


Tutorial

Anybus[®] CompactCom Implementation

Doc.Id. SCM-1200-046
Rev. 1.00

HMS Industrial Networks AB


Germany +49 - 721 - 96472 - 0
Japan +81 - 45 - 478 - 5340
Sweden +46 - 35 - 17 29 20
U.S.A. +1 - 312 - 829 - 0601
France +33 - 3 89 32 76 76
Italy +39 - 347 - 00894 - 70
China +86 - 10 - 8532 - 3183


ge-sales@hms-networks.com
jp-sales@hms-networks.com
sales@hms-networks.com
us-sales@hms-networks.com
fr-sales@hms-networks.com
it-sales@hms-networks.com
cn-sales@hms-networks.com



Important User Information

This document is intended to promote a good understanding of how to develop an application using Anybus CompactCom. The document describes functions and actions that are common to all products in the Anybus CompactCom family. For information regarding the specific Anybus CompactCom modules, consult the Anybus CompactCom Network Interface Appendices.

Please note that the reader of this document is expected to be familiar with high level software design, and communication systems in general. The document describes a simple application using an Anybus CompactCom module. For more advanced applications, please consult the Anybus CompactCom Software Design Guide and the respective network interface appendices.

Liability

Every care has been taken in the preparation of this document. Please inform HMS Industrial Networks AB of any inaccuracies or omissions. The data and illustrations found in this document are not binding. We, HMS Industrial Networks AB, reserve the right to modify our products in line with our policy of continuous product development. The information in this document is subject to change without notice and should not be considered as a commitment by HMS Industrial Networks AB. HMS Industrial Networks AB assumes no responsibility for any errors that may appear in this document.

There are many applications of this product. Those responsible for the use of this device must ensure that all the necessary steps have been taken to verify that the applications meet all performance and safety requirements including any applicable laws, regulations, codes, and standards.

HMS Industrial Networks AB will under no circumstances assume liability or responsibility for any problems that may arise as a result from the use of undocumented features, timing, or functional side effects found outside the documented scope of this product. The effects caused by any direct or indirect use of such aspects of the product are undefined, and may include e.g. compatibility issues and stability issues.

The examples and illustrations in this document are included solely for illustrative purposes. Because of the many variables and requirements associated with any particular implementation, HMS Industrial Networks AB cannot assume responsibility for actual use based on these examples and illustrations.

Intellectual Property Rights

HMS Industrial Networks AB has intellectual property rights relating to technology embodied in the product described in this document. These intellectual property rights may include patents and pending patent applications in the US and other countries.

Trademark Acknowledgements

Anybus ® is a registered trademark of HMS Industrial Networks AB. All other trademarks are the property of their respective holders.

Warning:	This is a class A product. In a domestic environment this product may cause radio interference in which case the user may be required to take adequate measures.
ESD Note:	This product contains ESD (Electrostatic Discharge) sensitive parts that may be damaged if ESD control procedures are not followed. Static control precautions are required when handling the product. Failure to observe this may cause damage to the product.

Table of Contents

Preface	About This Document	1
	Related Documents.....	1
	Document History	1
	Conventions & Terminology.....	2
	<i>Definitions</i>	2
Chapter 1	How to use this Document	3
Chapter 2	Introduction to the Anybus CompactCom	4
Chapter 3	Tutorial.....	5
	Introduction	5
	Connect the Anybus CompactCom Module.....	6
	<i>Host Interface Signals</i>	6
	Set Operating Mode.....	8
	Telegrams	9
	Parallel Interface Mode	10
	<i>Initial Handshake</i>	10
	<i>DPRAM</i>	11
	<i>Sending Telegrams - Parallel Communication</i>	13
	Serial Interface Mode.....	18
	<i>Initial Handshake</i>	18
	<i>Serial Telegram Frame</i>	19
	<i>Sending Telegrams - Serial Communication</i>	21
	Setup continued.....	28
	<i>Anybus CompactCom State Machine</i>	28
	<i>Accessing the Anybus CompactCom</i>	29
	<i>Mapping ADIs</i>	30
	<i>Setup Complete</i>	32
	Network Initialization.....	33
	Further Configuration and Certification.....	34

Chapter 4	Resources	35
	Categorization of Functionality.....	35
	<i>Basic</i>	35
	<i>Extended</i>	35
	<i>Advanced</i>	35
	Design Guides	36
	<i>Anybus CompactCom Software Design Guide</i>	36
	<i>Anybus CompactCom Hardware Design Guide</i>	36
	Network Interface Appendices	36
	Drivers	36
	Starter Kit	36
	Drive Profiles	37
Appendix A	Trace, DeviceNet	38
Appendix B	Trace, Profibus DP-V1	44
Support	Support	50

P. About This Document

For more information, documentation etc., please visit the HMS website, www.anybus.com.

P.1 Related Documents

In the tutorial you will find references to these documents in the margins, showing the abbreviation of the document name within a frame in the right margin. All documents are available at www.anybus.com.

Document	Abbr.	Doc. Id.	Author
Anybus CompactCom Software Design Guide	SWDG	SCM-1200-037	HMS
Anybus CompactCom Hardware Design Guide,	HWDG	SCM-1200-061	HMS
Anybus CompactCom Standard Driver Implementation Guide	SDRV	SCM-1200-043	HMS
Anybus CompactCom Lite Driver Implementation Guide	LDRV	SCM-1200-042	HMS
Anybus CompactCom Network Interface Appendices	"network" APP.	See www.anybus.com	HMS

P.2 Document History

Summary of Recent Changes (... 1.00)

Change	Page(s)
-	-
-	-
-	-

Revision List

Revision	Date	Author(s)	Chapter(s)	Description
1.00	2010-04-19	KeL	-	First official release

P.3 Conventions & Terminology

The following conventions are used throughout this manual:

- Numbered lists provide sequential steps
- Bulleted lists provide information, not procedural steps
- The terms ‘Anybus’ or ‘module’ refers to the Anybus CompactCom module.
- The terms ‘host’ or ‘host application’ refers to the device that hosts the Anybus module.
- Hexadecimal values are normally written in the format NNNNh, where NNNN is the hexadecimal value. When a hexadecimal value is present in code or in pseudo code in an example, the value is written in the format 0xnnnn, where nnnn is the hexadecimal value.

P.3.1 Definitions

Word	Explanation
Process data	Fast cyclic network I/O data
Acyclic data	Data that is updated on demand, e.g. application parameters
ADI	Application Data Instance. An ADI is an application variable that is accessible for the updating of acyclic data or for the mapping of process data, exactly what is defined by the application.
ABCC	Anybus CompactCom
object model	See “Accessing the Anybus CompactCom” on page 29
object	
instance	
attribute	
telegram	See “Telegrams” on page 9
message	

1. How to use this Document

The purpose of this tutorial is to help the user to a better understanding of the Anybus CompactCom concept and how to configure the module, by discussing a simple example application.

When to Read and Use the Document

Whether you already have decided to develop an application with the Anybus CompactCom or not, you can read this tutorial as a first introduction on how to communicate with the module. It will walk you through examples to show you how to connect the module and how to set up the communication. It is recommended to perform this “in real life” to get a deeper understanding of how the module is functioning.

This tutorial does not cover any network specific issues. Each network interface appendix includes a short tutorial, that will help you prepare an application with the necessary elements for the certification of your product.

The Anybus CompactCom concept is further described in the Anybus CompactCom Software Design Guide (SWDG).

2. Introduction to the Anybus CompactCom

Background

Today there is no single standard for industrial communication. Several different industrial networks exist, using different protocols for exchanging information with devices connected to the network. The growing use and fragmentation of industrial Ethernet and fieldbus communication makes it even more difficult to decide which networks to support when you release a new product. To exchange one network or one device for another may be expensive and time consuming, not the least in development efforts and in extra equipment.

Anybus CompactCom

With Anybus CompactCom you have full flexibility. The different modules in the Anybus CompactCom family are designed to make it possible for a user to easily connect a device to any industrial network. The developing efforts when changing network is minimal i.e. an application developed for one network, is easy to move to another network.

Each module in the Anybus CompactCom family is dedicated to a specific industrial network. The module includes network specific firmware, that makes any data exchanged on the network available to the host application. The host application will thereby act as a device on the network through the module.

Regardless of network, the software interface that meets the host application and the developer is the same. The interface is structured in an object oriented manner. Each object is centred around a group of related information and services. This provides an efficient way of categorizing and addressing information.

The firmware in the Anybus CompactCom module typically controls the behavior of the module and its actions on the network. The host application is accessed from the Anybus CompactCom module, and the host application must be designed to handle these requests appropriately. This handling, though, is to a great extent independent of which industrial network the module is dedicated to. Ideally you should be able to exchange any module for any other in the Anybus CompactCom family, and the application should still be able to run without any changes to the host application firmware.

By using Anybus CompactCom you can save on your development costs, up to 70%. You will achieve maximum flexibility by getting access to all common fieldbuses and Ethernet networks. This gives you fast access to the market with very competitive network connectivity solutions.

3. Tutorial

3.1 Introduction

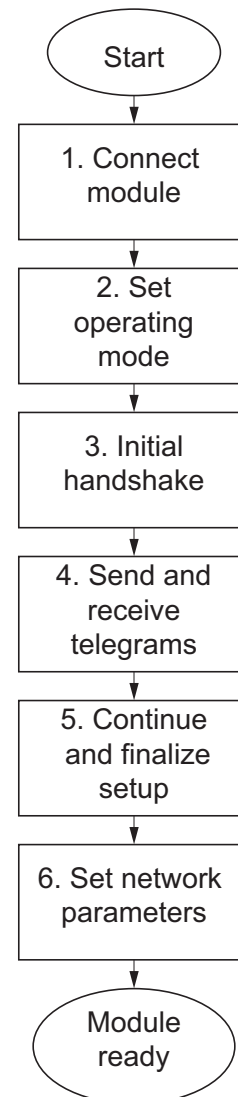
To implement an application, follow these steps:

1. Connect the Anybus CompactCom module, see “Connect the Anybus CompactCom Module” on page 6
2. Set operating mode (parallel or serial, baud rate), see “Set Operating Mode” on page 8
3. Verify initial handshaking functionality, see “Initial Handshake” on page 10 (parallel mode) or page 18 (serial mode).
4. Verify sending/receiving telegrams, see “Sending Telegrams - Parallel Communication” on page 13 or “Sending Telegrams - Serial Communication” on page 21
5. Finalize setup, see “Setup continued” on page 28.
6. Set necessary network specific parameters, see respective network interface appendix.

Step 1 - 5 are described below. The text presents examples and descriptions of functionality and characteristics of the Anybus CompactCom concept and modules.

