)
Setzt den Status von 8 Ports auf *status*.

void outportw(bool status)
Setzt den Status von allen 16 Ports auf *status*.

char inportad(const char port)
Liest den Wert des Analogeingangs *port* aus und gibt ihn als Char zurück.

void outportpwm(const char port, char value)
Setzt den Wert des Analogausgangs *port* auf value.

void invport(const char port)
Invertiert den Port *port*.

void pulseport(const char port)
Gibt einen kurzen Puls auf dem Port *port* aus (2x invertieren, 34 Taktzyklen lang invertiert = 8,5 μs).

void deactport(const char port)
Deaktiviert den Port *port*.

void deactportb(const char port)
Deaktiviert 8 Ports.

void deactportw()
Deaktiviert alle Ports.

3.2 Mathematische Funktionen

int abs(int number)
Gibt den Absolutwert (d. h. ohne Vorzeichen) von *number* als integer-Variable zurück.

int sqrt(int number)
Gibt die Quadratwurzel aus *number* als integer-Variable zurück.

bool sgn(int number)
Gibt das Vorzeichen von *number* als Boolean zurück (false = positiv, true = negativ).

void randomize()
Startet den Zufallsgenerator neu.

int rand()
Gibt eine 16-Bit Zufallszahl als integer-Variable zurück.

3.3 Zeit

void delay(int time)
Unterbricht die Programmausführung für $time \cdot 20ms$.

char getyear()
Gibt das Jahr als Char zurück (Zweistellig, z.B. 06).

char getmon()
Gibt den Monat als Char zurück.

char getday()
Gibt den Tag des Monats als Char zurück.

char getdow()
Gibt den Wochentag als Char zurück.

char gethour()
Gibt die Stunde als Char zurück.

char getmin()
Gibt die Minute als Char zurück.

char getsec()
Gibt die Sekunde als Char zurück.

int gettimer()
Gibt den Zählerstand den internen Timers als integer-Variable zurück.

void setyear(char year)
Setzt das Jahr auf *year*.

void setmon(char month)
Setzt den Monat auf *month*.

void setday(char day)
Setzt den Tag des Monats auf *day*.

void setdow(char dow)
Setzt den Wochentag auf *dow*.

void sethour(char hour)
Setzt die Stunde auf *hour*.

void setmin(char minute)
Setzt die Minute auf *minute*.

void setsec(char second)
Setzt die Sekunde auf *second*.

3.4 Tonausgabe

void sound(int frequency)
Gibt einen Ton mit einer Frequenz von $250.000 \div frequency$ (Bei 4 MHz) aus.

void nosound()
Schaltet den Ton wieder ab.

3.5 Serielle Schnittstelle

void rs_baud(const char baud)
Setzt den BAUD-Register auf *baud*. In sys/baud.h sind Konstanten für *baud* definiert.

bool rs_avail()
Gibt true zurück wenn ein Byte über die serielle Schnittstelle empfangen wurde, sonst false.

void rs_send(char byte)
Sendet das Byte *byte* über die serielle Schnittstelle.

void rs_sendint(int number)
Sendet die Zahl *number* im ASCII-Format über die serielle Schnittstelle.

void rs_sendstr(const char[] string)
Sendet die Zeichenkette *string* über die serielle Schnittstelle.

char rs_recieve()
Empfängt ein Byte von der seriellen Schnittstelle und gibt es als Char zurück.

int rs_recieveint()
Empfängt eine Zahl von der seriellen Schnittstelle und gibt sie als Integer zurück.

void rs_handshake(bool activated)
Aktiviert bzw. deaktiviert den Hardware-Handshake (Schauen ob Gegenseite bereit ist) der seriellen Verbindung.

bool rs_ready()
Überprüft ob die Gegenseite bereit ist Daten zu empfangen.

