Control III Programmierung in C (Klein SPS)

Beschreibung der Befehle

Ausgabe: 2013-02-21

Änderungen vorbehalten.

Die Nennung von Waren erfolgt in diesem Werk in der Regel ohne Erwähnung bestehender Patente, Gebrauchsmuster oder Warenzeichen.

Das Fehlen eines solchen Hinweises begründet nicht die Annahme, eine Ware sei frei.

Inhaltsverzeichnis

Control III Programmierung in C (Klein SPS)

Ausgabedatum: 21.02.13

Ausgabedatum: 21.02.13

Ausgabedatum: 21.02.13

Abbildungsverzeichnis

Ausgabedatum: 21.02.13

1. Die verwendeten Symbole

Hinweis!

Dieses Zeichen macht auf eine wichtige Information aufmerksam.

Ausgabedatum: 21.02.2013

2. Allgemein

2.1 Produktinformation

Control III erfüllt alle Aufgaben einer leistungsfähigen Klein-SPS und bietet Ihnen die Möglichkeit alle Sensor- und Aktorinformationen vorzuverarbeiten.

2.2 Einführung

Control III, die in die AS-i Master integrierte SPS-Funktionalität, bildet zusammen mit handelsüblichen AS-i E/A-Bausteinen eine Klein-SPS mit bis zu 248 Ein- und Ausgängen pro AS-i Kreis, im Doppelmaster insgesamt bis zu 496 E/A's. Die Programmierung der Klein-SPS erfolgt komplett in Standard C.

Die Verwendung von **Control III** in den Gateways bietet die Möglichkeit der Vorverarbeitung der Sensor-Aktor-Informationen. Typische Anwendungsfälle sind die Vorverarbeitung von Daten sowie die schnelle Ausführung von zeitkritischen Operationen direkt im Gateway.

Das Programm für **Control III** wird auf einem handelsüblichen PC erstellt und anschließend in den AS-i Master geladen. Als Programmierwerkzeug steht eine eigens dafür erstellte Eclipse-Version (IDE) zur Verfügung. Diese beinhaltet alle nötigen Tools, um **Control III**-Programm zu erstellen und zu testen.

Control III wird in der weitverbreiteten und sehr leistungsfähigen Programmiersprache „C“ geschrieben. Als Unterstützung dienen Ihnen verschiedene Bibliotheksfunktionen, welche das Programmieren erleichtern soll.

Das **Control III** Programm wird über eine der Schnittstellen am Gateway geladen und nichtflüchtig in einem Flash gespeichert.

Ausgabedatum: 21.02.2013

3 Sicherheit

3.1 Bestimmungsgemäße Verwendung

Warnung!

Der Schutz von Betriebspersonal und Anlage ist nicht gewährleistet, wenn die Baugruppe nicht entsprechend ihrer bestimmungsgemäßen Verwendung eingesetzt wird.

Das Gerät darf nur von eingewiesenem Fachpersonal entsprechend der vorliegenden Betriebsanleitung betrieben werden.

3.2 Allgemeine Sicherheitshinweise

Warnung!

Ein anderer Betrieb, als der in dieser Anleitung beschriebene, stellt die Sicherheit und Funktion des Gerätes und angeschlossener Systeme in Frage.

Der Anschluss des Gerätes und Wartungsarbeiten unter Spannung dürfen nur durch eine elektrotechnische Fachkraft erfolgen.

Können Störungen nicht beseitigt werden, ist das Gerät außer Betrieb zu setzen und gegen versehentliche Inbetriebnahme zu schützen.

Reparaturen dürfen nur direkt beim Hersteller durchgeführt werden. Eingriffe und Veränderungen im Gerät sind nicht zulässig und machen jeden Anspruch auf Garantie nichtig.

Hinweis!

Die Verantwortung für das Einhalten der örtlich geltenden Sicherheitsbestimmungen liegt beim Betreiber.

3.3 Entsorgung

Hinweis!

Verwendete Geräte und Bauelemente sachgerecht handhaben und entsorgen!

Unbrauchbar gewordene Geräte als Sondermüll entsorgen!

Die nationalen und örtlichen Richtlinien bei der Entsorgung einhalten!

Ausgabedatum: 21.02.2013

4 Programmieren von Control III

Das Programmieren von **Control III** basiert auf der weitverbreiteten und höchst modularen Programmiersprache 'C'. Um ein Control III Programm zu erstellen, müssen alle Aufgaben, welche das Gateway übernehmen soll, in 'C' geschrieben und in das Gateway geladen werden.

Um mit **Control III** zu programmieren, beinhaltet die Header-Datei 'control.h' eine Reihe von Bibliotheksfunktionen, welche verwendet werden können, um bestimmte Master-Funktionen auszuführen. Im Folgenden werden diese Funktionen aufgelistet und erläutert.

4.1 Übersicht

AASiDataExchange	AS-i Data Exchange dient zum Austausch von AS-i Daten und liest die 'execution control' flags.
AASiReadIDI	Liest die Eingangsdaten von Slaves und die 'execution control' flags.
AASiWriteODI	Schreibt die Ausgangsdaten von Slaves.
AASiReadODI	Liest die Ausgangsdaten aus dem AS-i Master.
AASiWritePP	Schreibt die ständigen Parameter eines Slaves im Master.
AASiReadPP	Liest die ständigen Parameter eines Slaves im Master.
AASiSendParameter	Sendet die Parameter zu einem Slave.
AASiReadPI	Liest die aktuellen Parameter eines Slaves.
AASiStorePI	Speichert die aktuellen Slave-Parameter als ständige Parameter.
AASiReadDuplicateAdrList	Liest die Liste aller Doppeladressen.
AASiReadFaultDetector	Liest Überspannung, Nois, EFLT und Doppeladressen.
AASiWritePCD	Schreibt die projektierte Konfiguration eines Slaves.
AASiReadPCD	Liest die projektierte Konfiguration eines Slaves.
AASiStoreCDI	Speichert die aktuelle Konfiguration als ständige Konfiguration.
AASiReadCDI	Liest die aktuelle Konfiguration eines Slaves.
AASiWriteExtID1	Schreibt den extended ID-Code 1 von Slave 0.
AASiWriteLPS	Schreibt die projektierten Slaves.
AASiReadLPS	Liest die projektierten Slaves.
AASiReadLAS	Liest die aktivierten Slaves
AASiReadLDS	Liest die erkannten Slaves.
AASiReadLCS	Liest alle fehlerhaften Slaves.

Tab. 4-1.

Ausgabedatum: 21.02.2013

AASiWriteLOS	Liste der Slaves welche bei einem Konfigurations-fehler offline gehen sollen.
AASiReadLOS	Liest die Liste der Slaves, welche bei einem Konfi-gurationsfehler offline gehen.
AASiReadLPF	Liest die Liste der Slaves mit Peripheriefehler.
AASiReadEcFlags	Liest die 'execution control' flags
AASiSetConfigMode	Setzt den AS-i Master in den Konfigurationsmodus oder in den geschützten Betriebsmodus.
AASiWriteHiFlags	Schreibt die Host-Interface-Flags des AS-i Masters.
AASiReadHiFlags	Liest die Host-Interface-Flags des AS-i Masters.
AASiAddressSlave	Ändert die Adresse eines Slaves.
AASiExecuteCommand	Direkt zu sendendes AS-Interface Kommando
AASiReadAllConfig	Liest alle Konfigurationsdaten (z.B. LPS, PP[ ] und PCD [ ]) aller angeschlossenen Slaves.
AASiWriteAllConfig	Schreibt alle Konfigurationsdaten (z.B. LPS, PP[ ] und PCD [ ]) aller angeschlossenen Slaves.
AASiReadErrorCounters	Liest den Slave-Error-Counter
AASiMailbox	Generic Mailbox-Funktion.
AASiWrite16BitODI	Schreibt vier Kanäle 16bit ODI an einen AS-i-Slave mit z.B. Analog-Slave profil 7.3 oder 7.4.
AASiRead16BitODI	Liest vier 16bit ODI Kanäle eines AS-i Slaves mit z.B. Analog-Slave profil 7.3 oder 7.4.
AASiRead16BitIDI	Liest vier 16bit IDI Kanäle eines AS-i Slaves mit z.B. Analog-Slave profil 7.3 oder 7.4.
AASiReadCtrlAccODI	Liest die Control III Berechtigung, um Ausgangsda-ten zu verändern.
AASiWriteCtrlAccODI	Schreibt die Control III Berechtigungen der Slaves, um Ausgangsdaten zu verändern.
AASiReadCtrlAccAODI	Liest die Control III Berechtigung, um analoge Aus-gangsdaten zu verändern.
AASiWriteCtrlAccAODI	Schreibt die Control III Berechtigungen der Slaves, um analoge Ausgangsdaten zu verändern.
CCtrlInitTimer	Initialisierung einer Timerfunktion
CCtrlDelay	Verzögerung in ms
CCtrlInitWdg	Initialisierung des Control III Watchdogs
CCtrlTriggerWdg	Trigger Control III Watchdog.
CCtrlEvalCycletime	Bewertung der Control III Zykluszeit
CCtrlReadParameter	Liest Control NV-Parameter.
CCtrlWriteParameter	Schreibt Control NV-Parameter.
CCtrlReadFlags	Liest Control-Flags.
CCtrlWriteFlags	Schreibt Control-Flags.

Tab. 4-1.

Ausgabedatum: 21.02.2013

CCtrlReadkey	Liest benutzerdefinierten eindeutigen Control-Schlüssel.
CCtrlPrintf	Printf-Funktion
CCtrlBreakpoint	Initialisierung des Debuggers
CCtrlDisplay	Control III Displayfunktion

Tab. 4-1.

4.1.1 ASi Data Exchange

AS-i Data Exchange dient zum Austausch von AS-i Daten zwischen dem Master und der Applikation und liest die 'execution control' flags.

```
int (*AASiDataExchange) (unsigned char Circuit, AASiProcessData
ODI, AASiProcessData IDI, AASiEcFlags *EcFlags);
```

Parameter:

`Circuit:` AS-i Master Kreis
`ODI:` 32 Byte Ausgangsdaten

Return:

`IDI:` 32 Byte Eingangsdaten
`EcFlags:` Execution control flags (2 Bytes) des AS-i Master.