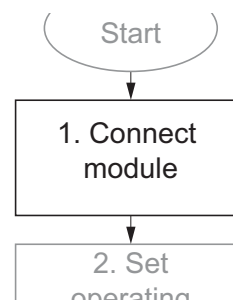
Step 6 is described in the respective Network Interface Appendices.

To illustrate the procedure, two example traces of communication have been included, see appendices “Trace, DeviceNet” on page 32 and “Trace, Profibus DP-V1” on page 39.



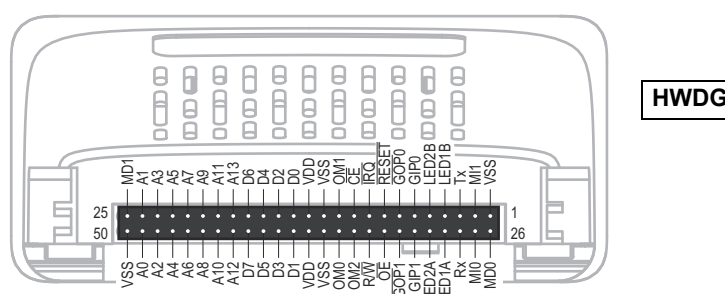
3.2 Connect the Anybus CompactCom Module

If you use the Anybus CompactCom Starter Kit, you can go straight to “Serial Interface Mode” on page 18. Please note that the starter kit does not allow use of the parallel mode.



3.2.1 Host Interface Signals

The Anybus CompactCom host interface uses a 50 pin CompactFlash™ style connector.



Note 1: The host interface is *not* pin compatible with the CompactFlash™ standard.

Note 2: The interface is not hot-swap capable. The power must be turned off before the module is attached or detached. Failure to observe this may damage the host product and/or the Anybus CompactCom module.

Each signal presented in the table below is described in detail in the Anybus CompactCom Hardware Design Guide.

Position	Signal	Type	Function	How to connect for the tutorial example
36, 11, 35	OM[0...2]	I	Operation Mode	See "Set Operating Mode" on page 8
27, 2	MI[0...1]	O	Module Identification	Leave unconnected
8	RESET	I	Reset Input, active low	Connect to a host application controllable output
26, 25	MD[0...1]	O	Module Detection	Leave unconnected
14, 39, 15, 40, 16, 41, 17, 42	D[0...7]	BI	Parallel Interface, active if set by the Operation Mode pins	Tie to VSS when unused
49, 24, 48, 23, 47, 22, 46, 21, 45, 20, 44	A[0...10]	I		Tie to VDD when unused
19, 43, 18	A[11...13]	I		Tie to VSS when unused
10	CE	I		Tie to VDD when unused
33	OE	I		Tie to VDD when unused
34	R/W	I		Tie to VDD when unused
9	IRQ	O		Leave unconnected if unused
28	Rx	I		Tie to VDD when unused
3	Tx	O	Serial Interface, active if set by the Operation Mode pins	Leave unconnected if unused
30	LED2A	O	Network Status LED Outputs	Leave unconnected
29	LED1A	O		
5	LED2B	O		
4	LED1B	O		
6, 31	GIP[0...1]	I	General Purpose I/O	Tie to VSS
7, 32	GOP[0...1]	O		Leave unconnected
13, 38	VDD	PWR	Power Supply	3.3 V
1, 12, 37, 50	VSS	PWR	Ground	Ground

I = Input, CMOS (3.3V)
O = Output, CMOS (3.3V)
BI = Bidirectional, Tristate
P = Power supply inputs

Note 1: For mechanical properties, measurements etc. see Appendix B "Mechanical Specification" in Hardware Design Guide.

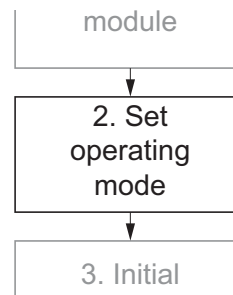
Note 2: Connect either the parallel interface OR the serial interface to the host application. The interface not used must be connected as indicated in the table.

Note 3: None of the host interface signals are 5V tolerant.

3.3 Set Operating Mode

Following the flowchart on page 5, choose either parallel mode or serial mode as shown in the table below.

The inputs OM2, OM1, and OM0 select which interface that should be used to exchange data (parallel or serial) and, if the serial interface option is used, the operating baud rate. The state of these signals is sampled once during startup, i.e. any change requires a reset in order to have effect.



SWDG

Operating Mode		Setting		
Parallel interface State	Serial interface State	OM2	OM1	OM0
Enabled	(disabled)	LOW	LOW	LOW
(disabled)	Enabled, baud rate: 19.2 kbps	LOW	LOW	HIGH
	Enabled, baud rate: 57.6 kbps	LOW	HIGH	LOW
	Enabled, baud rate: 115.2 kbps	LOW	HIGH	HIGH
	Enabled, baud rate: 625 kbps	HIGH	LOW	LOW

$LOW = V_{IL}$

$HIGH = V_{IH}$

Note 1: VDD, VSS and the state of the operation mode inputs must be stable prior to releasing the $\overline{RE-SET}$ signal. Failure to observe this may result in faulty or unexpected behavior.

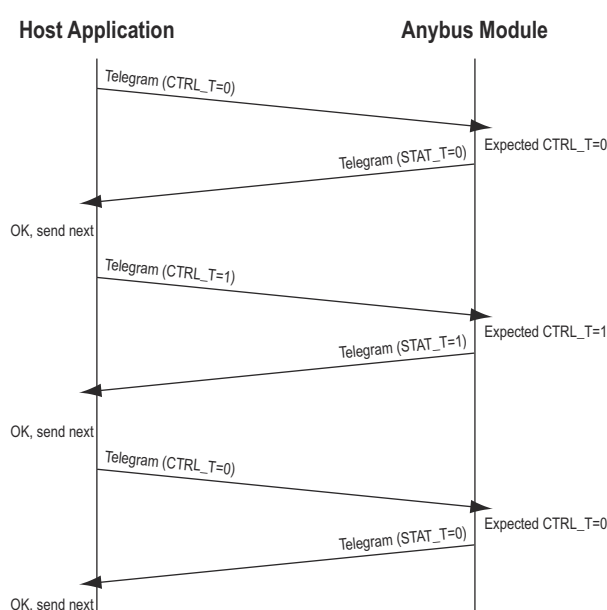
Note 2: If you use the Anybus CompactCom Starter Kit, which includes the Anybus CompactCom serial adapter, the parallel mode is not available.

3.4 Telegrams

The hardware setup of the module is finished and the rest of the tutorial will describe the setup of the communication between the host application and the Anybus CompactCom module and the configuration of the module.

Communication with the Anybus CompactCom module is handled through telegrams. These telegrams have a well defined format, where each telegram always include at least control or status information. They are sent in a ping-pong protocol, i.e. the application sends a telegram, including at least control information, to the module and waits for a telegram in return before it sends the next telegram. The return telegram always includes at least status information.

The control and status information is structured in registers, and dedicated bits, CTRL_T in the control register and STAT_T in the status register, are used to indicate need of retransmission and to make polling applications feasible.¹



All telegrams, whether in parallel or serial mode, have dedicated sections for control/status information and message data. Once the module is up and running, there is also a section for process data. Telegrams, registers and messages will be described further later in this chapter.

The user can define a maximum time for an expected response, yielding a time out exception if the response time is too long. For more information see “Anybus CompactCom Software Design Guide”, chapter three, sections “Anybus Watchdog” and “Application Watchdog”.

SWDG

1. See descriptions of these bits and registers on page 12 for parallel mode and on page 19 for serial mode.

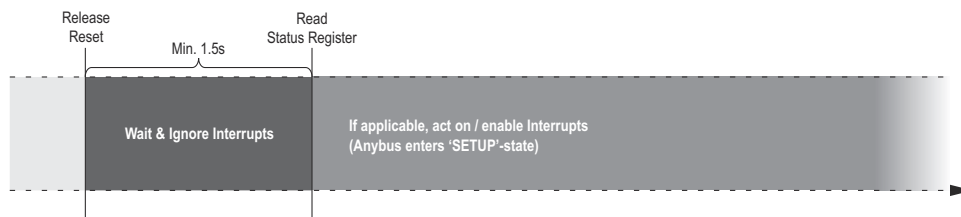
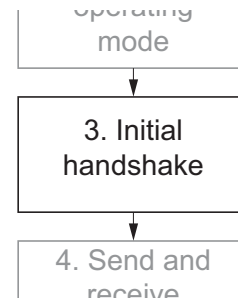
3.5 Parallel Interface Mode

For serial interface mode, please proceed to section 3.6 on page 18.

3.5.1 Initial Handshake

The purpose of the initial handshake is to make sure that the Anybus CompactCom module is ready to communicate. When done, the module will enter the SETUP-state.

The internal status register, see page 12, is cleared at startup, which is essential for a correct setup. The host application must wait at least 1.5 s after reset or power up, to be sure that a correct reading of the status register can be done. Any interrupts detected during this period must be ignored. After this initial handshaking the module has entered SETUP state and can be configured. The host application can either react on interrupts or poll the internal Status Register for an indication that information is available.



3.5.2 DPRAM

SWDG

After the initial handshake has been performed, the host application can start to communicate with the Anybus CompactCom module with telegrams. This section gives a short introduction to the means of communication used.

The communication between the application and Anybus CompactCom module in parallel mode is performed in the internal DPRAM (dual port memory). The host application can access and address this memory just as any memory in an embedded system. When a telegram is sent from the host application, the contents are written by the host application to specific areas in the DPRAM. When the host application receives a telegram, specific areas of the memory hold the contents of the telegram for the host application to read.

The DPRAM memory includes areas for the Control and Status Registers (used to control the communication between the host application and the module), read and write areas for Messages, and read and write areas for Process Data. The addresses for the different areas are fixed. A telegram sent by the host application to the Anybus CompactCom always contain data to be written to the Control Register in the DPRAM. A telegram sent the other way, from the Anybus CompactCom to the host application, always replicates the Status Register in the DPRAM. Depending on the contents of these registers, the telegram may contain Messages and/or Process Data. The Process Data Read and Write Areas are not used during setup and configuration.

Memory Map

The address offset specified below is relative to the base address of the module, i.e. from the beginning of the area in the host application memory space where the parallel interface has been implemented.

