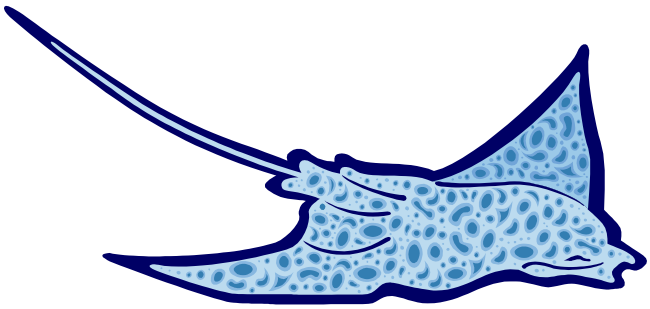
Einsatz von Renesas C für das

ROBO Interface

Ulrich Müller



Inhaltsverzeichnis

Übersicht

Allgemeines

Hier wird eine Möglichkeit zur direkten Programmierung des ROBO Interfaces in Renesas C beschrieben. Programmiert wird in jedem Fall auf dem PC. Da ein Test auf dem ROBO Interface kaum möglich ist, wird hier ein mehrstufiges Vorgehen vorgestellt :

1. **Erstellen** der Anwendung mit einer Teilmenge von VC++ 6.0 auf dem PC
   Testen mit den Debug-Mitteln der IDE von VC++ 6.0.
   Hierzu wird ein VC++ Rahmenprogramm "SimRenesas" fest vorgegeben, in das eine C++ Source mit der eigentlichen Anwendung eingefügt wird. es muß zwingend UCHAR main() {} enthalten. Der Zugriff auf die Interface-Funktionen erfolgt über die Funktionsbibliothek roboFish.H / CPP, die eine klassenlose Version der Klasse CFishFace ist. Sie baut aber direkt auf der FtLib von fischertechnik auf (CFishFace baut auf umFish40.DLL auf). roboFish verfügt wie CFishFace über "RobMotoren" und Counter zur Positionierung in Anzahl Impulsen.
2. **Compilieren** der Anwendung in einer Renesas C Umgebung "RunRenesas".
   RunRenesas enthält die gleiche Funktionsbibliothek roboFish.H / C wie (1.) diesmal in einer Version, die auf FtCComp von fischertechnik aufbaut. Die Extension der Source muß dazu von .CPP in .C geändert werden. Die Source wird mit Project | Add Files... in das Project eingefüft.
   Achtung : Das File ProgNr.INC des Projektes RunRenesaas enthält die Festlegung in welchen Speicherbereich das Programm geladen werden soll. Standard ist RAM1. Bei Bedarf ist das auf FLASH1 bzw. FLASH2 zu ändern.
3. **Laden** in das ROBO Interface mit dem Tool FtLoader aus dem Paket FtCComp UND Starten.

Unterstützt wird das ROBO Interface mit Funk-Betrieb aber ohne Extensions.

Hinweis : Da die C++ Source der Anwendung von (1.) möglichst ohne Änderungen in (2.) eingebracht werden sollten, ist bei der Auswahl der Sprachmenge restriktiv vorzugehen. Als Datentypen werden z.Zt. folgende eingesetzt : UCHAR, UINT, int, BOOL. Die unterschiedliche Datenlänge auf den Systemen stört nicht weiter, da der Bereich von 16bit wohl nicht überschritten werden wird. Die normale C Sprachmenge ist in beiden Umgebungen verfügbar. Weitere include's sind sorgfältig zu überlegen, ebenso weitere Datentypen.

Beteiligte Komponenten

1. VC++ 6.0 von Microsoft.
2. FtLib von www.fischertechnik.de/de/computing/download.html
3. Das VC++ Projekt SimRenesas aus roboFish.ZIP
4. Renesas C über fischertechnik Downloads oder Knobloch
5. FtCComp über fischertechnik Downloads
6. Das Renesas C Projekt RunRenesas aus roboFish.ZIP

Beispiel : Lichtsucher – Hinderniserkennung

Für das Modell aus dem aktuellen ROBO Mobile Set

```
UCHAR main() {
  UCHAR Inputs;
  InitFish();
  do {
    Inputs = GetInputs();
    if((Inputs & 0x0C) == 0x0C) Ausweichen90();
    else if(Inputs & 0x04) AusweichenLinks();
    else if(Inputs & 0x08) AusweichenRechts();
    else if((Inputs & 0xC0) == 0xC0) Vorwarts();
    else if(Inputs & 0x40) Linkskurve();
    else if(Inputs & 0x80) Rechtskurve();
    else Lichtsuche();
  } while(1);
  FinishFish();
  return 0;
}

void Lichtsuche() {
  UINT i;
  Links();
  for(i = 0; i < 10; i++) {
    if(GetInputs() & 0xC0) return;
    WaitForHigh(iImpRechts);
  }
  Rechts();
  for(i = 0; i < 10; i++) {
    if(GetInputs() & 0xC0) return;
    WaitForHigh(iImpRechts);
  }
  Stopp();
}

void AusweichenLinks() {
  Stopp();
  Pause(1000);
  Ruckwarts();
  WaitForChange(iImpRechts, 10, iTasHinten);
  Links();
  WaitForChange(iImpRechts, 2);
  Stopp();
  Pause(500);
}
```