AS-i Kreis 1 / 2 : Eingangsdatenabbild IDI

Bit	7	6	5	4	3	2	1	0	Bit	7	6	5	4	3	2	1	0
Byte	D3	D2	D1	D0	D3	D2	D1	D0	Byte	D3	D2	D1	D0	D3	D2	D1	D0
0	Slave 1/1A				Slave 0/0A				1	Slave 3/3A				Slave 2/2A			
2	Slave 5/5A				Slave 4/4A				3	Slave 7/7A				Slave 6/6A			
4	Slave 9/9A				Slave 8/8A				5	Slave 11/11A				Slave 10/10A			
6	Slave 13/13A				Slave 12/12A				7	Slave 15/15A				Slave 14/14A			
8	Slave 17/17A				Slave 16/16A				9	Slave 19/19A				Slave 18/18A			
10	Slave 21/21A				Slave 20/20A				11	Slave 23/23A				Slave 22/22A			
12	Slave 25/25A				Slave 24/24A				13	Slave 27/27A				Slave 26/26A			
14	Slave 29/29A				Slave 28/28A				15	Slave 31/31A				Slave 30/30A			
16	Slave 1/1B				Slave 0/0B				17	Slave 3/3B				Slave 2/2B			
18	Slave 5/5B				Slave 4/4B				19	Slave 7/7B				Slave 6/6B			
20	Slave 9/9B				Slave 8/8B				21	Slave 11/11B				Slave 10/10B			
22	Slave 13/13B				Slave 12/12B				23	Slave 15/15B				Slave 14/14B			
24	Slave 17/17B				Slave 16/16B				25	Slave 19/19B				Slave 18/18B			
26	Slave 21/21B				Slave 20/20B				27	Slave 23/23B				Slave 22/22B			
28	Slave 25/25B				Slave 24/24B				29	Slave 27/27B				Slave 26/26B			
30	Slave 29/29B				Slave 28/28B				31	Slave 31/31B				Slave 30/30B			

Tab. 4-2. Eingangsdatenabbild IDI

Ausgabedatum: 21.02.2013

AS-i Kreis 1 / 2 : Ausgangsdatenabbild ODI

Bit	7	6	5	4	3	2	1	0	Bit	7	6	5	4	3	2	1	0
Byte	D3	D2	D1	D0	D3	D2	D1	D0	Byte	D3	D2	D1	D0	D3	D2	D1	D0
0	Slave 1/1A				Slave 0/0A				1	Slave 3/3A				Slave 2/2A			
2	Slave 5/5A				Slave 4/4A				3	Slave 7/7A				Slave 6/6A			
4	Slave 9/9A				Slave 8/8A				5	Slave 11/11A				Slave 10/10A			
6	Slave 13/13A				Slave 12/12A				7	Slave 15/15A				Slave 14/14A			
8	Slave 17/17A				Slave 16/16A				9	Slave 19/19A				Slave 18/18A			
10	Slave 21/21A				Slave 20/20A				11	Slave 23/23A				Slave 22/22A			
12	Slave 25/25A				Slave 24/24A				13	Slave 27/27A				Slave 26/26A			
14	Slave 29/29A				Slave 28/28A				15	Slave 31/31A				Slave 30/30A			
16	Slave 1/1B				Slave 0/0B				17	Slave 3/3B				Slave 2/2B			
18	Slave 5/5B				Slave 4/4B				19	Slave 7/7B				Slave 6/6B			
20	Slave 9/9B				Slave 8/8B				21	Slave 11/11B				Slave 10/10B			
22	Slave 13/13B				Slave 12/12B				23	Slave 15/15B				Slave 14/14B			
24	Slave 17/17B				Slave 16/16B				25	Slave 19/19B				Slave 18/18B			
26	Slave 21/21B				Slave 20/20B				27	Slave 23/23B				Slave 22/22B			
28	Slave 25/25B				Slave 24/24B				29	Slave 27/27B				Slave 26/26B			
30	Slave 29/29B				Slave 28/28B				31	Slave 31/31B				Slave 30/30B			

Tab. 4-3. Ausgangsdatenabbild ODI

Ausgabedatum: 21.02.2013

4.1.2 AS-i Read IDI

AS-i Read IDI liest die Eingangsdaten von Slaves und die execution control flags.

```
int (*AASiReadIDI) (unsigned char Circuit, AASiProcessData IDI,
AASiSlaveAddr First, unsigned char Amount, AASiEcFlags
*EcFlags);
```

Parameter:

`Circuit:`	AS-i Master Kreis
`First:`	Index des ersten Slaves
`Amount:`	Anzahl der zu lesenden Slaves

Return:

`IDI:`	32 Byte Eingangsdaten Jeder Slave verwendet 4 Bit (Nibble) eines Bytes. Unbenutzte Bytes werden auf Null gesetzt
`EcFlags:`	Execution control flags (2 Bytes) des AS-i Master

4.1.3 AS-i Write ODI

AS-i Write ODI schreibt die Ausgangsdaten von Slaves.

```
int (*AASiWriteODI) (unsigned char Circuit, AASiProcessData ODI,
AASiSlaveAddr First, unsigned char Amount);
```

Parameter:

`Circuit:`	AS-i Master Kreis
`ODI:`	32 Byte Ausgangsdaten
`First:`	Index des ersten Slaves
`Amount:`	Anzahl der zu schreibenden Slaves

Jeder Slave verwendet 4 Bit (Nibble) eines Bytes. Unbenutzte Bytes werden auf Null gesetzt.

Return: —

4.1.4 AS-i Read ODI

AS-i Read ODI liest die Ausgangsdaten aus dem AS-i Master.

```
int (*AASiReadODI) (unsigned char Circuit, AASiProcessData ODI);
```

Parameter:

`Circuit`	AS-i Master Kreis

Return:

`ODI:`	32 Byte Ausgangsdaten Jeder Slave verwendet 4 Bit (Nibble) eines Bytes. Unbenutzte Bytes werden auf Null gesetzt.

Ausgabedatum: 21.02.2013

4.1.5 AS-i Write Permanent Parameter

AS-i Write Permanent Parameter schreibt die ständigen Parameter eines Slaves im Master.

```
int (*AASiWritePP) (unsigned char Circuit, AASiSlaveAddr Address, AASiSlaveData PP);
```

Parameter:

`Circuit:` AS-i Master Kreis
`Address:` Adresse des Slaves
`PP:` permanente Parameter (low Nibble)

Return: —

4.1.6 AS-i Read Permanent Parameter

AS-i Read Permanent Parameter liest die ständigen Parameter eines Slaves im Master.

```
int (*AASiReadPP) (unsigned char Circuit, AASiSlaveAddr Address, AASiSlaveData *PP);
```

Parameter

`Circuit:` AS-i Master Kreis
`Address:` Adresse des Slaves

Return:

`PP:` permanente Parameter (low nibble)

4.1.7 AS-i Send Parameter

AS-i Send Parameter sendet die Parameter zu einem Slave.

```
int (*AASiSendParameter) (unsigned char Circuit, AASiSlaveAddr Address, AASiSlaveData PI, AASiSlaveData *Return);
```

Parameter:

`Circuit:` AS-i Master Kreis
`Address:` Adresse des Slaves
`PI:` zu sendende Parameter (low Nibble)

Return:

`Return:` Bei einem Fehler wird PI invertiert zurückgegeben.

Ausgabedatum: 21.02.2013

4.1.8 AS-i Read PI

AS-i Read PI liest die aktuellen Parameter eines Slaves.

```
int (*AASiReadPI) (unsigned char Circuit, AASiSlaveAddr Address,
AASiSlaveData *PI);
```

Parameter:

`Circuit:` AS-i Master Kreis
`Address:` Adresse des Slaves

Return:

`PI:` zu sendende Parameter (low Nibble)

4.1.9 AS-i Store PI

AS-i Store PI speichert die aktuelle Slave-Parameter als ständige Parameter.

```
int (*AASiStorePI) (unsigned char Circuit);
```

Parameter:

`Circuit:` AS-i Master Kreis
`Address:` Adresse des Slaves

Return: —

4.1.10 AS-i Read Duplicate Address List

AS-i Read Duplicate Address List liest die Liste aller Doppeladressen.

```
int (*AASiReadDuplicateAdrList) (unsigned char Circuit,
AASiSlaveList, *DpAdrList);
```

Parameter:

`Circuit:` AS-i Master Kreis

Return:

`DpAdrList:` Liste der doppelten Adressen

Ausgabedatum: 21.02.2013

4.1.11 AS-i Read Fault Detector

AS-i Read Fault Detector liest Überspannung, Nois, EFLT und Doppeladressen.

```
int (*AASiReadFaultDetector) (unsigned char Circuit,
unsigned char *pucFaultDetectorActiv,
unsigned char *pucFaultDetectorHistoric);
```

Parameter:

`Circuit:` AS-i Master Kreis

Return:

`PucFaultDetectorActiv:` Aktiver Fehlererfasser
`PucFaultDetectorHistoric:` Historischer Fehlererfasser

4.1.12 AS-i Write PCD

AS-i Write PCD schreibt die projektierte Konfiguration eines Slaves.

```
int (*AASiWritePCD) (unsigned char Circuit, AASiSlaveAddr
Address, AASiConfigData PCD);
```

Parameter:

`Circuit:` AS-i Master Kreis
`Address:` Adresse des Slaves

Return:

`PCD:` zu schreibende projektierte Konfiguration des Slaves

Bit	7	6	5	4	3	2	1	0
Byte	**D3**	**D2**	**D1**	**D0**	**D3**	**D2**	**D1**	**D0**
0	Slave ID				I/O Configuration			
1	Extended ID2				Extended ID1			

Tab. 4-4. PCD Konfiguration

4.1.13 AS-i Read PCD

AS-i Read PCD liest die projektierte Konfiguration eines Slaves.

```
int (*AASiReadPCD) (unsigned char Circuit, AASiSlaveAddr
Address, AASiConfigData *PCD);
```

Parameter:

`Circuit:` AS-i Master Kreis
`Address:` Adresse des Slaves

Return:

`PCD:` projektierte Konfiguration des Slaves (siehe Tab. <PCD Konfiguration>).

Ausgabedatum: 21.02.2013

4.1.14 AS-i Store CDI

AS-i Store CDI speichert die aktuelle Konfiguration als ständige Konfiguration.

```
int (*AASiStoreCDI) (unsigned char Circuit);
```

Parameter:

`Circuit:` AS-i Master Kreis

4.1.15 AS-i Read CDI

AS-i Read CDI liest die aktuelle Konfiguration eines Slaves.

```
int (*AASiReadCDI) (unsigned char Circuit, AASiSlaveAddr
Address, AASiConfigData *CDI);
```

Parameter:

`Circuit:` AS-i Master Kreis
`Address:` Adresse des Slaves

Return:

`CDI:` Konfiguration des Slaves

Bit	7	6	5	4	3	2	1	0
Byte	**D3**	**D2**	**D1**	**D0**	**D3**	**D2**	**D1**	**D0**
0	Slave ID				I/O Configuration			
1	Extended ID2				Extended ID1			

Tab. 4-5. CDI Konfiguration

4.1.16 AS-i Write Extended ID1

AS-i Write Extended ID1 schreibt den extended ID-Code 1 von Slave 0.

```
int (*AASiWriteExtID1) (unsigned char Circuit, AASiSlaveData
ID1);
```

Parameter:

`Circuit:` AS-i Master Kreis
`Address:` Adresse des Slaves
`ID1:` Extended ID-Code 1

Return:

Spezielle Fehlercodes, welche den Grund der fehlerhaften Übertragung beschreiben.