Address Offset:	Area:	Access:	Notes:
0000h... 37FFh	(reserved)	-	(reserved for future use)
3800h... 38FFh	Process Data Write Area	Write Only	Not used during setup and configuration. See "Process Data Subfield" in SWDG
3900h... 39FFh	Process Data Read Area	Read Only	Not used during setup and configuration. See "Process Data Subfield" in SWDG
3A00h... 3AFFh	(reserved)	-	
3B00h... 3C06h	Message Write Area	Write Only	Used for messages to the module from the host application. See "Object Messaging" in SWDG
3C07h... 3CFFh	(reserved)	-	
3D00h... 3E06h	Message Read Area	Read Only	Used for messages from the module to the host application. See "Object Messaging" in SWDG
3E07h... 3FFDh	(reserved)	-	
3FFEh	Control Register	Write Only	See "Control Register" on page 12
3FFFh	Status Register	Read Only	See "Status Register" on page 12

IMPORTANT: *C-programmers are reminded to declare the entire shared memory area as volatile.*

Control Register (Read/Write)

The Control Register controls the communication with the Anybus CompactCom module. In this tutorial only bits 5 (CTRL_R), 6 (CTRL_M) and 7 (CTRL_T) are used. The other bits can be ignored for the time being. For a complete description, see Anybus CompactCom Software Design Guide.

SWDG

b7 (MSB)	b6	b5	b4	b3	b2	b1	b0 (LSB)
CTRL_T	CTRL_M	CTRL_R	CTRL_AUX	-	-	-	-

Bit	Description
CTRL_T	The host application shall toggle this bit when sending a new telegram. CTRL_T must be set to "1" in the initial telegram sent by the application to the module.
CTRL_M	Set if the telegram contains message data.
CTRL_R	If set, the host application is ready to receive a new command.
CTRL_AUX	Auxiliary bit
-	(reserved, set to zero)

Status Register (Read Only)

This register holds the current status of the Anybus CompactCom module. At this stage in the tutorial we will only use bits 5 (STAT_R), bit 6 (STAT_M) and bit 7 (STAT_T). The remaining bits can be ignored for the time being. For a complete description see Anybus CompactCom Software Design Guide.

SWDG

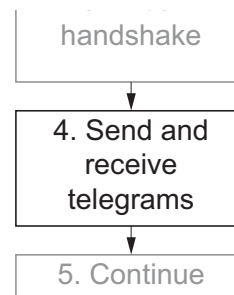
b7 (MSB)	b6	b5	b4	b3	b2	b1	b0 (LSB)
STAT_T	STAT_M	STAT_R	STAT_AUX	SUP	S2	S1	S0

Bit	Description																																				
STAT_T	When the Anybus CompactCom module issues new telegrams, this bit will be set to the same value as CTRL_T in the last telegram received from the host application.																																				
STAT_M	Set if the telegram contains message data.																																				
STAT_R	If set, the Anybus CompactCom module is ready to receive a new command.																																				
STAT_AUX	Auxiliary bit																																				
SUP	<u>Value:Meaning:^a</u> 0: Anybus CompactCom module is not supervised. 1: Anybus CompactCom module is supervised by another network device																																				
S _[0... 2]	These bits indicates the current state of the Anybus CompactCom module																																				
	<table><tr><th>S2</th><th>S1</th><th>S0</th><th>Anybus CompactCom State</th></tr><tr><td>0</td><td>0</td><td>0</td><td>SETUP</td></tr><tr><td>0</td><td>0</td><td>1</td><td>NW_INIT</td></tr><tr><td>0</td><td>1</td><td>0</td><td>WAIT_PROCESS</td></tr><tr><td>0</td><td>1</td><td>1</td><td>IDLE</td></tr><tr><td>1</td><td>0</td><td>0</td><td>PROCESS_ACTIVE</td></tr><tr><td>1</td><td>0</td><td>1</td><td>ERROR</td></tr><tr><td>1</td><td>1</td><td>0</td><td>(reserved)</td></tr><tr><td>1</td><td>1</td><td>1</td><td>EXCEPTION</td></tr></table>	S2	S1	S0	Anybus CompactCom State	0	0	0	SETUP	0	0	1	NW_INIT	0	1	0	WAIT_PROCESS	0	1	1	IDLE	1	0	0	PROCESS_ACTIVE	1	0	1	ERROR	1	1	0	(reserved)	1	1	1	EXCEPTION
	S2	S1	S0	Anybus CompactCom State																																	
	0	0	0	SETUP																																	
	0	0	1	NW_INIT																																	
	0	1	0	WAIT_PROCESS																																	
	0	1	1	IDLE																																	
	1	0	0	PROCESS_ACTIVE																																	
	1	0	1	ERROR																																	
1	1	0	(reserved)																																		
1	1	1	EXCEPTION																																		

a. This bit is not used in all networks. Please consult the appropriate appendix for explanation

3.5.3 Sending Telegrams - Parallel Communication

The examples in this section show what the host application writes to and reads from the different areas in the DPRAM, illustrating the steps when the application sends and receives telegrams. For example, “W Control Reg 0x80” means that the host application writes the hexadecimal value 80 to the control register area of the DPRAM. Bit 7 of the register is set to 1, the rest of the register is all zeroes.



Sending Telegrams (Parallel)

Once the initial handshaking is done, see page 10, the host application can start to send telegrams to the module. That is, the host applications starts writing to the control register area in the DPRAM.

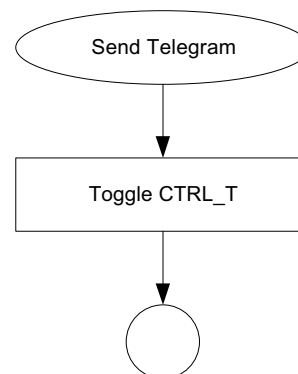
The example shows the underlying telegram ping-pong protocol. Each telegram to the module from the host application generates a response telegram from the module to the host application.

Example:

```

W Control Reg 0x80
R Status Reg 0x80
W Control Reg 0x00
R Status Reg 0x00
W Control Reg 0x80
R Status Reg 0x80
W Control Reg 0x00
R Status Reg 0x00
etc.
  
```

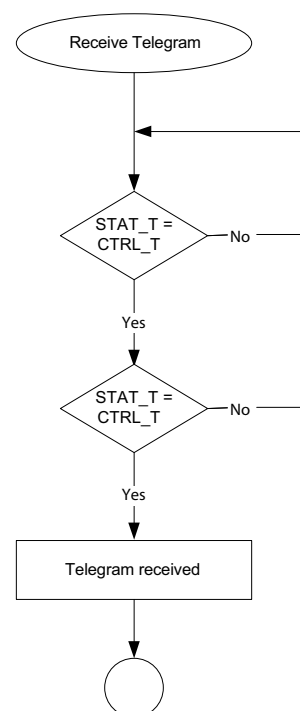
During parallel communication, the host application sends a telegram by toggling CTRL_T in the Control Register in the DPRAM. When this register is written an interrupt is generated in the Anybus CompactCom module, and the module checks the CTRL_T bit to see if a new telegram is available



The Anybus CompactCom module checks its status, and when it is satisfied that it has performed all actions possible at the time being, it toggles the STAT_T bit in Status register to equal CTRL_T. A valid response telegram is available for the host application.

The host application keeps checking the STAT_T bit. When this bit equals CTRL_T at two consecutive readings, it has received a response telegram. Two consecutive readings are performed to be sure that the value of the STAT_T bit is stable.

Telegrams are sent and received like this, in a ping-pong protocol, as long as the Anybus CompactCom module is communicating normally.



The continuous exchange of telegrams is established. The host application waits for a signal to send a message to the module.

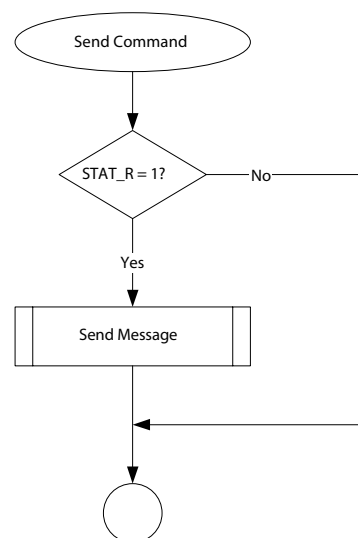
Example:

```

W Control Reg 0x80
R Status Reg 0x80
W Control Reg 0x00
R Status Reg 0x00
W Control Reg 0x80
R Status Reg 0xa0
  
```

In the last line of the example above, the status register has not only had its STAT_T bit toggled to equal CTRL_T, but also the STAT_R bit set to 1. This indicates that the module is ready to accept a command from the application. In the next telegram the application sends, it can add message data to the telegram. It is not necessary or mandatory to send a message at this stage, but if you do, it has to be a command.

Please note that the number of telegrams exchanged, before the module is ready to accept a message, can vary.



Sending a Message to the Anybus CompactCom Module (Parallel)

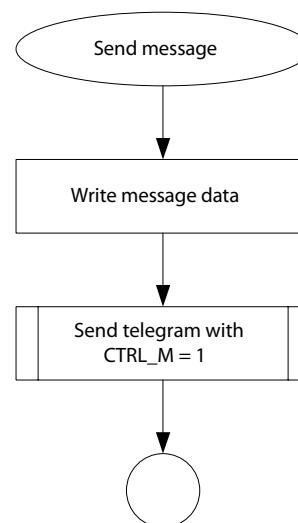
To check that the communication is working, a message including a command can be sent to the module. For example, a message is sent to the module to return its module type, as in this case.

Example:

```
W Control Reg 0x80
R Status Reg 0xa0
W Message Write Area 0x01, 0x01, 0x01,
0x00, 0x41, 0x00, 0x01, 0x00
W Control Reg 0x40
R Status Reg 0x60
R Message Read Area 0x01, 0x01, 0x01,
0x00, 0x01, 0x02, 0x01, 0x00, 0x01, 0x04
```

Note: Please refer to the table on page 16 for an explanation of the format and the contents of the message write and read areas in the DPRAM.

The application writes a message to the message write area in the DPRAM, see the memory map on page 11. Once this is done, it writes to the control register, toggling the CTRL_T bit AND setting the CTRL_M bit simultaneously. Each time anything is written to this register, an interrupt is generated in the Anybus module, so it is important that all bits in the register are written at the same time. It is equally important that the application has finished writing the message write area prior to writing to the Control Register.



Receiving a Message from the Anybus CompactCom Module (Parallel)

The module reads the telegram the host application has sent, and returns the module type.