3.6 Systemfunktionen

void sys(const int offset, ...)
Ruft die Assemblerroutine an der Adresse *offset* auf und speichert die folgenden Werte in den Rechenstack.

void slowmode(const bool enabled)
Schaltet den Slowmode (1/16 der normalen Geschwindigkeit) ein oder aus.

void setinterrupt(const int address)
Schaltet den IRQ-Interrupt ein und setzt die Adresse der Interruptfunktion auf address.

void clearinterrupt()
Schaltet den IRQ-Interrupt aus.

char getregister(const char register)
Gibt den Inhalt des Registers *register* der C-Control zurück. *register* muss aufgrund des C-Control-Betriebsystems leider mindestens 1 sein.

char setregister(const char register, char value)
Setzt den Inhalt des Registers *register* der C-Control auf value. *register* muss aufgrund des C-Control-Betriebsystems leider mindestens 8 sein.

4 Header-Dateien

Hier sind alle standardmäßig beiliegenden Header-Dateien aufgelistet, die mit `#include <Dateiname>` eingebunden werden können.

4.1 ccrp5.h

ccrp5.h enthält Funktionen und Definitionen für den Robby RP5 ohne Basiserweiterung.

4.1.1 Funktionen

void acs_interrupt(char frequency)
Aktiviert das ACS-Interrupt-Signal des Subsystems, das dann alle $frequency \cdot 4ms$ ein Signal an den IRQ-Pin des Microcontrollers sendet.

void acs_power(char level)
Stellt die ACS-Stärke ein (ACSLO = ca. 30 cm, ACSHI = ca. 60 cm, ACSMAX = ca. 100 cm).

void dst_clear()
Löscht die Wegstreckenzähler.

int dst_counter(bool side)
Liest einen Wegsteckenzähler aus und gibt ihn als Integer zurück. Das Argument *side* gibt an, welcher Wegstreckenzähler ausgelesen werden soll (LEFTDST = Linker Zähler, RIGHTDST = Rechter Zähler).

void initrp5()
Initialisiert den Robby und sollte immer am Anfang eines Robby-Programmes stehen.

void ir_mode(char mode)
Setzt die IR-Einstellungen auf *mode*. Die Flags sind das IR-Protokoll (RC5 oder REC80) und der IR-Interrupt (IRINT). `ir_mode(RC5 | IRINT)` setzt also das Protokoll auf RC5 und schaltet den Interrupt an.

int ir_recieve()
Gibt ein empfangenes IRSignal zurück. Die Adresse ist das High-Byte der zurückgegebenen integer-Variable (ir_recieve() $\gg$ 8) und der Befehl das Low-Byte (ir_recieve() && 0x00ff).

void ir_send(char address, char command)
Sendet ein IR-Signal. *address* ist dabei die Adresse und *command* der Befehl.

void leds(char mask, char state)
Steuert die LED's auf der Basisplatine. *mask* gibt an, welche LED's verändert werden sollen und *state* gibt an, welche LED's davon angeschaltet werden sollen. `leds(LED1 | LED2 | LED4, LED2 | LED 4)` ändert also LED 1, 2 und 4 und schaltet dabei LED 1 aus und LED 2 und 4 an. LEDX beeinflusst alle 4 LED's.

void setextport(char data)
Setzt die 8 Erweiterungsports auf den Wert *data* (Wie ein Byteport)

char subsys_getmode()
Liest die Modus-Flags des Subsystems aus und gibt sie als Char zurück.

char subsys_getstate()
Leist die Status-Flags des Subsystems aus und gibt sie als Char zurück. Dies sind SS_ACSL und SS_ACSR für das ACS und SS_GOTIR, welches anzeigt, ob ein IR-Signal empfangen wurde.

void subsys_power(bool activated)
Schaltet das Subsystem ein oder aus.