Ausgabedatum: 21.02.2013

4.1.17 AS-i Write LPS

AS-i Write LPS schreibt die projektierten Salves.

```
int (*AASiWriteLPS) (unsigned char Circuit, AASiSlaveList LPS);
```

Parameter:

`Circuit:` AS-i Master Kreis
`LPS:` 8 Byte Slave-Liste

Jedes Bit in der LPS entspricht einem Slave wie folgt:

Bit	Slave	Bit	Slave	Bit	Slave	Bit	Slave
0	Slave 0/0A kann nicht eingestellt werden!	16	Slave 16/16A	32	Slave 0/0B kann nicht eingestellt werden!	48	Slave 16/16B
1	Slave 1/1A	17	Slave 17/17A	33	Slave 1/1B	49	Slave 17/17B
2	Slave 2/2A	18	Slave 18/18A	34	Slave 2/2B	50	Slave 18/18B
3	Slave 3/3A	19	Slave 19/19A	35	Slave 3/3B	51	Slave 19/19B
4	Slave 4/4A	20	Slave 20/20A	36	Slave 4/4B	52	Slave 20/20B
5	Slave 5/5A	21	Slave 21/21A	37	Slave 5/5B	53	Slave 21/21B
6	Slave 6/6A	22	Slave 22/22A	38	Slave 6/6B	54	Slave 22/22B
7	Slave 7/7A	23	Slave 23/23A	39	Slave 7/7B	55	Slave 23/23B
8	Slave 8/8A	24	Slave 24/24A	40	Slave 8/8B	56	Slave 24/24B
9	Slave 9/9A	25	Slave 25/25A	41	Slave 9/9B	57	Slave 25/25B
10	Slave 10/10A	26	Slave 26/26A	42	Slave 10/10B	58	Slave 26/26B
11	Slave 11/11A	27	Slave 27/27A	43	Slave 11/11B	59	Slave 27/27B
12	Slave 12/12A	28	Slave 28/28A	44	Slave 12/12B	60	Slave 28/28B
13	Slave 13/13A	29	Slave 29/29A	45	Slave 13/13B	61	Slave 29/29B
14	Slave 14/14A	30	Slave 30/30A	46	Slave 14/14B	62	Slave 30/30B
15	Slave 15/15A	31	Slave 31/31A	47	Slave 15/15B	63	Slave 31/31B

Tab. 4-6. LPS

Der Slave ist projektiert, wenn das Bit gesetzt ist.

Ausgabedatum: 21.02.2013

4.1.18 AS-i Read LPS

AS-i Read LPS liest die projektierten Slaves.

```
int (*AASiReadLPS) (unsigned char Circuit, AASiSlaveList *LPS);
```

Parameter:

`Circuit:` AS-i Master Kreis

Return:

`LPS:` 8 Byte Slave-Liste (siehe Tab. <LPS>)

Der Slave ist projektiert, wenn das Bit gesetzt ist.

4.1.19 AS-i Read LAS

AS-i Read LAS liest die aktivierten Slaves.

```
int (*AASiReadLAS) (unsigned char Circuit, AASiSlaveList *LAS);
```

Parameter:

`Circuit:` AS-i Master Kreis

`LAS:` 8 Byte Slave-Liste

Jedes Bit in der LAS entspricht einem Slave.

Bit	Slave	Bit	Slave	Bit	Slave	Bit	Slave
0	Slave 0/0A	16	Slave 16/16A	32	Slave 0/0B	48	Slave 16/16B
1	Slave 1/1A	17	Slave 17/17A	33	Slave 1/1B	49	Slave 17/17B
2	Slave 2/2A	18	Slave 18/18A	34	Slave 2/2B	50	Slave 18/18B
3	Slave 3/3A	19	Slave 19/19A	35	Slave 3/3B	51	Slave 19/19B
4	Slave 4/4A	20	Slave 20/20A	36	Slave 4/4B	52	Slave 20/20B
5	Slave 5/5A	21	Slave 21/21A	37	Slave 5/5B	53	Slave 21/21B
6	Slave 6/6A	22	Slave 22/22A	38	Slave 6/6B	54	Slave 22/22B
7	Slave 7/7A	23	Slave 23/23A	39	Slave 7/7B	55	Slave 23/23B
8	Slave 8/8A	24	Slave 24/24A	40	Slave 8/8B	56	Slave 24/24B
9	Slave 9/9A	25	Slave 25/25A	41	Slave 9/9B	57	Slave 25/25B
10	Slave 10/10A	26	Slave 26/26A	42	Slave 10/10B	58	Slave 26/26B
11	Slave 11/11A	27	Slave 27/27A	43	Slave 11/11B	59	Slave 27/27B
12	Slave 12/12A	28	Slave 28/28A	44	Slave 12/12B	60	Slave 28/28B
13	Slave 13/13A	29	Slave 29/29A	45	Slave 13/13B	61	Slave 29/29B
14	Slave 14/14A	30	Slave 30/30A	46	Slave 14/14B	62	Slave 30/30B
15	Slave 15/15A	31	Slave 31/31A	47	Slave 15/15B	63	Slave 31/31B

Tab. 4-7. LAS, LDS, LOS und LPF

Der Slave ist aktiviert, wenn das Bit gesetzt ist.

Ausgabedatum: 21.02.2013

4.1.20 AS-i Read LDS

AS-i Read LDS liest die erkannten Slaves.

```
int (*AASiReadLDS) (unsigned char Circuit, AASiSlaveList *LDS);
```

Parameter:

`Circuit:` AS-i Master Kreis

Return:

`LDS:` 8 Byte Slave-Liste

Jedes Bit der LDS entspricht einem Slave (siehe Tab. <LAS, LDS, LOS und LPF>).

4.1.21 AS-i Read LCS

AS-i Read LCS liest alle fehlerhaften Slaves.

```
int (*AASiReadLCS) (unsigned char Circuit, AASiSlaveList *LCS);
```

Parameter:

`Circuit:` AS-i Master Kreis

Return:

`LCS:` 8 Byte Slave-Liste

Jedes Bit der LCS entspricht einem Slave (siehe Tab. <LAS, LDS, LOS und LPF>).

Hinweis!

Die Liste wird nach dem Lesen zurückgesetzt!

Ausgabedatum: 21.02.2013

4.1.22 AS-i Write LOS

AS-i Write LOS schreibt die Liste der Slaves, welche bei einem Konfigurationsfehler offline gehen sollen.

```
int (*AASiWriteLOS) (unsigned char Circuit, AASiSlaveList LOS);
```

Parameter:

`Circuit:` AS-i Master Kreis

Return:

`LOS:` 8 Byte Slave-Adressen-Liste

Jedes Bit der LOS entspricht einem Slave (siehe Tab. <LAS, LDS, LOS und LPF>).

4.1.23 AS-i Read LOS

AS-i Write LOS liest die Liste der Slaves, welche bei einem Konfigurationsfehler offline gehen.

```
int (*AASiReadLOS) (unsigned char Circuit, AASiSlaveList *LOS);
```

Parameter:

`Circuit:` AS-i Master Kreis

Return:

`LOS:` 8 Byte Slave-Adressen-Liste

Jedes Bit der LOS entspricht einem Slave (siehe Tab. <LAS, LDS, LOS und LPF>).

4.1.24 AS-i Read LPF

AS-i Read LPF liest die Liste der Slaves mit Peripheriefehler.

```
int (*AASiReadLPF) (unsigned char Circuit, AASiSlaveList *LPF);
```

Parameter:

`Circuit:` AS-i Master Kreis

Return:

`LPF:` 8 Byte Slave-Adressen-Liste

Jedes Bit der LPS entspricht einem Slave (siehe Tab. <LAS, LDS, LOS und LPF>).

Ausgabedatum: 21.02.2013

4.1.25 AS-i Read Ec Flags

AS-i Read Ec Flags liest die execution control flags.

```
int (*AASiReadEcFlags) (unsigned char Circuit, AASiEcFlags *EcFlags);
```

Parameter:

`Circuit:` AS-i Master Kreis

Return:

`EcFlags:` Zwei Byte EcFlags.

4.1.26 AS-i set Config Mode

AS-i set Config Mode setzt den AS-i Master in den Konfigurationsmodus oder in den geschützten Betriebsmodus.

```
int (*AASiSetConfigMode) (unsigned char Circuit, unsigned char Mode);
```

Parameter:

`Circuit:` AS-i Master Kreis
`Mode:` 1 = Konfigurationsmodus
0 = Geschützter Betriebsmodus

Return: —

4.1.27 AS-i Write Hi Flags

AS-i Write Hi Flags schreibt Host-Interface-Flags des AS-i Masters.

```
int (*AASiWriteHiFlags) (unsigned char Circuit, AASiHiFlags HiFlags);
```

Parameter:

`Circuit:` AS-i Master Kreis
`HiFlags:` Host-Interface-Flags (ein Byte)

Return: —

4.1.27.1 AS-i Read Hi-Flags

AS-i Read Hi-Flags liest die Host-Interface-Flags des AS-i Masters.

```
int (*AASiReadHiFlags) (unsigned char Circuit, AASiHiFlags *HiFlags);
```

Parameter:

`Circuit:` AS-i Master Kreis

Return:

`HiFlags:` Host-Interface-Flags (ein Byte)

Ausgabedatum: 21.02.2013

4.1.28 AS-i Address Slave

AS-i Address Slave ändert die Adresse eines Slaves.

```
int (*AASiAddressSlave) (unsigned char Circuit, AASiSlaveAddr
OldAddress, AASiSlaveAddr NewAddress);
```

Parameter:

Circuit:	AS-i Master Kreis
OldAddress:	Alte Slave-Adresse
NewAddress:	Neue Slave-Adresse

Return:

Spezielle Fehlercodes, welche den Grund der fehlerhaften Übertragung beschreiben.

4.1.29 AS-i Execute Command

AS-i Execute Command ist ein direkt zu sendendes AS-i Kommando.

```
int (*AASiExecuteCommand) (unsigned char Circuit, AASiSlaveAddr
Address,  AASiSlaveData Request, AASiSlaveData *Response);
```

Parameter:

Circuit:	AS-i Master Kreis
Address:	Adresse des Slaves
Request:	AS-i Request Befehl

Return:

Response:	AS-i Request

4.1.30 AS-i Read All Config

AS-i Read All Config liest alle Konfigurationsdaten (z.B. LPS, PP[ ] und PCD [ ]) aller angeschlossenen Slaves.

```
int (*AASiReadAllConfig) (unsigned char Circuit, AASiConfig
*Configurations );
```

Parameter:

Circuit:	AS-i Master Kreis
Configurations:	Array zur Aufnahme der Konfigurationsdaten

Return:

Die Konfigurationsdaten in Configurations [ ]

Ausgabedatum: 21.02.2013

4.1.31 AS-i Write All Config

AS-i Write All Config schreibt alle Konfigurationsdaten (z.B. LPS, PP[ ] und PCD [ ]) aller angeschlossenen Slaves.

```
int (*AASiWriteAllConfig) (unsigned char Circuit, AASiConfig
Configurations);
```

Parameter:

`Circuit:` AS-i Master Kreis
`Configurations:` Array zur Aufnahme der Konfigurationsdaten

Return: —

4.1.32 AS-i read Error Counter

AS-i read Error Counter liest den Slave-Error-Counter.

```
int (*AASiReadErrorCounters) (unsigned char Circuit, AASiError-
Counters     Counters);
```

Parameter:

`Circuit:` AS-i Master Kreis

Return:

`Counters:` 64 Byte (ein Byte pro Slave)

Hinweis!

Die Liste wird nach dem Lesen zurückgesetzt!