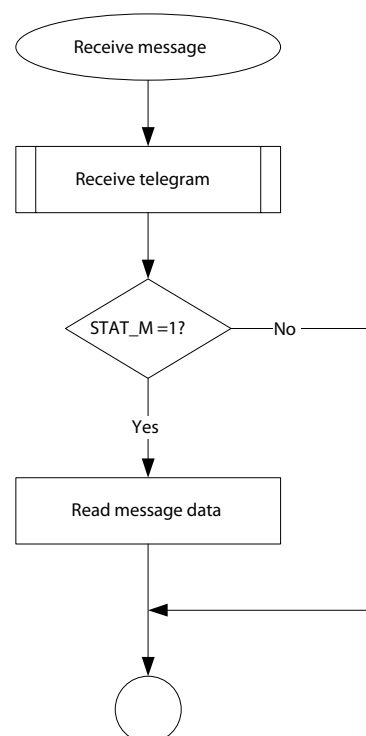
Example (repeated):

```
W Message Write Area 0x01, 0x01, 0x01,
0x00, 0x41, 0x00, 0x01, 0x00
W Control Reg 0x40
R Status Reg 0x60
R Message Read Area 0x01, 0x01, 0x01,
0x00, 0x01, 0x02, 0x01, 0x00, 0x01, 0x04
```

When the application has finished sending its telegram by writing to the control register, an interrupt is generated and the module detects the arrival of the telegram including the message. It reads the message write area in the DPRAM and processes the command. (For a description of the contents of the message write and read areas, see the table on page 16).

The module processes the data and then writes its response to the message read area. This done, it writes to the status register, toggling the STAT_T bit to equal CTRL_T bit and setting STAT_M to one to indicate that a message is available in the message read area.

The application detects that STAT_T has changed and that STAT_M is set to 1. It will then have to fetch the message in the DPRAM message read area, before the next telegram is sent.



Please note that after writing to the control register, the application is not allowed to read the contents of the DPRAM, except for the status register, until the Anybus CompactCom module has finished updating the status register. This can be detected by the application either by polling the status register until $STAT_T^1$ equals $CTRL_T$ or by using an interrupt signal (\overline{IRQ}) from the Anybus CompactCom module.

The contents of the messages in the telegrams are explained in the tables below. For a complete description of the message layout, please refer to the Software Design Guide, chapter 5

SWDG

Command:

Area offset	Contents ^a of request in Write Message Area	Description (request) ^b
0	00h	Source ID
1	01h	The Anybus object in the Anybus CompactCom module.
2	01h (lsb)	Instance 1
3	00h (msb)	
4	41h	Message type: command (request to return module type)
5	00h	Size of Message Data (0 bytes)
6	01h	Attribute 1 (Module type)
7	00h	(not used)

a. All data is little endian.

b. Object messaging is used when addressing the objects in the Anybus CompactCom module and in the host application. See "Accessing the Anybus CompactCom" on page 29 for more information.

Response:

Area offset	Contents ^a of response in Read Message Area	Description (response)
0	00h	Source ID ^b
1	01h	The Anybus object in the Anybus CompactCom module.
2	01h (lsb)	Instance 1
3	00h (msb)	
4	01h	Message type: response (the command is copied, with the C bit set to 0)
5	02h	Size of Message Data (2 bytes)
6	01h	Attribute 1 (Module type)
7	00h	(not used)
8-9	01h, 04h	Message data: module type (0401h = Anybus CompactCom)

a. All data is little endian.

b. To keep track of which response belongs to which request, each message is tagged with a Source ID. When issuing commands, the host application may choose Source ID arbitrarily, and the response from the module will have the same Source ID. When sending a response to a command from the module, the host application should always copy the Source ID, the object number and the instance number of the message with the original command. The command is also copied, but the C bit is set to 0.

Note: As stated above, telegram transmission is triggered via the Control Register, and the reception of a new telegram is indicated in the Status Register. This means that the Process Data and Message sub-fields must be written *prior* to accessing the Control Register. Do not use bit handling or other read-modify-write instructions directly on this register, since the module may interpret this as multiple accesses. All bit handling etc. must instead be performed in a temporary register, and after manipulation be written back. Also, while waiting for reception of a telegram, any access to the parallel interface other than polling of the Status Register must be avoided.

1. To ensure valid results when polling the Status Register, it is required to use a read-until-two-consecutive-readings-agree procedure, i.e. the same value is read at two consecutive readings.

Parallel Telegram Handling, Summary

On the parallel interface, transmission and reception is managed through the Handshake Registers, i.e. the Status Register and the Control Register.

To transmit a telegram, perform the following steps:

1. If applicable, write the Write Process Data to the Process Data Write Area (3800h...38FFh)
2. If applicable, write the message to the Message Write Area (3B00h...3C06h)
3. Update the Control Register to trigger the transmission.

Telegram reception is signalled by the STAT_T-bit in the Status Register. The host application can either poll the Status Register cyclically to detect new telegrams, or rely on interrupt operation.

IMPORTANT:

- To ensure valid results when polling the Status Register, it is required to use a read-until-two-consecutive-readings-agree procedure, i.e. the same value is read at two consecutive readings.
- The host application must ensure continuous ping-pong communication, so that the ABCC module will be able to send commands as well.
- One command always results in one and only one response.
- Message responses can always be sent, regardless of the state of the CTRL_R-bit or the STAT_R-bit.
- Several telegrams can occur before a response is sent.
- Several commands can be waiting for a response, in both directions. As long as the CTRL_R-bit or the STAT_R-bit is set, new commands are allowed in the respective direction.
- If several commands have been sent without waiting for responses, the order of responses may be different than the order of commands.

Please continue to “Setup continued” on page 28.

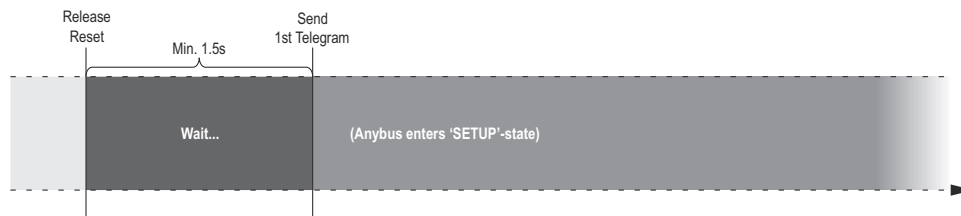
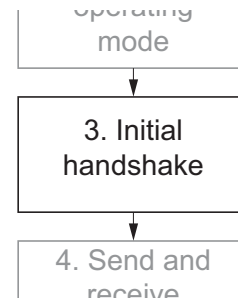
3.6 Serial Interface Mode

In this section, the same steps are described as in section 3.5, but for the serial interface mode.

3.6.1 Initial Handshake

The purpose of the initial handshake is to make sure that the Anybus CompactCom module is ready to communicate. When done, the module will enter the SETUP-state.

The internal status register, see page 20, is cleared at startup, which is essential for a correct setup. The host application must thus wait at least 1.5 s after reset or power up before it sends the first telegram to be sure that the module is ready to receive serial data.



3.6.2 Serial Telegram Frame

The telegrams sent in serial mode all use the frame shown below. During SETUP, the Process Data Subfield is not present in the telegrams, leaving 19 bytes to be sent in each telegram.

1 byte	16 bytes	Up to 256 bytes	2 bytes
Handshake Register Field	Message Subfield	Process Data Subfield, not existing during SETUP	CRC16
(1st byte)	Please consult the Software Design Guide for a thorough description of this field.		(last byte)

- Handshake Register Field**

This field contains the Control Register, see page 19, in telegrams sent *to* the Anybus CompactCom module, and the Status Register, see page 20, in telegrams received *from* the Anybus CompactCom module.

- Message Subfield**

This field holds the message (or message fragment) that is to be sent. To maintain throughput for the Process Data, the message subfield is limited to 16 bytes when using the serial interface. Longer messages are exchanged as several smaller fragments (see “Fragmentation” in the Software Design Guide).

SWDG

- Process Data Subfield**

This field contains the Write Process Data in telegrams sent *to* the Anybus CompactCom module, and the Read Process Data in telegrams received *from* the Anybus CompactCom module. Please note that this field does not exist when the Anybus CompactCom module is operating in the ‘SETUP’-state.

- CRC16**

This field holds a 16 bit Cyclic Redundancy Check, see “CRC Calculation” in the Software Design Guide. The CRC covers the entire telegram except for the CRC itself.

SWDG

Control Register (Sent)

The Control Register controls the communication with the Anybus CompactCom module. In this tutorial we will only use bits 5 (CTRL_R), 6 (CTRL_M) and 7 (CTRL_T). The other bits can be ignored for the time being. For a complete description, see Anybus CompactCom Software Design Guide.

SWDG

b7 (MSB)	b6	b5	b4	b3	b2	b1	b0 (LSB)
CTRL_T	CTRL_M	CTRL_R	CTRL_AUX	-	-	-	-
Bit	Description						
CTRL_T	The host application shall toggle this bit when sending a new telegram. CTRL_T must be set to “1” in the initial telegram sent by the application to the module.						
CTRL_M	Set if the telegram contains message data.						
CTRL_R	If set, the host application is ready to receive a new command.						
CTRL_AUX	Auxiliary bit						
-	(reserved, set to zero)						

Status Register (Received)

This register holds the current status of the Anybus CompactCom module. At this stage in the tutorial we will only use bits 5 (STAT_R), bit 6 (STAT_M) and bit 7 (STAT_T). The remaining bits can be ignored for the time being. For a complete description see Anybus CompactCom Software Design Guide.

SWDG

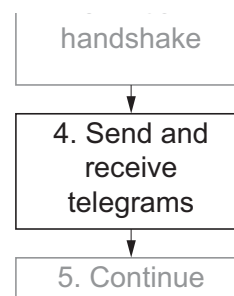
b7 (MSB)	b6	b5	b4	b3	b2	b1	b0 (LSB)
STAT_T	STAT_M	STAT_R	STAT_AUX	SUP	S2	S1	S0
Bit	Description						
STAT_T	When the Anybus CompactCom module issues new telegrams, this bit will be set to the same value as CTRL_T in the last telegram received from the host application.						
STAT_M	Set if the telegram contains message data.						
STAT_R	If set, the Anybus CompactCom module is ready to receive a new command.						
STAT_AUX	Auxiliary bit						
SUP	<u>Value: Meaning:^a</u> 0: Anybus CompactCom module is not supervised. 1: Anybus CompactCom module is supervised by another network device						
S _[0...2]	These bits indicates the current state of the Anybus CompactCom module						
	S2	S1	S0	Anybus CompactCom State			
	0	0	0	SETUP			
	0	0	1	NW_INIT			
	0	1	0	WAIT_PROCESS			
	0	1	1	IDLE			
	1	0	0	PROCESS_ACTIVE			
	1	0	1	ERROR			
	1	1	0	(reserved)			
	1	1	1	EXCEPTION			

a. This bit is not used in all networks. Please consult the appropriate appendix for explanation

3.6.3 Sending Telegrams - Serial Communication

The examples in this section show what is sent from the host application to the Anybus CompactCom module and what is received by the application from the module, illustrating the steps when using serial communication.