4.1.2 Definitionen

Sensoren (Mit inportad(*Sensor*) auslesen):

SYSVOLTAGE	Die Spannung, die am Robby anliegt
SYSCURRENT	Der Stromverbrauch des Robby
CHARGECURRENT	Der Ladestrom
LIGHTL	Der linke Lichtsensor
LIGHTR	Der rechte Lichtsensor
MICROPHONE	Das Mikrofon
TOUCH	Der Touchsensor

Motoren:

outportpwm(*Motor*, *Geschwindigkeit*): Setzt den Motor (SPEEDL für linken, SPEEDR für rechten) auf die Geschwindigkeit.
outport(*Motor, Richtung*): Setzt die Richtung (FWD für vorwärts, REV für rückwärts) des Motors (DIRL für linken, DIRR für rechten).

4.2 ccrp5be.h

ccrp5be.h enthält erweiterte Funktionen und Definitionen für den Robby RP5 mit Basiserweiterung. Hier werden nur die Unterschiede zu ccrp5.h erklärt.

void eleds(char mask, char state)
Steuert die LED's der Erweiterungsplatine. Die Ansteuerung ist gleich wie bei der Basisplatine (LED-Flags haben vorne aber immer ein E (z. B. ELED1) und gehen bis ELED8).

void setledport(char data)
Setzt die 8 Erweiterungsports der Basiserweiterung auf den Wert *data*. Wenn alle 8 LED's gleichzeitig geändert werden kann auch diese Funktion anstatt eleds verwendet werden, da die Ausführung von eleds mehr Zeit benötigt.

void subsys_power(bool activated)
Hier sind innerhalb der Funktion Unterschiede, da 8 weitere Erweiterungsports vorhanden sind. Die Funktion kann jedoch wie bei ccrp5.h verwendet werden.

4.3 asm/ccrp5.h

Enthält die Adressen der Assemblerroutinen in asm/ccrp5.s19.

COMNAV	command, data	Schickt Befehl *command* an Subsystem und gibt Antwort zurück
EXTPORT	data	Setzt den Extport auf *data*, keinen Rückgabewert

4.4 asm/p5driv.h

Enthält die Adressen der Assemblerroutinen in asm/p5driv.s19.
Diese Header-Datei ist nur aus Kompatiblitätsgründen zu den Conrad-Assemblerroutinen vorhanden, normalerweise sollte jedoch die asm/ccrp5.h verwendet werden.

PLM_SLOW	PLM auf langsam stellen
SYSTEM	Extport ausgeben
COMNAV	COM/NAV-Subsystem Anfrage schicken
REVR	Antrieb Rechts rückwärts
REVL	Antrieb Links rückwärts
FWDR	Antrieb Rechts vorwärts
FWDL	Antrieb Links vorwärts
ROTR	Rechts drehen
ROTL	Links drehen
REV	Rückwärts
FWD	Vorwärts
COMNAV_STATUS	Aktualisiert alle Flags im Status-Register
ACS_LO	ACS Power niedrig
ACS_HI	ACS Power hoch
ACS_MAX	ACS Power maximal
SEND_TLM	Sendet Telemetrie (Channel=hByte,Daten=lByte)
SEND_SPEEDR	Sendet Tlm Kanal 8, PWM Rechts
SEND_SPEEDL	Sendet Tlm Kanal 7, PWM Links
SEND_SYSSTAT	Sendet Tlm Kanal 0, Systemstatus

4.5 sys/68hc05b6.h

In asm/68hc05b6.h sind die Register des Motorola 68HC05B6 Microcontrollers, auf dem das Betriebsystem der C-Control läuft, definiert. Genauere Informationen können direkt aus der Header-Datei entnommen werden da eine komplette Auflistung der Register wohl zu lang wäre und nicht viel Sinn macht da im Allgemeinen die Registerbezeichnungen aus dem Datenblatt des Microcontrollers verwendet wurden.

4.6 sys/baud.h

in sys/baud.h sind die Baudraten, die an rs_baud übergeben werden, für den Betrieb mit dem Standard 4 MHz Quarz und einem 12 MHz Quarz definiert:

4 MHz	B75, B150, B300, B600, B1200, B2400, B4800 und B9600
12 MHz	B7200, B14400 und B28800