4.1.33 AS-i Mail Box

Generic Mailbox-Funktion

```
int (*AASiMailbox) (unsigned char Circuit, AASiMbRequestType
Request, int  ExpResLen, AASiMbResponseType *Response);
```

Parameter:

`Circuit:` AS-i Master Kreis
`Request:` Struktur für Mailbox Anfrage
`ExpResLen:` Erwartete Länge der Antwort (-1 = unbekannt)

Return:

`Response:` Struktur für Mailbox Antwort

Ausgabedatum: 21.02.2013

4.1.34 AS-i Write 16Bit ODI

AS-i Write 16Bit ODI schreibt vier 16bit ODI Kanäle eines AS-Interface-Slave mit z.B. Analog-Slave-Profil 7.3 oder 7.4.

```
int (*AASiWrite16BitODI) (unsigned char Circuit, AASiSlaveAddr
Address, AASi16BitData Out);
```

Parameter:

Circuit:	AS-i Master Kreis
Address:	Adresse des Slaves
ExpResLen:	Erwartete Länge der Antwort (-1 = unbekannt)
Out:	4 Kanäle mit 16 Bit Werten Word 0 : Kanal 1 Word 1 : Kanal 2 Word 2 : Kanal 3 Word 3 : Kanal 4

Return:—

4.1.35 AS-i Read 16Bit ODI

AS-i Read 16Bit ODI liest vier 16bit ODI Kanäle eines AS-i Slave mit z.B. Analog-Slave-Profil 7.3 oder 7.4.

```
int (*AASiRead16BitODI) (unsigned char Circuit, AASiSlaveAddr
Address, AASi16BitData In);
```

Parameter:

Circuit:	AS-i Master Kreis
Address:	Adresse des Slaves
In:	4 Kanäle mit 16 Bit Werten Word 0 : Kanal 1 Word 1 : Kanal 2 Word 2 : Kanal 3 Word 3 : Kanal 4

4.1.36 AS-i Read 16Bit IDI

AS-i Read 16Bit IDI liest vier 16bit IDI Kanäle eines AS-Interface Slave mit z.B. Analog-Slave-Profil 7.3 oder 7.4.

```
int (*AASiRead16BitIDI) (unsigned char Circuit, unsigned char
Address, AASi16BitData In);
```

Parameter:

Circuit:	AS-i Master Kreis
Address:	Adresse des Slaves
In:	4 Kanäle mit 16 Bit Werten Word 0 : Kanal 1 Word 1 : Kanal 2 Word 2 : Kanal 3 Word 3 : Kanal 4

Ausgabedatum: 21.02.2013

4.1.37 AS-i Read Ctrl Acc ODI

AS-i Read Ctrl Acc ODI liest die Control III Berechtigungen, um Ausgangsdaten von Slaves zu schreiben.

```
int  (*AASiReadCtrlAccODI) ( unsigned char Circuit, AASiCtrlAc-
cODI ODI );
```

Parameter:

`Circuit:` AS-i Master Kreis

Return:

`ODI:` 32 Byte Ctrl Zugangsdaten

4.1.38 AS-i Write Ctrl Acc ODI

AS-i Write Ctrl Acc ODI schreibt die Control III Berechtigungen der Slaves, um Ausgangsdaten zu verändern.

```
int  (*AASiWriteCtrlAccODI) ( unsigned char Circuit, AASiCtrlAc-
cODI ODI,   AASiSlaveAddr First, unsigned char Amount );
```

Parameter:

`Circuit:` AS-i Master Kreis
`ODI:` 32 Byte Ctrl Zugangsdaten
`First:` Index des ersten Slaves
`Amount:` Anzahl der folgenden Slaves nach First

Return:—

Ausgabedatum: 21.02.2013

4.1.39 AS-i Read Ctrl Acc AODI

AS-i Read Ctrl Acc AODI liest die Control III Berechtigungen, um analoge Ausgangsdaten von Slaves zu schreiben.

```
int  (*AASiReadCtrlAccAODI) ( unsigned char Circuit, AASiCtrlAc-
cAODI AODI );
```

Parameter:

`Circuit:` AS-i Master Kreis

Return:

`AODI:` 16 Byte Ausgangsdaten Ctrl Zugang

Bit	7	6	5	4	3	2	1	0	Bit	7	6	5	4	3	2	1	0
Kanal	3	2	1	0	3	2	1	0	Kanal	3	2	1	0	3	2	1	0
Byte	D3	D2	D1	D0	D3	D2	D1	D0	Byte	D3	D2	D1	D0	D3	D2	D1	D0
0	Slave 1/1A				Slave 0/0A				1	Slave 3/3A				Slave 2/2A			
2	Slave 5/5A				Slave 4/4A				3	Slave 7/7A				Slave 6/6A			
4	Slave 9/9A				Slave 8/8A				5	Slave 11/11A				Slave 10/10A			
6	Slave 13/13A				Slave 12/12A				7	Slave 15/15A				Slave 14/14A			
8	Slave 17/17A				Slave 16/16A				9	Slave 19/19A				Slave 18/18A			
10	Slave 21/21A				Slave 20/20A				11	Slave 23/23A				Slave 22/22A			
12	Slave 25/25A				Slave 24/24A				13	Slave 27/27A				Slave 26/26A			
14	Slave 29/29A				Slave 28/28A				15	Slave 31/31A				Slave 30/30A			
16	Slave 1/1B				Slave 0/0B				17	Slave 3/3B				Slave 2/2B			
18	Slave 5/5B				Slave 4/4B				19	Slave 7/7B				Slave 6/6B			
20	Slave 9/9B				Slave 8/8B				21	Slave 11/11B				Slave 10/10B			
22	Slave 13/13B				Slave 12/12B				23	Slave 15/15B				Slave 14/14B			
24	Slave 17/17B				Slave 16/16B				25	Slave 19/19B				Slave 18/18B			
26	Slave 21/21B				Slave 20/20B				27	Slave 23/23B				Slave 22/22B			
28	Slave 25/25B				Slave 24/24B				29	Slave 27/27B				Slave 26/26B			
30	Slave 29/29B				Slave 28/28B				31	Slave 31/31B				Slave 30/30B			

Tab. 4-8. Ausgangsdatenabbild AODI

Ausgabedatum: 21.02.2013

4.1.40 AS-i Write Ctrl Acc AODI

AS-i Write Ctrl Acc AODI schreibt die Control III Berechtigungen der Slaves, um analoge Ausgangsdaten zu verändern.

```
int  (*AASiWriteCtrlAccAODI)( unsigned char Circuit, AASiCtrlAc-
cAODI AODI, AASiSlaveAddr First, unsigned char Amount;
```

Parameter:

Circuit:	AS-i Master Kreis
AODI:	16 Byte Ctrl Zugangsdaten (siehe Tab. <Ausgangsdatenabbild AODI>)
First:	Index des ersten Slaves
Amount:	Anzahl der folgenden Slaves nach First

Return:—

4.1.41 Ctrl Init Timer

Initialisierung eines Timer Interrupts

```
int      (*CCtrlInitTimer)   (unsigned   long   ticks_ms,   void
(*timer_func)( void ) );
```

Parameter:

Circuit:	AS-i Master Kreis
ticks_ms:	Interrupt Time in ms
timer_func:	Callback Funktion des Timer-Interrupts

Return:—

4.1.42 Ctrl Delay

Verzögerungsfunktion

```
int  (*CCtrlDelay)  ( unsigned long ticks_ms );
```

Parameter:

Circuit:	AS-i Master Kreis
ticks_ms:	Verzögerung in ms

Return:—

Ausgabedatum: 21.02.2013

4.1.43 Ctrl Init wgd

Initialisierung eines Watchdogs für Control III

```
int  (*CctrlInitWdg)  ( unsigned long ticks );
```

Parameter:

`Circuit:` AS-i Master Kreis
`ticks:` Watchdog Zeit in ms

Return:—

4.1.44 Ctrl Trigger wdg

Triggern des Control III Watchdogs

```
int  (*CctrlTriggerWdg)  ( void );
```

Parameter: —

Return:—

4.1.45 Ctrl Eval Cycle time

Die Funktion ermittelt die Zykluszeit des Control III Programms.

```
int  (*CctrlEvalCycletime)  ( void );
```

Parameter: —

Return:—

4.1.46 Ctrl Read Parameter

Ctrl Read Parameter liest nichtflüchtige Daten aus dem Flash.

```
int  (*CctrlReadParameter)  ( unsigned char *buffer, unsigned
short len, unsigned short adr );
```

Parameter:

`len:` Länge des zu lesenden Speichers
`adr:` Adresse des ersten Bytes der zu lesenden Daten

Return:

`buffer:` Lese-Puffer

Ausgabedatum: 21.02.2013

4.1.47 Ctrl Write Parameter

Ctrl Write Parameter schreibt nichtflüchtige Daten in den Flash.

```
int  (*CctrlWriteParameter)  ( unsigned char *buffer, unsigned
short len,      unsigned short adr );
```

Parameter:

`buffer:` Schreib-Puffer
`len:` Länge des Puffers
`adr:` Adresse des ersten Bytes, an welches die Daten geschrieben werden.

Return:—

4.1.48 Ctrl Read Flags

Ctrl Read Flags liest die AS-i-Control-Flags.

```
int  (*CctrlReadFlags)  ( unsigned char *flags );
```

Parameter: —

Return:

`flags` AS-i-Control-Flags

4.1.49 Ctrl Write Flags

Ctrl Write Flags schreibt die AS-i-Control-Flags.

```
int  (*CctrlWriteFlags)  ( unsigned char  flags );
```

Parameter:

`flags:` AS-i-Control-Flags

Return:—

4.1.50 Crtl Read Key

Crtl Read Key liest benutzerdefinierten, eindeutigen Control III Schlüssel.

```
int  (*CctrlReadKey)  ( unsigned int *key );
```

Parameter: —

Return:

`key:` benutzerdefinierter Control III Schlüssel

Ausgabedatum: 21.02.2013

4.1.51 Ctrl printf

Printf-Funktion

```
int  (*CCtrlPrintf)  ( const char *format, ... );
```

Parameter: —

Return:—

4.1.52 Ctrl Breakpoint

Initialisierung des Debuggers; „Erster“ Breakpoint.

```
void (*CCtrlBreakpoint) (void);
```

Parameter: —

Return:—

4.1.53 Ctrl Display

Selbstdefinierbare Anzeige auf dem Display des Gateway

```
int  (*CCtrlDisplay)(  unsigned  char  mode,  cctrl_disp_t
disp_buffer );
```

Parameter:

mode	CCTRL_DISP_MODE_TRADITIONAL CCTRL_DISP_MODE_SPONTANEOUS
disp_buffer.show:	0 = löschen 1 = anzeigen
disp_buffer.type:	CCTRL_DISP_TYP_4LINES = 4 Zeilen Text CCTRL_DISP_TYP_BIGNUM = große Zahl + 1 Zeile Text
disp_buffer.time:	Zeit der Darstellung im Display (min 2 sek.)
disp_buffer.big:	Puffer für große Zahl
disp_buffer.lines:	Puffer für 4 Zeilen Text

Return: 0 = OK
!0 = nicht Ok

Ausgabedatum: 21.02.2013

5 Getting Started

Eclipse ist ein weitverbreitetes und quelloffenes Programmierwerkzeug zur Entwicklung von Software verschiedenster Art. Eclipse wurde für die Verwendung von **Control III** so angepasst, dass es dem Anwender bei der Realisierung seines Software-Programms für **Control III** unterstützt.

5.1 Installation von Eclipse Control III

Führen Sie die 'setup.exe' aus, wählen Sie ein Zielverzeichnis und klicken auf den Button 'Installieren'. Die Installation kann einige Minuten in Anspruch nehmen.

Hinweis!