All telegrams sent in serial mode contains 19 bytes, even when not all fields are needed. The fields not used, are filled with zeroes. The examples thus show the total content of each telegram.



Sending Telegrams (Serial)

Once the initial handshaking is done, see page 18, the host application can start to send telegrams to the module.

Example:

```
Send:  {0x80, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, bCrccHi, bCrccLo}
```

```
Receive: {0x80, 0x00, 0x00, 0x00,
          0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
          0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
          0x00, bCrccHi, bCrccLo}
```

```
Send:  {0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, bCrccHi, bCrccLo}
```

```
Receive: {0x00, 0x00, 0x00, 0x00,
          0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
          0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
          0x00, bCrccHi, bCrccLo}
```

```
Send:  {0x80, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, bCrccHi, bCrccLo}
```

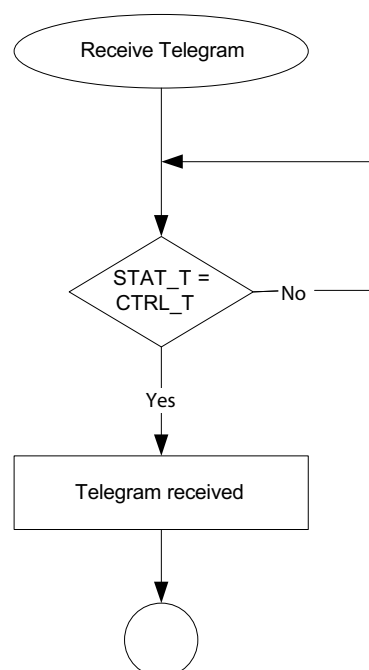
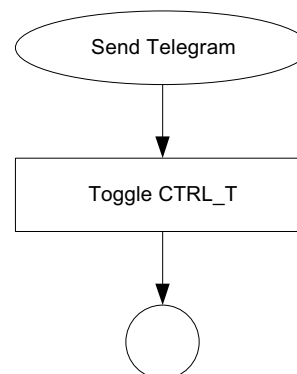
```
Receive: {0x80, 0x00, 0x00, 0x00,
          0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
          0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
          0x00, bCrccHi, bCrccLo}
```

etc.

The application sends a telegram with the CTRL_T bit in the Control Register set to 1. The module sends a telegram in return, toggling the STAT_T bit to equal CTRL_T. No data has been sent yet, the rest of the telegram frames are empty, apart from the CRC (bCrccHi, bCrccLo) which ends each frame.

Even though there is no information in most of the telegram frame, a complete frame has to be sent each time. The parts not used are filled with zeroes.

Telegrams are sent and received like this, in a ping-pong protocol, as long as the Anybus CompactCom module is communicating normally.



The continuous exchange of telegrams is established. The host application waits for a signal to send a message to the module.

Example:

```
Send: {0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
       0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
       0x00, 0x00, 0x00, 0x00, bCrcHi, bCrcLo}
```

```
Receive: {0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
          0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
          0x00, 0x00, 0x00, 0x00, 0x00, bCrcHi,
          bCrcLo}
```

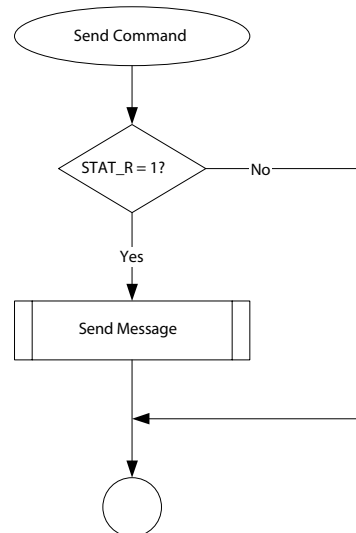
```
Send: {0x80, 0x00, 0x00, 0x00, 0x00, 0x00,
       0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
       0x00, 0x00, 0x00, 0x00, bCrcHi, bCrcLo}
```

```
Receive: {0xa0, 0x00, 0x00, 0x00, 0x00, 0x00,
          0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
          0x00, 0x00, 0x00, 0x00, 0x00, bCrcHi,
          bCrcLo}
```

In the last telegram in the example above, the status register has not only had its STAT_T bit toggled to equal CTRL_T, but also the STAT_R bit set to 1. This indicates that the module is ready to accept a command from the application.

In the next telegram the application sends, it can add message data to the telegram. It is not necessary or mandatory to send a message at this stage, but if you do, it has to be a command.

Please note that the number of telegrams exchanged, before the module is ready to accept a message, can vary.



Sending a Message to the Anybus CompactCom Module (Serial)

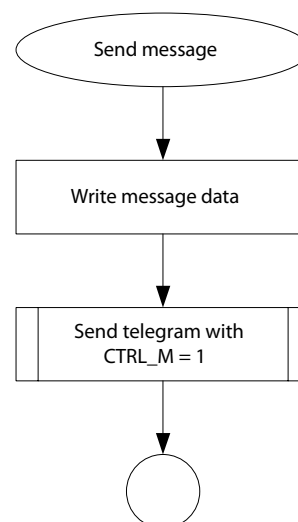
To check that the communication is working, a telegram with a message including a command can be sent from the host application to the module. For example, a message can be sent to the module to return its module type, as in this case.

Example:

```
Send: {0x40, 0x00, 0x00, 0x01, 0x01, 0x00,
      0x41, 0x00, 0x01, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, bCrccHi, bCrccLo}
```

```
Receive: {0x20, 0x00, 0x00, 0x00, 0x00,
          0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
          0x00, 0x00, 0x00, 0x00, bCrccHi,
          bCrccLo}
```

The application sends the command in the message data subfield in the telegram frame, see the table below. The module acknowledges the telegram, but doesn't take any action yet, as it doesn't know if all data in the message has arrived.



Description of the contents of the first telegram sent:

Byte no	From application to module ^a	Description (to module) ^b
1	40h	Control Register, CTRL_M = 1
2	00h	Source ID
3	01h	The Anybus object in the Anybus CompactCom module
4	01h (lsb)	Instance 1
5	00h (msb)	
6	41h	Message type: command (request to return module type)
7	00h	Size of Message Data (0 bytes)
8	01h	Attribute 1 (Module type)
9	00h	(not used)
10-17	00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h	(no message data)
18-19	bCrccHi, bCrccLo	CRC

a. All data is little endian.

b. Object messaging is used when addressing the objects in the Anybus CompactCom module and in the host application. See "Accessing the Anybus CompactCom" on page 29 for more information.

The layout of the message subfield, bytes 2 - 17, is described in detail in SWDG, chapter 5.

The telegram the application receives from the module in this example is empty, apart from the status register in the first byte (20h), and the CRC in the two last bytes. The STAT_T bit is toggled to the same value as CTRL_T. STAT_R is set, to indicate that a new command can be accepted. As a telegram frame always has 19 bytes, the empty message subfield is filled with zeroes.

SWDG

Receiving a Message (Serial)

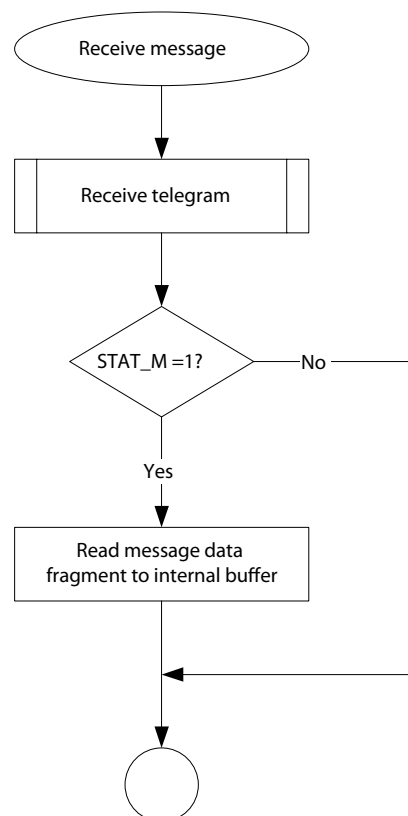
The host application has sent all message data needed to the Anybus CompactCom module, but it has to tell the module that so is the case. This done, the module sends a telegram that includes the response to the command sent by the host application.

Example:

```
Send:  {0x80, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        bCrccHi, bCrccLo}

Receive: {0xE0, 0x00, 0x01, 0x01, 0x00,
          0x01, 0x02, 0x01, 0x00, 0x01, 0x04,
          0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
          bCrccHi, bCrccLo}
```

The telegram, sent by the application, contains no message data. This tells the module that all message data has arrived, and it can start to process the previously arrived command. In this example, the response from the module is included in the next telegram, that the application receives from the module, see table below. This may not necessarily be the case. Several telegrams can be exchanged before the module has finished processing the command and sends the response.



Description of the contents of the second telegram received by the application:

Byte no	From module to application ^a	Description (to module/from module)
1	E0h	Status Register
2	00h	Source ID ^b
3	01h	The Anybus object in the Anybus CompactCom module/ Instance 1
4	01h (lsb)	
5	00h (msb)	
6	01h	Message type: response (the command is copied, with the C bit set to 0)
7	02h	Size of Message Data (0/2 bytes)
8	01h	attribute 1 (Module type)
9	00h	(not used)
10-11	01h, 04h	Message data: module type (0401h = Anybus CompactCom)
12-17	00h, 00h, 00h, 00h, 00h, 00h	(not used)
18-19	bCrccHi, bCrccLo	CRC

a. All data is little endian.

b. To keep track of which response belongs to which request, each message is tagged with a Source ID. When issuing commands, the host application may choose Source ID arbitrarily, and the response from the module will have the same Source ID. When sending a response to a command from the module, the host application should always copy the Source ID, the object number and the instance number of the message with the original command. The command is also copied, but the C bit is set to 0.