Programm HindernisLicht.C (Ausschnitt)

Die Programmstruktur entspricht dem ROBO Pro Programm. Lediglich die drei Sub's Licht, Hindernis und Fahren im Hauptprogramm wurden aufgelöst, so ist es deutlich übersichtlicher. Anstelle von if((Inputs & 0x0C) können die Hindernistaster auch explizit abgefragt werden : if(GetInput(3) && GetInput(4)), ist einfacher zu durchschauen.

Weitere Details auf Programmieren und Debuggen

Downloads

In roboFish.ZIP sind die Projekte SimRenesas und RunRenesas einschließlich der erforderlichen FtLib
bzw. der .h/.c Files enthalten, nicht aber die Tools.

Die genannten Projekte enthalten jeweils die gleichen Beispiel Projekte :
HanoiRobot(Turm von Hanoi mit Rekursion),
Hindernis2, HindernisLicht, MR2A, MR2B, MR2C, (Mobile Robots in Varianten)
PoorSlave(Funk),
Stanze (nach eTec Module)

Weitere Details in www.ftcomputing.de/pdf/ccFish40e.PDF

Referenz

Bezeichnungen

AnalogNr	Nummer des Analog-Einganges (AX = 1, AY = 2)
DistanceMode	Modus in dem die D-Eingänge betrieben werden (ftiDisOff = 0, ftiDisUltra = 1, ftiDisVol = 2)
DisNr	Nummer des DistanceSensorEinganges (D1 = 1, D2 = 2)
ICCode	Mit Auswertung der Code Taste des IR Senders (Alle = 0, Code1 = 1, Code2 = 2)
Dir	Drehrichtung eines Motors (Links = 1, Rechts = 2, Aus = 0)
InputNr	Nummer des I-Einganges (1 – 8)
IRKey	Nummer der Taste des IR Senders (M3R = 1, M3L = 2, M1 = 3, M2 = 4, M3 = 5, Code2 = 5, M1BW = 7 M1FW = 8, M2L = 9, M2R = 10, Code1 = 11)
LampNr	Nummer des O-Ausganges (1 – 8)
ModeStatus	Aktueller Betriebsmode der M-Ausgänge (00 : Normal, 01 RobMode)
MotorNr	Nummer des M-Ausganges (1 – 4)
MotorStatus	Aktuelle Drehrichtung aller M-Ausgänge (00 : Aus, 01 : Links, 10 : Rechts)
mSek	Zeit in Millisekunden
NrOfChanges	Anzahl der Impulse (false -> true oder umgekehrt)
OnOff	Vergleich mit true oder false
Position	Position in Anzahl von Impulsen ab zugehörenden Endtaster.
SMESSAGE	Struktur mit einer Nachricht
Speed	Geschwindigkeit mit der ein M-Ausgang betrieben werden soll (1 – 8)
SpeedStatus	Aktuelle Geschwindigkeit aller M-Ausgänge (4bit pro Ausgang 0000 – 0111)
TermNr	Nummer eines I-Einganges der bei true alternative eine Funktion beenden soll
VoltNr	Nummer eines Spannungseinganges (A1 = 1, A2 = 2, AV = 3, AZ = 4)

Funktionen

Die Funktionen entsprechen denen von CFishFace. Anstelle vonOpen/CloseInterface tritt hier Init/FinishFish. Da Renesas C wohl keine Überladungen kennt, wurde Finish in Finish und FinishIR und WaitForInput in WaitForInput und WaitForIR geteilt. Ebenso SetMotor in SetMotor und SetRobMotor, konsequent : WaitForRobMotors.

void **InitFish**(BOOL rf = FALSE, UINT DistanceMode = ftiDisVolt);

Starten des Interfaces (VC++ : Erstes ROBO Interface an USB)

void **FinishFish**();

Beenden des Interface-Betriebes

void **ClearCounter**(UCHAR InputNr);

Löschen (0) des zum I-Eingang gehörenden Counters

void **ClearCounters**();