Es werden keine Eintragungen im Windows-Startmenü vorgenommen. Die Dateien werden lediglich entpackt und im entsprechenden Verzeichnis abgelegt.

5.2 Freischalten von Control III

Sollte Ihr Gateway noch nicht für die Verwendung von **Control III** freigeschaltet sein, können Sie eine Freischaltung beantragen.

Sollten Sie bereits einen Freischaltcode für ihr Gateway besitzen, finden Sie unter den Konfigurationstools in Eclipse ein Unlock-Control-Tool (siehe Kap. <Unlock Control:>).

Wählen Sie hierfür ihr Gateway aus und geben Sie den entsprechenden Freischaltcode ein. Die **Control III** Funktion wird durch einen Klick auf 'Unlock/Lock' freigeschaltet. Nach erfolgreicher Aktivierung werden Sie dazu aufgefordert das Gateway neu zu starten. Weiter Informationen hierzu finden Sie im Dokument "Control_freischaltung.pdf".

Ausgabedatum: 21.02.2013

6. Verwenden von Eclipse

Nach erfolgreicher Installation können Sie direkt mit dem Erstellen eines eigenen **Control III**-Programmes fortfahren.

6.1 Starten

Folgen Sie dem Installationspfad (standardmäßig C:/Programme) zum Ordner "eclipse_control" und führen dort die "eclipse.exe" aus.

6.2 Eclipse Übersicht

Der Eclipse Startbildschirm ist übersichtlich in die wichtigsten Bereiche unterteilt.

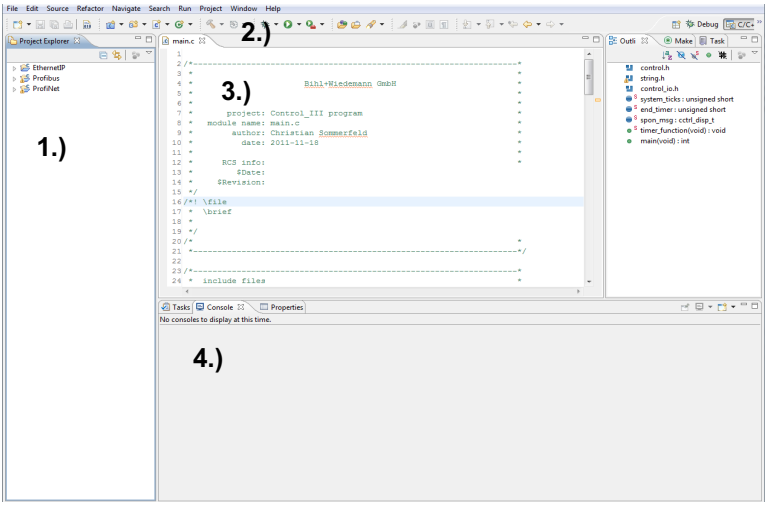


Abb. 6-1. Startfenster von Eclipse **Control III**

1. Projekt Explorer
2. Toolbar
3. Editor
4. Konsole

6.2.1 Der Projekt Explorer

Im Projekt Explorer finden Sie bei Neuinstallation drei Beispielprojekte, welche Sie direkt verwenden und anpassen können. Im Projekt Explorer können Sie alle Ihre Projekte verwalten, anpassen und testen. In jedem Projekt befinden sich nach Neuinstallation folgende Dateien:

- control_io.c
- control_io.h
- control.h
- main.c
- startup.c

Ausgabedatum: 21.02.2013

- *.ld
- *.mak
- settings.mak

6.2.2 Toolbar

In der Toolbar befinden sich alle nötigen Tools, um mit **Control III** zu arbeiten.

6.2.2.1 Compiler

- **clean**:
  Dient dazu den Projektordner zu säubern. Clean löscht alle durch das Kompilieren erstellte Dateien und Ordner.
- **debug**:
  Kompiliert das aktuell gewählte Projekt ohne Optimierungsgrad. Die daraus resultierende control.bin wird zum Debuggen verwendet. (Siehe Kap. <Der Debugger>).
- **release**:
  Kompiliert das aktuell gewählte Projekt. Die daraus resultierende control.bin ist auf Zeit optimiert, um schnellstmögliche Zykluszeiten ihres Programmcodes zu erreichen.

6.2.2.2 Debuggen

Mithilfe dieses Buttons wird unter anderem der Debugger gestartet. Sie müssen zum Debuggen noch den entsprechenden Port einstellen. Mehr Informationen finden Sie im Kap. <Start des Debuggers>.

Hinweis!

Steht das Control III Programm in einem 'Breakpoint', so steht das ganze Betriebssystem und die Feldbusschnittstelle wird nicht mehr bearbeitet.

6.2.2.3 Konfigurationstools

Der Button Konfigurationstools dient zur Kommunikation mit dem Gateway und hat folgende Funktionen:

Unlock Control:

Tool zum Freischalten von Control III im Gateway (siehe Kap. <Freischalten von Control III>).

Download Control:

Die Datei "control.bin" wird in das Gateway geschrieben.

Start Control:

Das **Control III**-Programm im Gateway wird gestartet.

Ausgabedatum: 21.02.2013

Hinweis!

Um ein neues Programm zu starten, muss nach dem Download das laufende Programm zuerst gestoppt werden.

Download + Start Control:

Das **Control III**-Programm wird zuerst gestopt, danach das neue Programm in das Gateway geladen und im Anschluss gestartet.

Stop Control:

Das **Control III**-Programm wird angehalten.

Set Auto Start:

Das Autostart-Flag wird gesetzt. Das **Control III**-Programm startet nach jedem Power-on automatisch.

Clear Auto Start:

Das Autostart-Flag wird gelöscht.

Read Merker:

Die Control III Merker werden gelesen und in der Konsole dargestellt.

Read Merker zyklisch:

Die Control III Merker werden gelesen und die Darstellung in der Konsole zyklisch aktualisiert. Hiermit können z.B. Variablen zur Laufzeit beobachtet werden.

Zykluszeit:

Die aktuellen Zykluszeiten werden angezeigt.

Reset Cycle Time:

Die Zykluszeit des **Control III** wird zurückgesetzt und neu berechnet.

6.2.3 Editor

In diesem Fenster wird der gesamte C-Code geschrieben und angepasst. Der Editor unterstützt des Weiteren auch Fehler im Syntax von 'C'.

6.2.4 Konsole

Die Konsole dient als Informationsfenster.Sie gibt bspw. Fehlermeldungen oder Statusmeldungen aus und zeigt nach dem Kompilieren den verwendeten Speicherplatz des **Control III** Programms an.

Ausgabedatum: 21.02.2013

6.3 Datei-Informationen

Wie bereits in Kap. <Der Projekt Explorer> beschrieben, liegen in jedem Projektordner verschiedene Dateien. In diesem Kapitel werden die einzelnen Dateien genauer beschrieben.

control_io.c

Diese Datei dient als Beispiel, um ein C Programm in mehrere Teilmodule zu zerlegen, um es so übersichtlicher und besser lesbar zu gestalten.

Die beiden Funktionen 'read_bit' und 'write_bit' lesen bzw. schreiben ein entsprechendes Ausgangs- oder Eingangsbit.

int read_bit (AASiProcessData idi, int slave_addr, int bit)
idi = Eingangsdaten der Slaves
slave_addr = die Adresse des entsprechenden Slaves
bit = Eingangsbit des Slaves (0-3)

void write_bit (AASiProcessData odi, int slave_addr, int bit, int value)
odi = Ausgangsdaten der Slaves
slave_addr = die Adresse des entsprechenden Slaves
bit = Ausgangsbit des Slaves (0-3)
value = Ausgangsbitwert (0 oder 1)

control_io.h

Header zu control_io.c. In dieser Datei befinden sich die Funktionsdefinitionen für 'read_bit' und 'write_bit'.

control.h

Die Header-Datei ’control.h’ beinhaltet alle Datentypen und Bibliotheksfunktionen. Außerdem erläutert diese gleichzeitig deren Funktions- und Verwendungsweise (siehe Kap. <Programmieren von Control III>).

main.c

Die Funktion ’main’ ist der „Startpunkt“ des eigentlichen Programmcodes. In ihr befindet sich auch die Hauptschleife des Programms (for ; ;).

startup.c

Diese Datei dient zur Initialisieren verschiedener Speicherbereiche. Die Datei ist für den Anwender ohne Bedeutung.

****.ld***

Ist das Linkerfile und die entsprechenden Speicherbereiche im Gateway fest. Die Datei ist für den Anwender ohne Bedeutung.

****.mak***

Diese Datei legt alle nötigen Informationen für den Compiler fest. Die Datei ist für den Anwender ohne Bedeutung.

Ausgabedatum: 21.02.2013

settings.mak

In dieser Datei wird der Kommunikationsport für das Gateway festgelegt. Näher Informationen finden Sie im Kap. <Einstellen der Schnittstelle>.

6.4 Einstellen der Schnittstelle

In jedem Projektordner befindet sich eine Datei 'settings.mak'. In dieser Datei können Sie den Port für die Kommunikation zum Gateway einstellen. Wählen Sie hierzu die entsprechende Datei aus und tragen Sie die Verbindung zum Gateway ein. Verwenden Sie bspw. den COM Port 3 an ihrem PC dann schreiben Sie in Zeile 30 der settings.mak `PORT=COM3`. Verwenden Sie eine Ethernet-Schnittstelle, dann tragen Sie bspw. `PORT=UDP:192.168.42.149` ein.

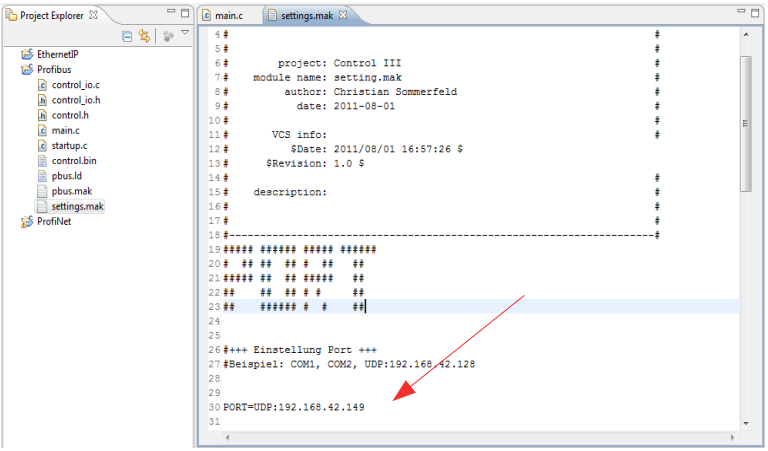


Abb. 6-2. Eclipse settings.mak

6.5 Anlegen eines neuen Projektes

Im Projektexplorer befinden sich nach einer Neuinstallation, für jedes Gateway mit **Conrtrol III** Beispielprojekte. Es ist somit möglich, direkt nach der Installation von Eclipse **Control III** mit der Programmierung zu starten.

Sollten Sie dennoch ein neues Projekt anlegen wollen, fahren Sie wie folgt fort:

- ☐ Wählen Sie unter 'File' -> 'New' ein neues 'C Projekt'.
- ☐ Vergeben sie einen neuen Projektnamen und wählen Sie ein leeres 'Makefile Project'.
- ☐ Bestätigen den Dialog mit 'Finish'.