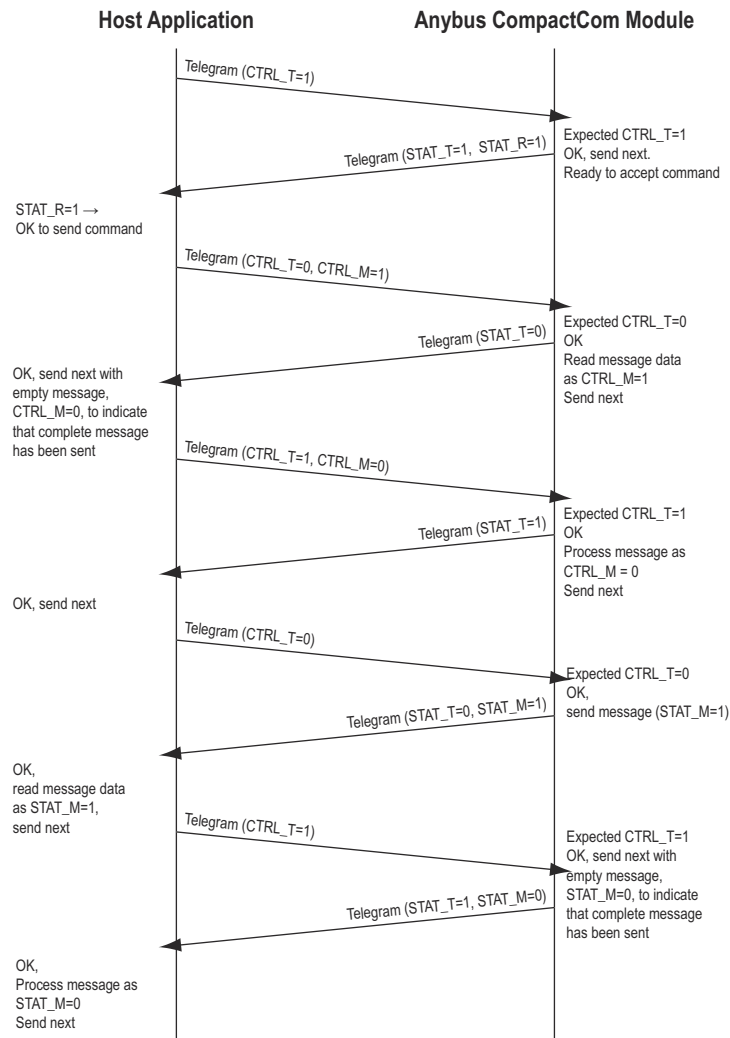
The layout of a message is described in detail in SWDG, chapter 5.

SWDG

Number of Telegrams in Serial Communication

In serial communication, a telegram with CTRL_M = 0 (from the application) or STAT_M = 0 (from the Anybus CompactCom module) signals that all data belonging to the message has been sent. This implies that any message needs at least two telegrams to be delivered to the module, as well as the other way around.

The figure below show an example of a sequence of telegrams between the host application and the Anybus CompactCom module. The application sends a telegram to check that the module is ready to accept a command. The module returns a telegram where STAT_R is set to 1, indicating that it is ready to accept a command. Then the application can send a telegram with a message including a command. The Anybus CompactCom module receives this telegram but does not process the message until it has received a telegram with an empty message field to ensure that all of the message has been sent. The Anybus CompactCom processes the message and sends a response back, also finished by a telegram including an empty message.



Each arrow in this example of ping-pong communication, indicates a telegram sent either from the application to the Anybus CompactCom module, or the other way around.

Serial Message Handling, Summary

On the serial interface, transmission and reception of messages are managed through the Handshake Registers, which contents always are available in the first byte of the telegram.

To transmit a message, perform the following steps:

1. Send a telegram to the Anybus CompactCom module with a message in the message subfield of the telegram frame and set CTRL_M = 1.
2. Wait for a telegram from the module where STAT_T equals CTRL_T
3. If the message is larger than the defined fields in the telegram frame, the message will have to be fragmented, see “Fragmentation” in SWDG. Continue sending telegrams with message data and CTRL_M set until all message data has been sent.
4. Send a telegram with CTRL_M = 0 to the Anybus CompactCom module to indicate that the transmission of the message is finished.

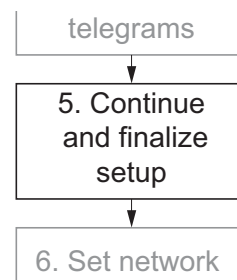
SWDG

IMPORTANT:

- The host application must ensure continuous ping-pong communication, so that the ABCC module will be able to send commands as well.
- One command always results in one and only one response.
- Message responses can always be sent, regardless of the state of the CTRL_R-bit or the STAT_R-bit.
- Several telegrams can occur before a response is sent.
- Several commands can be waiting for a response, in both directions. As long as the CTRL_R-bit or the STAT_R-bit is set, new commands are allowed in the respective direction.
- If several commands have been sent without waiting for responses, the order of responses may be different than the order of commands.

3.7 Setup continued

In the examples in the rest of this tutorial, only the contents of the Control and Status Registers, and the Message Read and Write Areas will be presented. If serial or parallel communication is used is not relevant at this level of communication as the telegrams only act as means of transport for messages and data according to the protocols presented in the previous sections.

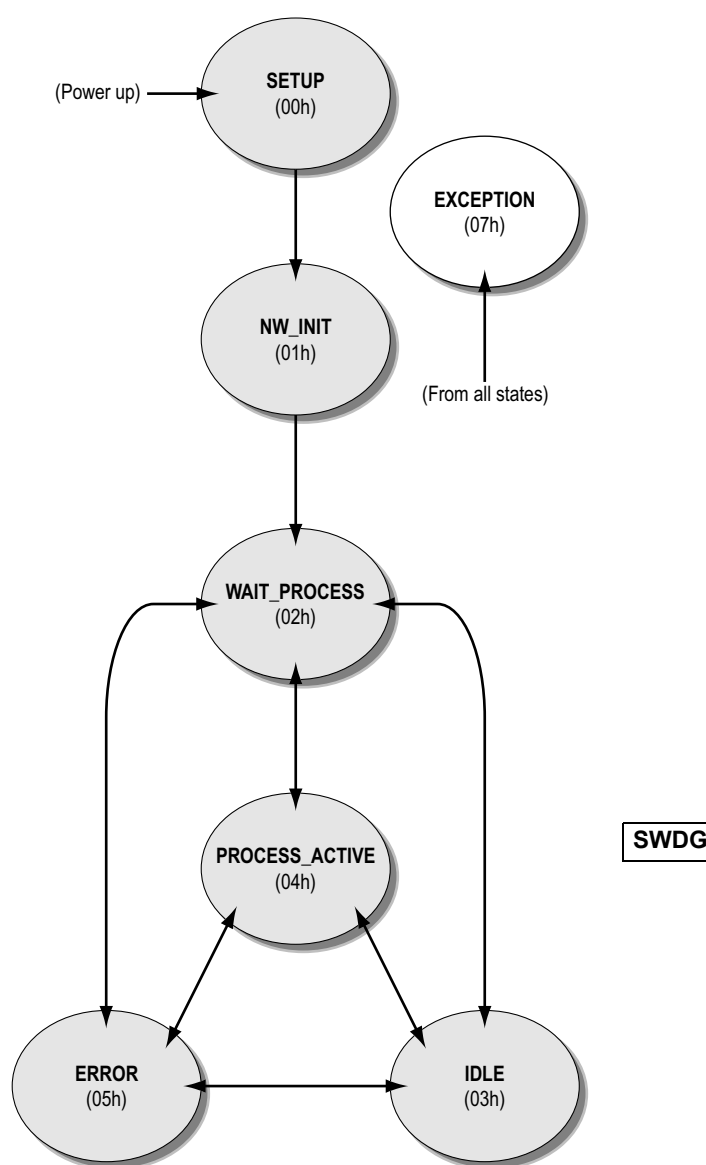


3.7.1 Anybus CompactCom State Machine

After the initial handshake the Anybus CompactCom enters the SETUP state.

The host application can send telegrams to the module, and read/receive the responses. All communication to and from the module is defined in telegrams that act as means of transport for messages and data.¹ In this section the continued setup of the Anybus CompactCom will be described, to show how to make the module ready for transmitting and receiving process data.

After setup has been finished the module will enter the NW_INIT state, where it performs network related initialization tasks. These tasks differs depending on the network interface used. When this is done, the module enters the WAIT_PROCESS state, and is ready to take part in the communication of the selected network interface. The state of the CompactCom module can always be decided by reading bits 0-2 in the status register, see “Status Register (Read Only)” on page 12, but the transitions between the states may be so fast, that you will not be able to detect all of them.



1. No process data can be sent during the SETUP state, only message data.

3.7.2 Accessing the Anybus CompactCom

The first part of a message sent to the module, contains information to identify where the command/request is to be sent within the module firmware.

Information and services in the module are grouped into objects, each holding information and services that belong to or are related to each other. This provides an easy and efficient way of accessing the module for configuration, Object Messaging. Each message is tagged with an object number and an instance number, specifying the location of the data or setting associated with the message.

Requests from the module are of the same format, expecting the application to be designed in the same manner, implementing objects that are similar in structure to the objects in the Anybus CompactCom Firmware.

The host application must be able to handle all requests arriving from the module, even when the object the request is asking for does not exist. If the module sends a message, that addresses an object that is not implemented in the application, the application should respond with an error message. For further information see “The Object Model” in the Anybus CompactCom Software Design Guide

SWDG

There is only one object that is mandatory for the application to implement, the Application Data object, that is used to exchange data through so called ADIs (Application Data Instances, see below). In most applications it is recommended to implement more objects, as they may be needed for certification. Please refer to the Network Interface Appendices.

3.7.3 Mapping ADIs

Data is exchanged on the network through ADIs (Application Data Instances) in the Application Data Object, residing in the host application. Each ADI represents a block of network data. An ADI is normally associated with acyclic parameters on the network side. It may also be mapped as Process Data. Process data is exchanged through a dedicated data channel in the Anybus CompactCom host protocol, and is normally associated with fast cyclical network I/O.

Each ADI can be tagged with a name, data type, range and default value, which makes it easily accessible from the network.