Löschen (0) aller Counter

void **ClearMessagesIn**();

Löschen der Queque mit den eingehenden Nachrichten

void **ClearMessagesOut**();

Löschen der Queque mit den ausgehenden Nachrichten

void **ClearMotors**();

Löschen aller M-Ausgänge

BOOL **Finish**(UCHAR InputNr = 0);

Anforderung zum Abbruch des Programms (ohne Parameter nur VC++)

BOOL **FinishIR**(UCHAR IRCode, UCHAR IRKey);

Anforderung zum Abbruch des Programms über IR Sender

UINT **GetAnalog**(UCHAR AnalogNr);

Auslesen eines Analogwertes

UINT **GetCounter**(UCHAR InputNr);

Auslesen eines zu einem I-Eingang gehörenden Impulszählers

UINT **GetDistanceValue**(UCHAR DisNr);

Auslesen eines zu einem D-Eingang gehörenden Distanzsensorwertes
Achtung InitFish muß mit DistanceMode = ftiDisUltra aufgerufen werden.

BOOL **GetInput**(UCHAR InputNr);

Auslesen eines I-Einganges

UCHAR **GetInputs**();

Auslesen aller I-Eingänge

BOOL **GetIRKey**(UCHAR IRCode, UCHAR IRKey);

Auslesen von Tastendrücken auf dem IR Sender

UCHAR **GetRFMessage**(SMESSAGE* mData);

Lesen der ältesten Nachricht aus der Queque der eingehenden Nachrichten

UINT **GetVoltage**(UCHAR VoltNr);

Auslesen eines Spannungseinganges

BOOL **IsRFMessage**();

Abfrage, ob mindestens eine Nachricht in der Queque der eingehenden Nachrichten vorhanden ist.

UCHAR **Outputs**();

Lasen des Status aller M-Ausgänge

void **Pause**(UINT mSek);

Anhalten des Programmablauf für eine Anzahl von Millisekunden

UCHAR **SendRFMessage**(SMESSAGE* mData, UCHAR Spez = 0);

Senden einer Nachricht

void **SetCounter**(UCHAR InputNr, UINT Value);

Setzen eines zu einem I-Eingang gehörenden Impulszählers

void **SetLamp**(UCHAR LampNr, BOOL OnOff, UCHAR Power = 8);

Schalten eines O-Ausganges

void **SetMotor**(UCHAR MotorNr, UCHAR Dir, UCHAR Speed = 8);

Schalten eines M-Ausganges

void **SetMotors**(UCHAR MotorStatus, UINT SpeedStatus = 0x7777, UINT ModeStatus = 0);

Schalten aller M-Ausgänge

void **SetRobMotor**(UCHAR MotorNr, UCHAR Dir, UCHAR Speed, UINT Counter);

Schalten eines RobMotors (M-Ausgang mit zu geordnetem Impulszähler und Endtaster)

void **WaitForChange**(UCHAR InputNr, UINT NrOfChanges, UCHAR TermNr = 0);

Warten auf das Auftreten einer bestimmten Anzahl von Impulsen (false -> true und umgekehrt) am angegebenen I-Eingang, alternativ I-Eingang TermNr = true

void **WaitForGreater**(UCHAR DisNr, UINT DisValue);

Warten auf einen Wert des angegebenen Distanzsensors, der größer ist als DisValue

void **WaitForHigh**(UCHAR InputNr);

Warten auf einen false / true Durchgang am angegebenen I-Eingang

void **WaitForInput**(UCHAR InputNr, BOOL OnOff = TRUE);

Warten auf einen Zustand am angegebenen I-Eingang

void **WaitForIR**(UCHAR IRCode, UCHAR IRKey, BOOL OnOff = TRUE);

Warten auf einen Zustand am IR Sender

void **WaitForLess**(UCHAR DisNr, UINT DisValue);

Warten auf einen Wert des angegebenen Distanzsensors, der kleiner ist als DisValue

void **WaitForLow**(UCHAR InputNr);

Warten auf einen true/false Durchgang am angegebenen I-Eingang

UCHAR **WaitForRFMessage**(UINT mSek, SMESSAGE* mData);

Warten auf das Eintreffen einer Nachricht. Gewartet wird die angegebene Anzahl von Millisekunden, bei Angabe 0 wird endlos gewartet. Rückgabe : Art der Beendung (1 : Nachricht, 0 = Zeitablauf)

UCHAR **WaitForRobMotors**(UINT mSek, UCHAR M1, UCHAR M2 = 0, UCHAR M3 = 0, UCHAR M4 = 0);