Ausgabedatum: 21.02.2013

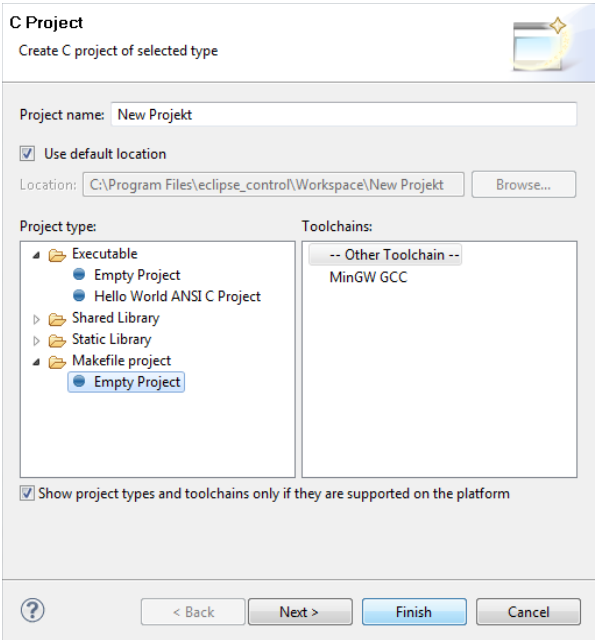


Abb. 6-3. Neues Control III Projekt

Sie finden nun im Projekt Explorer ihren neuen leeren Projektordner. Mit einem Rechtsklick auf das neue Projekt und 'Import...' können Sie alle nötigen Projektdateien für ihr Gateway hinzufügen.

- Wählen Sie 'File System' und klicken danach auf 'Next'.
- Wählen Sie nun 'Browse...' und navigieren zur Ihren Eclispe-Installation .../eclipse_control/Templates/
- Wählen Sie nun ihr verwendetes Gateway aus.
  - EthernetIP
  - Profibus
  - ProfiNET
- Wählen Sie 'Select All' und Finish.
- Um alle Einstellungen zu übernehmen, wählen Sie in folgendem Dialog 'Yes to All'.

6.6 Ein Beispielprojekt

In folgendem Kapitel wird die komplette Vorgehensweise vom Schreiben des Codes bis zum Starten im Gateway erläutert.

Ausgabedatum: 21.02.2013

6.6.1 Der C-Code

Als Beispiel dient ein Programm, welches nacheinander jede Sekunde die Ausgänge eines 4E/4A Slave mit der Adresse 1 setzt und wieder löscht. Ändern Sie hierfür die main.c wie folgt:

```
/*---------------------------------------------------------------------*
 *  include files                                                      *
 *---------------------------------------------------------------------*/
#include "control.h"
#include "string.h"
#include "control_io.h"

/*---------------------------------------------------------------------*
 *  local definitions                                                  *
 *---------------------------------------------------------------------*/

/*---------------------------------------------------------------------*
 *  external declarations                                              *
 *---------------------------------------------------------------------*/

/*---------------------------------------------------------------------*
 *  public data                                                        *
 *---------------------------------------------------------------------*/

/*---------------------------------------------------------------------*
 *  private data                                                       *
 *---------------------------------------------------------------------*/


static unsigned short system_ticks;
static unsigned short end_timer;

/*---------------------------------------------------------------------*
 *  private functions                                                  *
 *---------------------------------------------------------------------*/

static void timer_function ( void )
{
       /* timer interrupt every 10 ms */
       system_ticks++;
}

/*---------------------------------------------------------------------*
 *  public functions                                                   *
 *---------------------------------------------------------------------*/
int main ( void )
{
//initialization of the Debugger
//cctrl_func.CCtrlBreakpoint();
unsigned charctrl_flags;
int             i = 0;
int             x = 1;

AASiProcessData odi[2];
AASiProcessData idi[2];
AASiCtrlAccODI acc_odi;
AASiEcFlags     ecflags;
```

Ausgabedatum: 21.02.2013

```
/* We want to access all odis */
for (i=0;i<32;i++)
{
        acc_odi[i] = 0xFF;
}
cctrl_func.AASiWriteCtrlAccODI ( 0, acc_odi, 0, 64 );

/* init timer function with 10ms ticks */
cctrl_func.CCtrlInitTimer ( 10, timer_function );

/* init watchdog */
//cctrl_func.CCtrlInitWdg( 10 );

    // clear outputs from slave 1
    odi[0][0] = 0x00;

for(;;)
        {
                /* trigger watchdog */
                //cctrl_func.CCtrlTriggerWdg();

                 /* Define data exchange for AS-i Circuit 1 and 2*/
                 cctrl_func.AASiDataExchange(0, odi[0], idi[0],
&ecflags);
                 cctrl_func.AASiDataExchange(1, odi[1], idi[1],
&ecflags);

                //Timer 1 100 * 10ms = 1sec.
                if ( ((unsigned short)(system_ticks - end_timer)) > 100)
                {
                // set and clear outputs circuit=1, slave=1, output=0-3
                if (x == 1) write_bit(odi[0], 1, 0, 1);
                else if (x == 2) write_bit(odi[0], 1, 1, 1);
                else if (x == 3) write_bit(odi[0], 1, 2, 1);
                else if (x == 4) write_bit(odi[0], 1, 3, 1);
                else if (x == 5) write_bit(odi[0], 1, 0, 0);
                else if (x == 6) write_bit(odi[0], 1, 1, 0);
                else if (x == 7) write_bit(odi[0], 1, 2, 0);
                else if (x == 8) write_bit(odi[0], 1, 3, 0);
                x++;

                if (x == 9) x = 1;

                end_timer = system_ticks;
                }

                /* to check Cycletime */
                cctrl_func.CCtrlEvalCycletime();

                /*read flags if we should stop control*/
                cctrl_func.CCtrlReadFlags( &ctrl_flags );
                if ( !( ctrl_flags & CCTRL_FLAG_RUN ) )
                {
                return 1;
                }

        }
}
```

6.6.2 Kompilieren

Der erstellte C-Code muss nun für den Prozessor übersetzt werden. Wählen Sie hierfür „release“ bei der Option „Compiler“ in der Toolbar. Es wird ein “release“-Ordner erstellt und in Ihrem Projektordner befindet sich nun die entsprechende Binär-Datei ’control.bin’.

6.6.3 Download

Um das neu erstelle Programm in das Gateway zu laden wählen Sie in Ihrem Projektordner die Datei settings.mak und stellen den entsprechenden Port ihres Gateways ein. Weitere Informationen hierfür finden Sie im Kap. <Einstellen der Schnittstelle>.

Als Nächsten wählen Sie in der Toolbar unter den Konfigurationstools: Download Control. Bei erfolgreichem Download erscheint in der Konsole von Eclipse folgende Meldung:

```
-----------------------
 ++++ CONTROL III ++++
-----------------------


 communication port set to UDP:192.168.42.157.

 writing C-Control control.bin to Master ...
 ....................................
 o.k.

 closing control.bin ...

 have a nice day.
```

6.6.4 Starten von Control III

Um das Programm zu starten und zu testen, wählen Sie nun unter der Toolbar / Konfigurationstools den Button Start Control. Im Display erscheint folgende Meldung:

6.7 Der Debugger

Ein Debugger ist ein Programmierwerkzeug und dient dem Diagnostizieren und Auffinden von Programmierfehlern. Wird die Debuggerfunktion verwendet, so steht bei der Diagnose das gesamte Betriebssystem.

Hinweis!

Die Debugg-Funktion dient lediglich dazu, ihr Programm in der Ausführung zu testen. Sie können mit dem Debugger keine Diagnose ihres Hardwareaufbaus durchführen.

Ausgabedatum: 21.02.2013

6.7.1 Initialisierung

Um den Debugger verwenden zu können, muss dieser im C-Code initialisiert werden. Dies geschieht über folgende Codezeile:

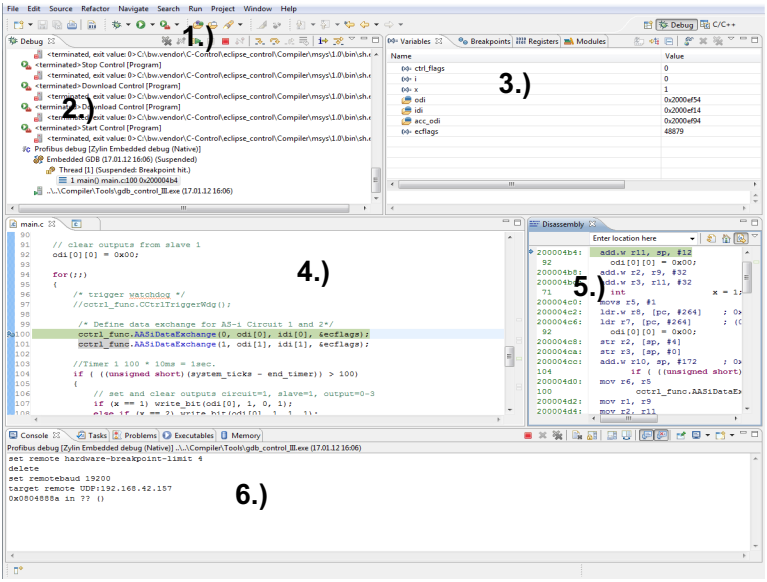
```
//initialization of the Debugger
cctrl_func.CCtrlBreakpoint();
```

Durch das Verwenden dieser Zeile wird das Betriebssystem gestoppt und Eclipse kann mit dem Prozessor kommunizieren. Der Debugger bekommt zu diesem Zeitpunkt die Daten des Programms mitgeteilt und springt in den nächsten, vom Anwender definierten, Haltepunkt des **Control III** Programms.

Hinweis!

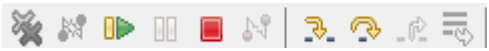
Die Initialisierung des Debuggers sollte nicht in der Hauptschleife des Programmcodes (for (;;)) erfolgen, da dieser Haltepunkt nicht über Eclipse gesteuert wird und daher nicht beendet werden kann.

6.7.2 Übersicht Debugger



1. Control Panel
2. Tasks
3. Debugger Übersicht
4. Codeübersicht
5. Disassembly
6. Konsole

6.7.2.1 Das Control Panel



Das Control Panel dient dazu, den Debugger zu steuern.

Resume (F8):

Mit dem Button 'Resume' läuft das **Control III** Programm weiter bis es auf einen neuen Haltepunkt stößt. Wird innerhalb von 10 Sekunden kein neuer Haltepunkt erreicht so beendet sich der Debugger in Eclipse automatisch und es wird eine entsprechende Meldung in der Konsole ausgegeben.

Hinweis!

Bei Benutzung des Debuggers sind keine Delays länger als 10 Sekunden möglich, da der Debugger sonst automatisch beendet wird. Dies dient dazu, bei fehlender oder falscher Kommunikation den Debugger zu beenden.

Terminate (Ctrl + F2):

Terminate beendet den Debugger und lässt das **Control III** Programm weiterlaufen, auch wenn noch Haltepunkte über Eclipse im Code vorhanden sind.

Step Into (F5):

Der Button ’Step Into’ wird dazu benutzt, um eine Programmzeile im Code weiterzuspringen.

Hinweis!

Die StepInto-Funktion kann nur zuverlässig funktionieren, wenn der Programmcode als „debug“ übersetzt worden ist.