The exact representation of ADIs is highly network specific, so in this example we only show a simple, comparatively general ADI mapping. First two bytes (data type UINT16), and then one byte (data type UINT8) are mapped to the Process Read Area. Please note, that this is only the setup of the area. The area itself will not be accessible until setup is finished. Only the content of the message will be shown in the example

Example:

First message to module:
{0x02, 0x03, 0x01, 0x00, 0x51, 0x04, 0x01, 0x00, 0x05, 0x01, 0x01, 0x00}

The module responds with:
{0x02, 0x03, 0x01, 0x00, 0x11, 0x01, 0x01, 0x00, 0x00}

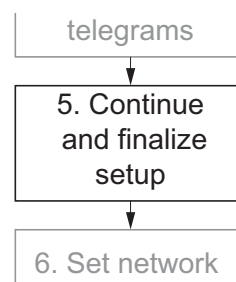
Second message to module:
{0x03, 0x03, 0x01, 0x00, 0x51, 0x04, 0x02, 0x00, 0x04, 0x01, 0x02, 0x00}

The module responds with:
{0x03, 0x03, 0x01, 0x00, 0x11, 0x01, 0x02, 0x00, 0x02}

For explanation of the example, please see tables below and on next page

.First command:

Area offset	Contents of request	Description (request)
0	02h	Source ID
1	03h	The Network Object in the Anybus CompactCom module.
2 - 3	01h 00h	Instance 1
4	51h	Message type: command (Map_ADI_Read_Area)
5	04h	Size of Message Data (4)
6 - 7	01h 00h	ADI instance number (1)
8	05h	Data type UINT16
9	01h	Number of elements in ADI (1)
10 - 11	01h 00h	Order number of the ADI (1)



First response:

Area offset	Contents of response	Description (response)
0	02h	Source ID
1	03h	The Network Object in the Anybus CompactCom module.
2 - 3	01h 00h	Instance 1
4	11h	Message type: response
5	01h	Size of Message Data (1 byte)
6 - 7	01h 00h	ADI instance number (1)
8	00h	Successful response, giving offset of mapped ADI

Second command:

Area offset	Contents of request	Description (request)
0	03h	Source ID
1	03h	The Network Object in the Anybus CompactCom module.
2 - 3	01h 00h	Instance 1
4	51h	Message type: command (Map_ADI_Read_Area)
5	04h	Size of Message Data (4)
6 - 7	02h 00h	ADI instance number (2)
8	04h	Data type UINT8
9	01h	Number of elements in ADI (1)
10 - 11	02h 00h	Order number of the ADI (2)

Second response:

Area offset	Contents of response	Description (response)
0	03h	Source ID
1	03h	The Network Object in the Anybus CompactCom module.
2 - 3	01h 00h	Instance 1
4	11h	Message type: response
5	01h	Size of Message Data (1 byte)
6 - 7	02h 00h	ADI instance number (2)
8	02h	Successful response, giving offset of mapped ADI

Note:

- The offset of the mapped ADI is 2 in has second response, as the first ADI that was mapped has the data type UINT8.

For details, see “Network Object” and “Message Layout” in the Anybus CompactCom Software Design Guide.

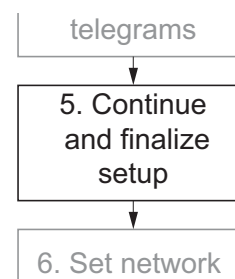
SWDG

3.7.4 Setup Complete

To finish setup the application sends this message to the module:

Example:

The following message is sent to the module:
 {0x04, 0x01, 0x01, 0x00, 0x42, 0x01, 0x05, 0x00, 0x01}
 The module responds with:
 {0x04, 0x01, 0x01, 0x00, 0x02, 0x00, 0x05, 0x00}



Command:

Area offset	Contents of request	Description (request)
0	04h	Source ID
1	01h	The Anybus object in the Anybus CompactCom module.
2	01h (lsb)	Instance 1
3	00h (msb)	
4	42h	Message type: command (Set_attribute)
5	01h	Size of Message Data (1 byte)
6	05h	Attribute 5 (Setup complete)
7	00h	(not used)
8	01h	Setup complete attribute set to 1

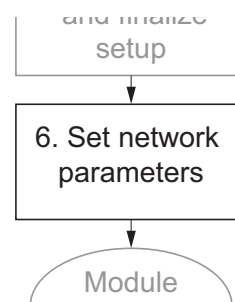
Response:

Area offset	Contents of response	Description (response)
0	04h	Source ID
1	01h	The Anybus object in the Anybus CompactCom module.
2	01h (lsb)	Instance 1
3	00h (msb)	
4	02h	Message type: response (the command is copied, with the C bit set to 0)
5	00h	Size of Message Data (0 bytes)
6	05h	Attribute 5 (Setup Complete)
7	00h	(not used)

Once setup is finished, signalled by the Setup Complete attribute set to 1, the Anybus CompactCom will have changed states from SETUP to NW_INIT. The state of the module is always presented on bits b2, b1 and b0 in the Status Register, see page 12.

3.8 Network Initialization

The Anybus CompactCom has now entered the NW_INIT state. The module will start to send commands to the application, to initialize the network communication. Exactly what commands are sent, depend on choice of industrial network, see appendices for examples (DeviceNet on page 38 and PROFIBUS DP-V1 on page 44). For each command received, the application must give a valid response to the module, even if the requested objects is not implemented in the application. In this case a request to return the Vendor ID, that is a parameter in the DeviceNet Object, is sent from the module. This object is not implemented in the application, so the application returns an error message that says “unsupported object”.



Example:

The following message is sent from the module to the application:
 {0x06, 0xfc, 0x01, 0x00, 0x41, 0x00, 0x01, 0x00}

The application responds with:
 {0x06, 0xfc, 0x01, 0x00, 0x81, 0x01, 0x01, 0x00, 0x03}

Command (from the module to the host application):

Area offset	Contents of request	Description (request)
0	06h	Source ID
1	FCh	The DeviceNet object in the host application.
2	01h (lsb)	Instance 1
3	00h (msb)	
4	41h	Message type: command (Get_attribute)
5	00h	Size of Message Data (0 bytes)
6	01h	Attribute 1 (Vendor ID)
7	00h	(not used)

Response (from the host application to the module):

Area offset	Contents of response	Description (response)
0	06h	Source ID
1	FCh	The DeviceNet object in the host application.
2	01h (lsb)	Instance 1
3	00h (msb)	
4	81h	Message type: error response
5	01h	Size of Message Data (1 byte)
6	01h	Attribute 1 (Vendor ID)
7	00h	(not used)
8	03h	Error message: Unsupported object

As mentioned above, the commands will differ between the industrial networks supported by Anybus CompactCom. Also the number of commands will differ. There are default values for all attributes that are requested by the module, so as long as a valid response is given (as shown in the example), the module will continue the network initialization. Once finished it will enter the WAIT_PROCESS state, see “Anybus CompactCom State Machine” on page 28.

3.9 Further Configuration and Certification

The examples given above show a very simple implementation. The host application can contain several more objects of different kinds, and in some cases it is recommended to implement some of them, e.g. when the end product will be certified with the fieldbus organization in question.

Please consult the respective appendices where any further necessary settings and initializations are described.

4. Resources

The previous chapter gives a relatively simple example of how to implement an application with an Anybus CompactCom module. The modules are very versatile, and include a lot more functionality than is used in this tutorial. This chapter lists other documentation and material, that can be used when making your own application.

4.1 Categorization of Functionality

The objects, including attributes and services, of the Anybus CompactCom and the application are divided into three categories: basic, advanced and extended.

4.1.1 Basic

This category includes objects, attributes and services that are mandatory to implement or to use. They will be enough for starting up the Anybus CompactCom and sending/receiving data with the chosen network protocol. The basic functions of the industrial network are used.

These objects are presented in this tutorial (and the appropriate industrial network appendix). Additional objects etc, that will make it possible to certify the product also belong to this category.

4.1.2 Extended

Use of the objects in this category extends the functionality of the application. Access is given to the more specific characteristics of the industrial network, not only the basic moving of data to and from the network. Extra value is given to the application.

4.1.3 Advanced

The objects, attributes and services that belong to this group offer specialized and/or seldom used functionality. Most of the available network functionality is enabled and accessible. Access to the specification of the industrial network is normally required.

4.2 Design Guides

4.2.1 Anybus CompactCom Software Design Guide

This document is the complete manual for the software interface. It provides information, that covers all parts of the software interface that are common to all Anybus CompactCom communication modules.

4.2.2 Anybus CompactCom Hardware Design Guide

This document is intended to provide a good understanding of the mechanical and electric properties of the Anybus CompactCom platform.

4.3 Network Interface Appendices

Each module is accompanied by an appendix describing the network interface specific issues, such as what is necessary to implement in an application, so that it can be certified.

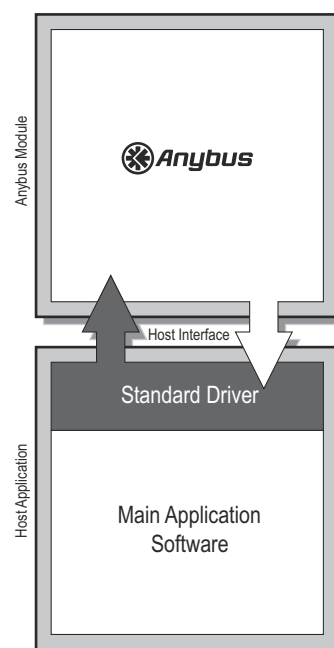
4.4 Drivers

HMS supplies two free source level (C language) software drivers, the Standard Driver and the Lite Driver. These drivers are designed to speed up the development process, by acting as “glue” between the Anybus module and the host application. They separate low level communication tasks from the host software environment.

The Standard Driver is designed to exploit the versatility of the Anybus-CompactCom concept while still keeping a high level of abstraction and flexibility. This means that it needs a certain degree of memory and processing power by itself, which may prove impractical in smaller applications. To bridge this gap, HMS supplies an alternative driver, known as the ‘Lite Driver’, which provides a bare-bones solution suitable for applications with tight memory and/or performance demands.

The Standard Driver is completely self-contained, i.e. it does not require an operating system to function, and can be run either as an interrupt driven service or polled cyclically by the host firmware.

The drivers, including documentation are available for download at www.anybus.com.



4.5 Starter Kit

A starter kit is available for the Anybus CompactCom platform. It includes an evaluation board which can be used to develop networking applications via the serial host interface channel.

The starter kit also includes a Software Development Kit (SDK). The aim of this is to show how a software application can be implemented using the Anybus CompactCom driver for communication with the Anybus CompactCom.

The SDK is designed to run as a console application on win32 PC, communicating with the Anybus CompactCom using a serial port.

Please visit www.anybus.com for more information.

4.6 Drive Profiles

The Anybus-CompactCom Drive Profile range of products extends the Anybus-CompactCom concept with additional Drive Profile functionality. This extended software functionality makes it easier for manufacturers to make products which comply to the latest communication standards for drives. This tutorial does not cover drive profiles, but the issues covered are of course valid for those products as well. On www.anybus.com documentation is available both for drive profiles in general and for those modules that incorporate drive profiles.

Appendix A

A. Trace, DeviceNet

This appendix holds a transcription of an example communication in parallel mode between a host application and an Anybus CompactCom DeviceNet module. The **Data** column contains what is either written to or read from the specified DPRAM area by the host application. See also the Anybus CompactCom Software Design Guide and the Network Interface Appendix for DeviceNet.