Warten auf die Readymeldung aller angegebenen RobMotoren. Gewartet wird die angegebene Anzahl von Millisekunden, bei Angabe 0 wird endlos gewartet. Rückgabe : Art der Beendung (1 : Ready, 0 = Zeitablauf)

int **WaitForPositionDown**(UCHAR InputNr, int Counter, int Position, UCHAR TermNr = 0);

Warten auf Erreichen der Bestimmungsposition durch Dekrementieren des aktuellen Counters

int **WaitForPositionUp**(UCHAR InputNr, int Counter, int Position, UCHAR TermNr = 0);

Warten auf Erreichen der Bestimmungsposition durch Inkrementieren des aktuellen Counters

void **WaitForTime**(UINT mSek);

Wie Pause : Anhalten des Programmablauf für eine Anzahl von Millisekunden

Anmerkungen zu den Counters

Ein wesentliches Element zur Positionsbestimmung sind die Counters. Sie sind den I-Eingängen zugeordnet In den Countern wird jede Veränderung des Zustandes der I-Eingänge gezählt. Also z.B. das Öffnen oder auch das Schließen eines Tasters.

Es werden entsprechende Funktionen/Methoden zur Handhabung der Counter angeboten. Die Counter werden auch intern von einigen Funktionen/Methoden (z.B. SetMotor mit Parameter Counter und den meisten Wait-Methoden) genutzt

Anmerkungen zu den Rob-Funktionen

Die Rob-Funktionen laufen in einem bsonderen Betriebsmodus, dem RobMode. In diesem Modus werden die betroffenen Counter decrementiert. Bei Erreichen des Wertes 0 wird der betroffen Motor abgeschaltet. Gelegentlich kann es vorkommen, daß noch um einen Impuls weiter gefahren wird. Das kann man durch Abfrage des entsprechenden ImpulsCounters (wert > 0) feststellen und bei der Speicherung der aktuellen Position entsprechend berücksichtigen.

Der Betrieb eines Motors mit den Rob-Funktionen setzt ein festes Anschlußkonzept voraus. Zum jeweiligen Motor gehören je ein Impulstaster und ein Endtaster. Dazu folgende Tabelle :

Motor	**Endtaster**	**Impulstaster**
1	1	2
2	3	4
3	5	6
4	7	8
5	9	10
6	11	12
7	13	14
8	15	16

Und bei genügend vielen Extensions weiter bis Motor 16

Die Motoren sind „linksdrehend“ d.h. sie drehen bei ftiLinks in Richtung Endtaster.

Die Motoren können einzeln über SetRobMotor bzw. eine Variante von SetMotors geschaltet werden. Das Argument ICount/Counter gibt die Fahrstrecke in Impulsen (true/false- oder false/true-Durchgang am zugehörenden Impulstaster) an. Sie werden während des Pollens auf Null heruntergezählt. Sie sind über den ftiDCB.Counter bzw. die Funktion csGetCounter zugänglich. Also Achtung, etwa durch die Anwendung gesetzte Counter können in diesem Fall geändert werden.

Die Motoren können auch alle mit einem Befehl geschaltet werden : rbRobMotors bzw. einer Variante von SetMotor. Dazu müssen vorher die Parameter aufbereitetet werden.
MotorStatus : pro Motor 2bit, mit M1 : bit 0 und 1 beginnend.
00 : aus, 01 links, 10 rechts.
SpeedStatus : pro Motor 4bit, mit M1 : bit 0-3 beginnend,
0000 aus, 1000 halbe Kraft, 0111 voll.
ModeStatus : proMotor 2 bit, mit M1 : bit 0-1 beginnend,
00 Normal-Mode, 01 Rob-Mode, Rest z.Zt. nicht besetzt
(vorgesehen z.B. für Schrittmotorenbetrieb).

Beispiel : SetMotors(ft, 0x9, 0x74, 0x0, 0x05);
0x steht für Hexa, binär : MotorStatus 1001 SpeedStatus 01110100 ModeStatus 0101 -> M2 = rechts, Speed 7 im Rob-Mode, M1 = links, Speed 4 im RobMode. Der Rest steht. Vor SetMotors sind für jeden Motor einzeln die Impuls-Counter auf die gewünschte Fahrstrecke zu setzen.

Direction = 0 bzw. die Angabe im MotorStatus hält den Motor unabhängig von den Speed-Werten an.

Die Motoren laufen simultan, sie können der Reihe nach mit SetRobMotor geschaltet werden. Sie starten dann beim nächsten Zyklus automatisch und laufen asynchron (d.h. unabhängig von den Aktionen des rufenden Programms) bis sie die vorgegebene Position erreicht haben. Sie werden dann ebenfalls (einzeln) während des Zyklus abgeschaltet.