Step Over (F6):

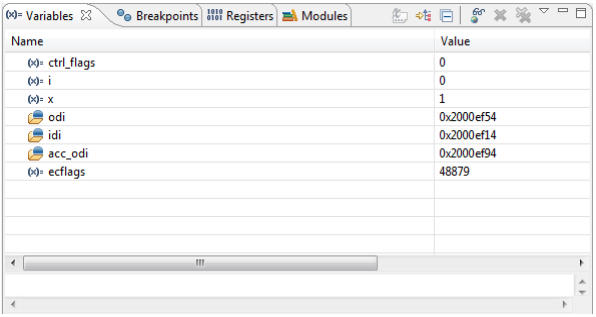
’Step Over’ wird verwendet, um bspw. einen Funktionsaufruf zu überspringen.

6.7.2.2 Tasks

In diesem Fenster werden alle verwendeten Programme, welche über Eclipse gesteuert werden, angezeigt. Dieses Fenster ist für den Anwender von keiner großer Bedeutung.

Ausgabedatum: 21.02.2013

6.7.2.3 Debugger Übersicht



Variablen:

Im Reiter Variablen in der „Debugger-Übersicht“ können alle vorhandenen Variablen und deren Werte angezeigt werden. Mit einem Rechtsklick in das Fenster können mit 'Add Global Variables...' weitere Variablen der Ansicht hinzugefügt werden. Der Wert der Variablen wird immer in Hex angezeigt. Die Ansicht kann mit einem Rechtsklick und dem Punkt 'Format' auf binär oder dezimal geändert werden.

Breakpoints:

Im Reiter Breakpoints befindet sich eine Übersicht über alle in Eclipse gesteuerten bzw. verwendeten Haltepunkte inklusive der Programmzeile. Einzelne Haltepunkte können mit einem Rechtsklick entfernt werden.

6.7.2.4 Codeübersicht

In diesem Fenster sehen Sie jederzeit, an welcher Stelle im Programmcode sich der Debugger gerade befindet. Hier können Sie auch mit einem Doppelklick neben der entsprechenden Codezeile, einen neuen Haltepunkt setzten oder löschen.

Hinweis!

Sobald der Debugger gestartet wird, sieht man zuerst ein leeres Fenster mit dem Inhalt: „Source not found“. Der Debugger befindet sich zu diesem Zeitpunkt im Betriebssystem und kennt den dazugehörigen C-Code nicht.

6.7.2.5 Disassembly

Das Disassembly-Fenster zeigt genau wie die Codeübersicht an, an welcher Stelle sich der Debugger gerade befindet. Hier werden sowohl die Speicheradresse als auch der dazugehörige assembler-Code angezeigt.

6.7.2.6 Konsole

Die Konsole dient als Ausgabe Fenster und informiert Sie über den Status des Debuggers.

Ausgabedatum: 21.02.2013

6.7.3 Start des Debuggers

Der Debugger wird über das Control Panel gestartet.

Bevor Sie Verbindung mit dem Target aufnehmen können, müssen Sie unter 'Debug-Configuration' die Schnittstelle zum Gateway definieren. Klicken Sie hierzu auf den Reiter 'Commands' und geben unter target remote die Schnittstelle an. (bspw. target remote `UDP:192.168.42.33 oder COM3).`

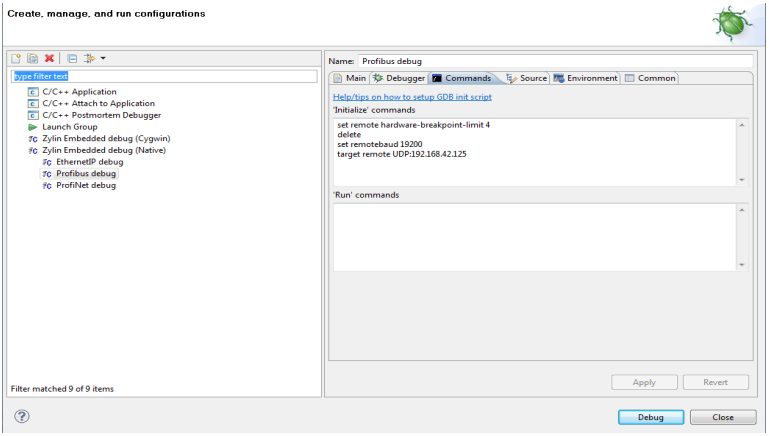


Abb. 6-4. Debug Configuration...

6.7.4 Beispiel

Als Beispiel für den Debugger dient ein Programm, welches in jedem Hauptschleifendurchlauf ein Ausgangsbit (0-3) des Slaves mit der Adresse 1 setzt. Verwenden Sie hierzu folgenden Programmcode.

Erstellen Sie nun zwei Haltepunkte, indem Sie am Rand des Programmcodes auf die entsprechende Code-Zeile doppelklicken. In unserem Beispiel verwenden wir die beiden Zeilen 103 und 109. Die Zeile, in der ein Haltepunkt eingefügt wurdee, wird durch einen Punkt markiert. Diese Zeilen werden später in der Debugger Übersicht unter Breakpoint dargestellt und dem Prozessor bei der Initialisierung mitgeteilt.

Ausgabedatum: 21.02.2013

```
*---------------------------------------------------------------------*
 *  include files                                                      *
 *---------------------------------------------------------------------*/
#include "control.h"
#include "string.h"
#include "control_io.h"

/*---------------------------------------------------------------------*
 *  local definitions                                                  *
 *---------------------------------------------------------------------*/

/*---------------------------------------------------------------------*
 *  external declarations                                              *
 *---------------------------------------------------------------------*/

/*---------------------------------------------------------------------*
 *  public data                                                        *
 *---------------------------------------------------------------------*/

/*---------------------------------------------------------------------*
 *  private data                                                       *
 *---------------------------------------------------------------------*/


static unsigned short system_ticks;
static unsigned short end_timer;

/*---------------------------------------------------------------------*
 *  private functions                                                  *
 *---------------------------------------------------------------------*/

static void timer_function ( void )
{
       /* timer interrupt every 10 ms */
       system_ticks++;
}

/*---------------------------------------------------------------------*
 *  public functions                                                   *
 *---------------------------------------------------------------------*/
```

```
int main ( void )
{
        //initialization of the Debugger
        cctrl_func.CCtrlBreakpoint();

        unsigned charctrl_flags;
        int      i = 0;
        int      x = 1;

        AASiProcessData odi[2];
        AASiProcessData idi[2];
        AASiCtrlAccODI acc_odi;
        AASiEcFlags ecflags;

        /* We want to access all odis */
        for (i=0;i<32;i++)
        {
                acc_odi[i] = 0xFF;
        }
        cctrl_func.AASiWriteCtrlAccODI ( 0, acc_odi, 0, 64 );

        /* init timer function with 10ms ticks */
    cctrl_func.CCtrlInitTimer ( 10, timer_function );

    /* init watchdog */
        //cctrl_func.CCtrlInitWdg( 10 );

    // clear outputs from slave 1
    odi[0][0] = 0x00;
```

```
        for(;;)
        {
                /* trigger watchdog */
                //cctrl_func.CCtrlTriggerWdg();

                 /* Define data exchange for AS-i Circuit 1 and 2*/
                 cctrl_func.AASiDataExchange(0, odi[0], idi[0],
&ecflags);
                 cctrl_func.AASiDataExchange(1, odi[1], idi[1],
&ecflags);

                 if (x == 1)
                 {
                 write_bit(odi[0], 1, 0, 1);
                 }
                 else if (x == 2)
                 {
                 write_bit(odi[0], 1, 1, 1);
                 }
                 else if (x == 3)
                 {
                 write_bit(odi[0], 1, 2, 1);
                 }
                 else if (x == 4)
                 {
                 write_bit(odi[0], 1, 3, 1);
                 x = 1;
                 }
                 x++;

                /* check Cycletime */
                cctrl_func.CCtrlEvalCycletime();

                /*read flags if we should stop control*/
                cctrl_func.CCtrlReadFlags( &ctrl_flags );
                if ( !( ctrl_flags & CCTRL_FLAG_RUN ) )
                {
                return 1;
                }

        }
}
/*---------------------------------------------------------------------*
 *  eof                                                                *
 *---------------------------------------------------------------------*/
```

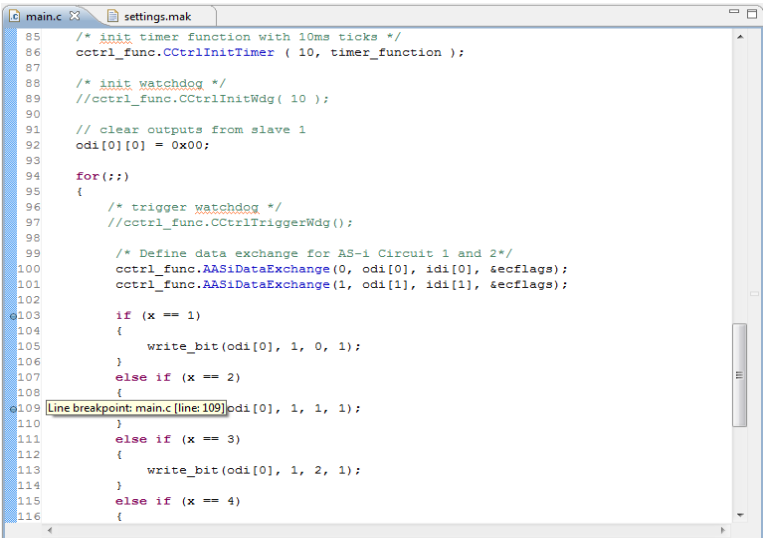
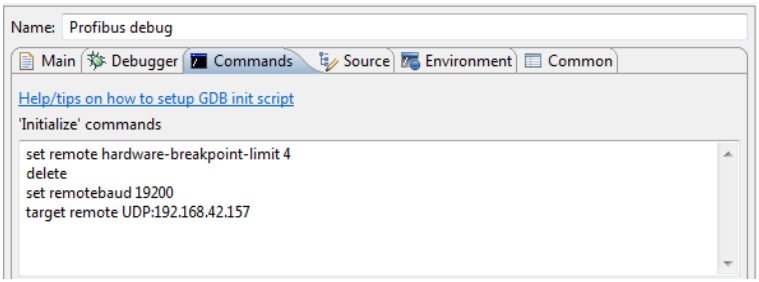


Abb. 6-5. Markierung eines Breakpoints im Programmcode

Kompilieren Sie nun das Programm, indem Sie in der Toolbar unter Compiler auf ’Debug’ klicken. Es wird ein neues control.bin-File ohne Optimierungsgrad erzeugt. Laden Sie dieses File, wie schon im Kap. <Editor> beschrieben, in das Gateway.

6.7.4.1 Start des Debuggers

Bevor Sie mit dem Debuggen beginnen können, müssen Sie den Port einstellen. Klicken Sie in der Toolbar unter Debugger auf 'Debug Configurations...' Öffnen Sie hier den Reiter 'Commands' und tragen unter 'target remote' ihre Schnittstelle (bspw. UDP:192.168.42.157 oder COM1) zum Gateway ein und wählen 'Apply'.