Um Festzustellen, ob die Motoren ihr Ziel erreicht haben und um das Programm mit den durch durch die Rob-Funktionen ausgelösten Aktionen wieder zu synchronisieren ist ein WaitForRobMotor(s) erforderlich.

Anmerkungen zum Funk-Betrieb

Am Funkbetrieb beteiligt sind das ROBO RF Datalink und ROBO Interfaces mit RF-Platine

Es gibt drei unterschiedliche Typen des Funkbetriebes :

1. **Route Through** : Ein Interface mit RF-Platine ist über das RF Datalink an den PC angeschlossen. Das Anwendungsprogramm läuft im PC ohne die Anschlußart kennen zu müssen. Vorteil : Ein mit dem Interface + RF-Platine ausgerüstetes Modell kann sich frei im Raum bewegen, Kontrolle und Bedienoberfläche liegen auf dem PC
   Wird durch umFish40.DLL voll unterstützt.
2. **Autonom** : Mehrere Interfaces mit RF-Platine kommunizieren miteinander über Funk
   Das RF Datalink übernimmt die Rolle des **Messages Routers**. Die Anwendungsprogramme laufen in den Interfaces.
   Wird durch umFish40.DLL nicht unterstützt.
3. **Route Through** und **Message Router**. Das erste Interface wird durch eine PC-Anwendung unterstützt, die weiteren laufen autonom. Sie können aber von der Anwendung des ersten aus dem PC über Funk erreicht werden.
   Dieser Funkverkehr wird durch umFish40.DLL von der PC-Seite aus unterstützt.
   Die Programmierung der Interfaces erfolgt in ROBO Pro oder Renesas C (ab dem zweiten Interface).

   Dazu gibt es einige speziellen Funktionen :
   - SendRFMessage zum Senden einer gepufferten (Broadcast) Nachricht.
   – GetRFMessage und IsRFMessage zum entgegennehmen einkommender Nachrichten aus dem Empfangspuffer. Sowie ClearMessagesIn, ClearMessagesOut zum Löschen der Messages-Queques.
   – Die Struktur SMESSAGE zur Darstellung der Funk-Daten. Bis auf HwId das die Versandart enthält (RF Broadcast über Funk, Code 2). Können die Felder der Struktur in Abstimmung der am Funkverkehrbeteiligten frei vereinbahrt werden.

Anhang

Anlegen eines Renesas Projekts für den Betrieb

1. New Project Workspace | Application
   M16C, Renesas M16C Standard
2. Target CPU : 5.40.00 | M16C/20-24
3. Select RTOS : M16C/20 | none | user
4. Finish : Verzeichnisse werden angelegt
5. Manuell kopieren : Verzeichnisse StartUpCode und TA_Firmware
   sowie die Files roboFish.H und roboFish.C
6. Project | Add Files ...
   Aus Verzeichnis StartUpCode : ncrt0.a30
   Aus TA_Firmware : TAF_00.C
   Und zusätzlich aus \Message (auch wenn nicht gefunkt wird)
   Msg_10.c und Msg_50.c
   Es werden jeweils automatisch weitere Files kopiert
   und dann noch : roboFish.H/C
7. Build | Renesas .. ToolChain .. | Lmc | Format ... HEX
8. #include "roboFish.h" im Hauptprogramm (Name wie Projekt)
   in roboFish sind weitere includes "TA_Firmware\TAF_00D.h" und ...TAF_00P.h"

Zusätzlich Hinweise

Achtung : Das File ProgNr.INC des Projektes RunRenesaas enthält die Festlegung in welchen Speicherbereich das Programm geladen werden soll. Standard ist RAM1. Bei Bedarf ist das auf FLASH1 bzw. FLASH2 zu ändern.

Laden in das ROBO Interface mit dem Tool FtLoader aus dem Paket FtCComp UND Starten.

Anlegen eines VC++ 6.0 Projektes für die Simulation

1. Neu | Projekt | Win32-Konsolen-Anwendung : Projektname und leeres Projekt
2. Projekt speichern und beenden.
3. Die Verzeichnisse FtLib und RoboFish von roboFish.ZIP manuell in den Projektpfad einfügen.
4. Projekt wieder starten und Projekt | Dem Projekt hinzufügen | Dateien :
   roboFish.H, roboFish.CPP, FtLib.H, FtLib_Static_LIBCMTD_Debug.LIB
   zusätzlich eine Datei mit einem UCHAR main void ... z.B. Stanze von den Beispielen
5. Projekt | Einstellungen | Win32 Debug | C/C++ Kategorie | Code Generation :
   Multithreaded debuggen.