Sollte ihr **Control III** Programm bereits in einem Haltepunkt stehen, so können Sie den Debugger direkt über 'Debug' starten. Ein Haltepunkt wird ihnen über das Display im Gateway mit der Nummer '79' angezeigt.

```
79
breakpoint
```

Sollte dies nicht der Fall sein, beenden Sie die Eingabe mit 'Close' und starten ihr Programm. Sie können den Debugger jetzt direkt mit einem Klick auf bspw. 'Profibus debug' in der Toolbar starten.

6.7.4.2 Debugger benutzen

Nachdem Sie den Debugger gestartet haben, wird dieser für die Verwendung konfiguriert. Das Debugging-Fenster wird automatisch von Eclipse geöffnet. Sie sehen jetzt unter Breakpoints den ersten Haltepunkt, der vom Programmcode ausgeführt wird. Dies ist die Initialisierung des Debuggers. Sie bekommen in diesem Fall ein leeres Fenster angezeigt.Klicken Sie hierfür auf 'Resume'. Der Programmcode wird bis zu der entsprechende Zeile mit dem ersten Haltepunkt ausgeführt und gestoppt.

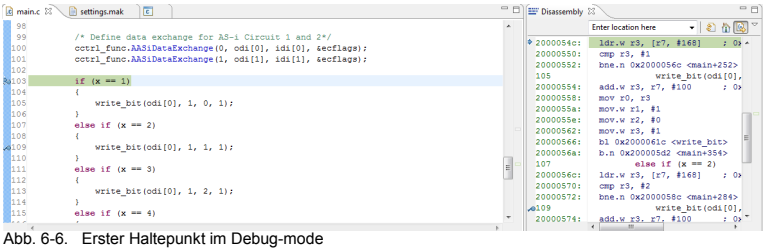


Abb. 6-6. Erster Haltepunkt im Debug-mode

Des Weiteren können Sie mit einem Klick auf 'Variables' in der Debugger Übersicht die Werte aller verwendeten Variablen anzeigen lassen (siehe hierzu auch Kap. <Debugger Übersicht>). Klicken Sie auf ’Resume’. Der Debugger bleibt an erneut an dieser Stelle stehen und nicht am zweiten Haltepunkt. Dies liegt daran, dass der zweite Haltepunkt erst erreicht wird, wenn die Variable 'x' in unserem Beispielprogramm den Wert 2 hat. Sie können sich den Wert der Variable anzeigen lassen, indem Sie mit dem Mauszeiger über die entsprechende Variable gehen. Die Variable x hat nun den Wert 2. Betätigen Sie erneut ’Resume’, um zum Haltepunkt in Zeile 109 zu springen.

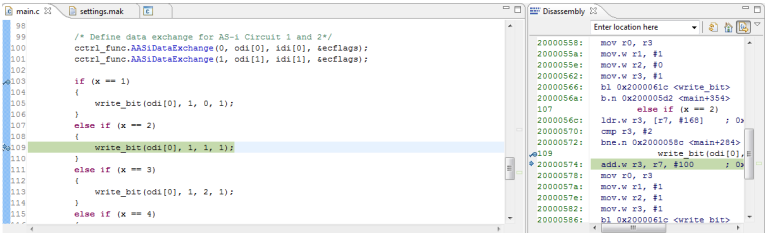


Abb. 6-7. Zweiter Haltepunkt im Debug-mode

Bei dieser Codezeile bietet sich die Möglichkeit zwischen ’Step Over’ oder ’Step Into’. Bei ’Step over’ springt das Programm in Zeile 120, also übersprint den Funktionsaufruf ’write_bit(...)’ und macht mit der nächsten gültigen Zeile im Code weiter. Bei ’Step Into’ wird die entsprechende Datei (control_io.c) geöffnet und der ’Debugg-Modus’ an der entsprechenden Stelle fortgeführt. Möchten Sie wieder zum nächsten Haltepunkt springen, so wählen Sie erneut den Button 'Resume'. Beenden Sie den Debugger mit einem Klick auf 'Terminate'.

Ausgabedatum: 21.02.2013

7. Technische Daten

Im folgenden Kapitel finden Sie eine Übersicht über alle technischen Kenndaten von Control III.

7.1 Übersicht

- 28 kByte Programmspeicher (variable aufteilbar in ROM/RAM)
- 1kByte nichtflüchtige Parameter
- 256 Byte Merkerbereich
- Steuerprogramm und Parameter auch auf der Chipkarte
- Ein konfigurierbarer Timerinterrupt, aus dem beliebig viele Timer abgeleitet werden können.
- Programmierbare Timerzeiten von 1 bis 2^{32} ms
- Bis zu 248 E/A und 248 Analogwerte mittels AS-i Slaves
- Eindeutige 32 Bit Kennung im Gerät
- Einfache Ermittlung der Zykluszeit
- Eclipse mit GCC und GDB als komplette Entwicklungsumgebung

7.2 Merker

Der Merkerbereich ist transparent und wird durch den Anwender verwaltet.

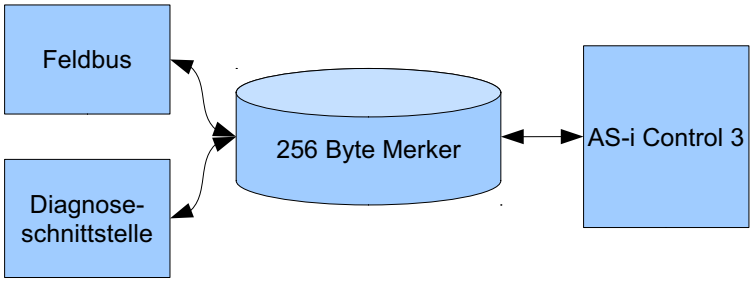


Abb. 7-8. Darstellung Merkerbereich

7.3 Nichtflüchtige Parameter

Die nichtflüchtigen Parameter sind transparent und werden durch den Anwender verwaltet.

Ausgabedatum: 21.02.2013

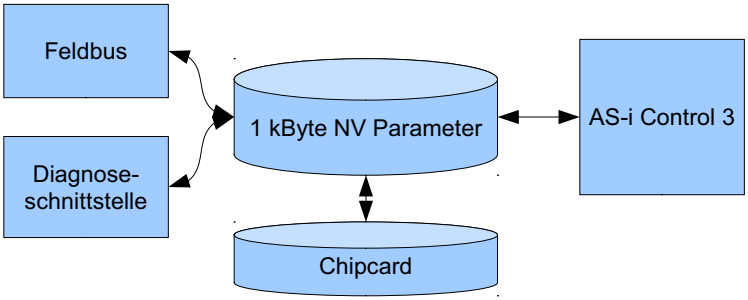


Abb. 7-9. Darstellung nichtflüchtige Parameter

7.4 Zugriffsrechte auf den Ausgangsdatenbereich

Feldbus und Control III Programm können gleichzeitig Ausgänge setzten. Die Zugriffsrechte können bitweise oder Kanalweise vergeben werden.

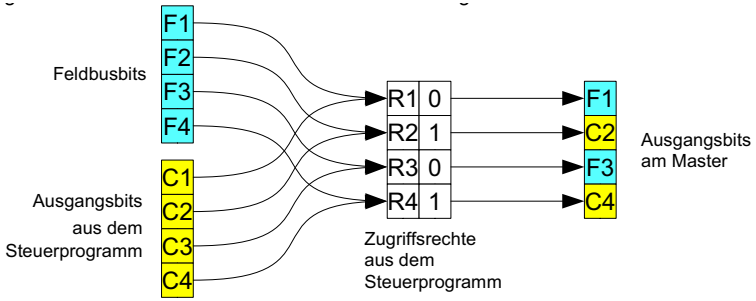


Abb. 7-10. Darstellung Zugriffsrechte Ausgangsdatenbereich

Ausgabedatum: 21.02.2013

8. Fehlermeldungen

Dieses Kapitel soll Ihnen bei eventuell auftretenden Problemen helfen, die Fehler zu erkennen und zu beheben.

8.1 error: control not activated!

Sollte die Eclipse-Konsole folgende Fehlermeldung anzeigen so muss Control III für Ihr Gateway noch freigeschaltet werden (siehe Kap. <Freischalten von Control III>).

```
 ----------------------
  ++++ CONTROL III ++++
 ----------------------


  communication port set to UDP:192.168.42.149.


error: control not activated!

  have a nice day.
```

Abb. 8-11. error: control not activated

8.2 error: wrong control version

Steht in der Eclipse-Konsole die Meldung 'error: wrong control version', so besitzen Sie ein Gateway mit einer anderen Control-Version, welche nicht über die Fähigkeit der C-Programmierung verfügt. Wenden Sie sich in diesem Fall an den Hersteller Support.

```
 ------------------------
  ++++ CONTROL III ++++
 ------------------------


  communication port set to COM4.


error: wrong control version!

  have a nice day.
```

Abb. 8-12. error: wrong control version!

8.3 Launching problem

Bei dieser Fehlermeldung fehlt Eclispe die Zuordnung zu Ihrem Projekt. Klicken Sie hierzu einfach in das Editorfenster und führen Sie ihre Eingabe erneut aus.

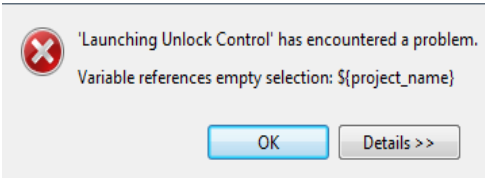


Abb. 8-13. Launching problem

8.4 Es wird keine oder eine falsche Zykluszeit angezeigt.

Sollten Sie die Zykluszeit über Eclipse auslesen und immer den Wert 0 angezeigt bekommen, so fehlt im Programmcode die Zeile:

```
/* check Cycletime */
cctrl_func.CCtrlEvalCycletime();
```

8.5 Das Gateway geht in keinen Haltepunkt.

Sollte das Gateway nicht in einem Haltepunkt stoppen, besteht die Möglichkeit, dass das auto-start-flag gesetzt ist oder es fehlt im Programmcode folgende Zeile zur Initialisierung des Debuggers:

```
//initialization of the Debugger
cctrl_func.CCtrlBreakpoint();
```

8.6 Das Programm geht immer in einen Haltepunkt.

Ist dies der Fall und es ist keine Initialisierung des Debuggers im Code vorhanden, so liegt der Fehler im Programmcode. Die häufigste Ursache hierfür ist ein uninitialisierter Zeiger/Pointer. Bitte prüfen Sie ihren Programmcode. Sollten Sie zudem noch das Autostart-Flag gesetzt haben, so können Sie beim Start des Gateways einen Reset durchführen. Betätigen Sie hierfür die beiden Tasten 'Mode' und 'Set' und schalten das Gateway ein. Der vorhandene Programmcode wird gelöscht und das Gateway startet wieder.

8.7 Es lassen sich keine Ausgänge durch Control III beeinflussen.

Prüfen Sie, ob die Slaves richtig projektiert sind und das Gateway keinen Konfigurationsfehler anzeigt. Führen sie ggf. die Projektierung der Slaves erneut durch. Sollte der Fehler immer noch bestehen, dann fehlen möglicherweise die Zugriffsrechte. Diese werden durch folgende Funktion vergeben:

```
cctrl_func.AASiWriteCtrlAccODI ( … );
```

Weitere Informationen finden Sie hierzu im Kap. <AS-i Read Ctrl Acc AODI>.

Ausgabedatum: 21.02.2013