R/W	DPRAM area	Data	Destination Object	Instance	Command/Response	Information
W	Control Register	0x80				Initial Handshake
R	Status Register	0x80				
W	Control Register	0x00				
R	Status Register	0x00				
W	Control Register	0x80				
R	Status Register	0xa0				
Ask for module type						
W	Message Write Area	0x01, 0x01, 0x01, 0x00, 0x41, 0x00, 0x01, 0x00	Anybus (0x01)	1	Get_Attribute	Attr = MODULE_TYPE (0x01)
W	Control Register	0xc0				
R	Status Register	0xa0				
R	Message Read Area	0x01, 0x01, 0x01, 0x00, 0x01, 0x02, 0x01, 0x00, 0x01, 0x04			ACK	ModuleType = ABCC (0x0401)
Map one byte to Read Process Data						
W	Message Write Area	0x02, 0x03, 0x01, 0x00, 0x51, 0x04, 0x01, 0x00, 0x04, 0x01, 0x01, 0x00	Network (0x03)	1	Map_ADI_Read_Area	ADI = 0x0001, DataType = UINT8 (0x04), NumElements = 0x01, OrderNum = 0x0001
W	Control Register	0xc0				
R	Status Register	0xa0				
W	Control Register	0x00				
R	Status Register	0x60				
R	Message Read Area	0x02, 0x03, 0x01, 0x00, 0x11, 0x01, 0x01, 0x00, 0x00			ACK	AreaOffset = 0x00
Set Setup Complete						
W	Message Write Area	0x03, 0x01, 0x01, 0x00, 0x42, 0x01, 0x05, 0x00, 0x01	Anybus (0x01)	1	Set_Attribute	Attr = SETUP_COMPLETE (0x05), Value = TRUE (≠0x00)
W	Control Register	0xc0				
R	Status Register	0xe0				
R	Message Read Area	0x03, 0x01, 0x01, 0x00, 0x02, 0x00, 0x05, 0x00			ACK	
W	Control Register	0x00				
R	Status Register	0x21				(Anybus State = NW_INIT)
W	Control Register	0x80				

R/W	DPRAM area	Data	Destination Object	Instance	Command/Response	Information
R	Status Register	0xa1				
W	Control Register	0x00				
R	Status Register	0x21				
Accept commands from module and send error responses in return (CTRL_R = 1)						
W	Control Register	0xa0				
R	Status Register	0xe1				
R	Message Read Area	0x06, 0xfc, 0x01, 0x00, 0x41, 0x00, 0x01, 0x00	DeviceNet (0xfc)	1	Get_Attribute	Attr = VENDOR_ID (0x01)
W	Message Write Area	0x06, 0xfc, 0x01, 0x00, 0x81, 0x01, 0x01, 0x00, 0x03			NAK	Error = UNSUP_OBJ (0x03)
W	Control Register	0x60				
R	Status Register	0x21				
W	Control Register	0xa0				
R	Status Register	0xe1				
R	Message Read Area	0x06, 0xfc, 0x01, 0x00, 0x41, 0x00, 0x02, 0x00	DeviceNet (0xfc)	1	Get_Attribute	Attr = DEVICE_TYPE (0x02)
W	Message Write Area	0x06, 0xfc, 0x01, 0x00, 0x81, 0x01, 0x02, 0x00, 0x03			NAK	Error = UNSUP_OBJ (0x03)
W	Control Register	0x60				
R	Status Register	0x21				
W	Control Register	0xa0				
R	Status Register	0xe1				
R	Message Read Area	0x06, 0xfc, 0x01, 0x00, 0x41, 0x00, 0x03, 0x00	DeviceNet (0xfc)	1	Get_Attribute	Attr = PRODUCT_CODE (0x03)
W	Message Write Area	0x06, 0xfc, 0x01, 0x00, 0x81, 0x01, 0x03, 0x00, 0x03			NAK	Error = UNSUP_OBJ (0x03)
W	Control Register	0x60				
R	Status Register	0x21				
W	Control Register	0xa0				
R	Status Register	0xe1				
R	Message Read Area	0x06, 0xfc, 0x01, 0x00, 0x41, 0x00, 0x04, 0x00	DeviceNet (0xfc)	1	Get_Attribute	Attr = REVISION (0x04)
W	Message Write Area	0x06, 0xfc, 0x01, 0x00, 0x81, 0x01, 0x04, 0x00, 0x03			NAK	Error = UNSUP_OBJ (0x03)

R/W	DPRAM area	Data	Destination Object	Instance	Command/Response	Information
W	Control Register	0x60				
R	Status Register	0x21				
W	Control Register	0xa0				
R	Status Register	0xe1				
R	Message Read Area	0x06, 0xfc, 0x01, 0x00, 0x41, 0x00, 0x05, 0x00	DeviceNet (0xfc)	1	Get_Attribute	Attr = SERIAL_NUMBER (0x05)
W	Message Write Area	0x06, 0xfc, 0x01, 0x00, 0x81, 0x01, 0x05, 0x00, 0x03			NAK	Error = UNSUP_OBJ (0x03)
W	Control Register	0x60				
R	Status Register	0x21				
W	Control Register	0xa0				
R	Status Register	0xe1				
R	Message Read Area	0x06, 0xfc, 0x01, 0x00, 0x41, 0x00, 0x06, 0x00	DeviceNet (0xfc)	1	Get_Attribute	Attr = PRODUCT_NAME (0x06)
W	Message Write Area	0x06, 0xfc, 0x01, 0x00, 0x81, 0x01, 0x06, 0x00, 0x03			NAK	Error = UNSUP_OBJ (0x03)
W	Control Register	0x60				
R	Status Register	0x21				
W	Control Register	0xa0				
R	Status Register	0xe1				
R	Message Read Area	0x06, 0xfc, 0x01, 0x00, 0x41, 0x00, 0x08, 0x00	DeviceNet (0xfc)	1	Get_Attribute	Attr = CONS_INSTANCE (0x08)
W	Message Write Area	0x06, 0xfc, 0x01, 0x00, 0x81, 0x01, 0x08, 0x00, 0x03			NAK	Error = UNSUP_OBJ (0x03)
W	Control Register	0x60				
R	Status Register	0x21				
W	Control Register	0xa0				
R	Status Register	0xe1				
R	Message Read Area	0x06, 0xfc, 0x01, 0x00, 0x41, 0x00, 0x07, 0x00	DeviceNet (0xfc)	1	Get_Attribute	Attr = PROD_INSTANCE (0x07)
W	Message Write Area	0x06, 0xfc, 0x01, 0x00, 0x81, 0x01, 0x07, 0x00, 0x03			NAK	Error = UNSUP_OBJ (0x03)
W	Control Register	0x60				
R	Status Register	0x21				

R/W	DPRAM area	Data	Destination Object	Instance	Command/Response	Information
W	Control Register	0xa0				
R	Status Register	0xe1				
R	Message Read Area	0x06, 0xfc, 0x01, 0x00, 0x41, 0x00, 0x09, 0x00	DeviceNet (0xfc)	1	Get_Attribute	Attr = ADDRESS_FROM_NET (0x09)
W	Message Write Area	0x06, 0xfc, 0x01, 0x00, 0x81, 0x01, 0x09, 0x00, 0x03			NAK	Error = UNSUP_OBJ (0x03)
W	Control Register	0x60				
R	Status Register	0x21				
W	Control Register	0xa0				
R	Status Register	0xe1				
R	Message Read Area	0x06, 0xfc, 0x01, 0x00, 0x41, 0x00, 0x0a, 0x00	DeviceNet (0xfc)	1	Get_Attribute	Attr = BAUD_RATE_FROM_NET (0x0a)
W	Message Write Area	0x06, 0xfc, 0x01, 0x00, 0x81, 0x01, 0x0a, 0x00, 0x03			NAK	Error = UNSUP_OBJ (0x03)
W	Control Register	0x60				
R	Status Register	0x21				
W	Control Register	0xa0				
R	Status Register	0xe1				
R	Message Read Area	0x06, 0xfc, 0x01, 0x00, 0x41, 0x00, 0x0b, 0x00	DeviceNet (0xfc)	1	Get_Attribute	Attr = ENABLE_APP_CIP_OBJECTS (0x0b)
W	Message Write Area	0x06, 0xfc, 0x01, 0x00, 0x81, 0x01, 0x0b, 0x00, 0x03			NAK	Error = UNSUP_OBJ (0x03)
W	Control Register	0x60				
R	Status Register	0x21				
W	Control Register	0xa0				
R	Status Register	0xe1				
R	Message Read Area	0x06, 0xfc, 0x01, 0x00, 0x41, 0x00, 0x0c, 0x00	DeviceNet (0xfc)	1	Get_Attribute	Attr = ENABLE_PARAM_OBJECT (0x0c)
W	Message Write Area	0x06, 0xfc, 0x01, 0x00, 0x81, 0x01, 0x0c, 0x00, 0x03			NAK	Error = UNSUP_OBJ (0x03)
W	Control Register	0x60				
R	Status Register	0x21				
W	Control Register	0xa0				
R	Status Register	0xe1				

R/W	DPRAM area	Data	Destination Object	Instance	Command/Response	Information
R	Message Read Area	0x06, 0xfc, 0x01, 0x00, 0x41, 0x00, 0x0d, 0x00	DeviceNet (0xfc)	1	Get_Attribute	Attr = ENABLE_QUICK_CONNECT (0x0d)
W	Message Write Area	0x06, 0xfc, 0x01, 0x00, 0x81, 0x01, 0x0d, 0x00, 0x03			NAK	Error = UNSUP_OBJ (0x03)
W	Control Register	0x60				
R	Status Register	0x21				
W	Control Register	0xa0				
R	Status Register	0xe1				
R	Message Read Area	0x06, 0xff, 0x01, 0x00, 0x41, 0x00, 0x02, 0x00	Application (0xff)	1	Get_Attribute	Attr = SUP_LANG (0x02)
W	Message Write Area	0x06, 0xff, 0x01, 0x00, 0x81, 0x01, 0x02, 0x00, 0x03			NAK	Error = UNSUP_OBJ (0x03)
W	Control Register	0x60				
R	Status Register	0x21				
W	Control Register	0xa0				
R	Status Register	0xa1				
W	Control Register	0x20				
R	Status Register	0x22				(Anybus State = WAIT_PROCESS)

Appendix B

B. Trace, Profibus DP-V1

This appendix holds a transcription of an example communication in parallel mode between a host application and an Anybus CompactCom Profibus DP-V1 module. The **Data** column contains what is either written to or read from the specified DPRAM area by the host application. See also the Anybus CompactCom Software Design Guide and the Network Interface Appendix for Profibus DP-V1.

R/W	DPRAM area	Data	Destination Object	Instance	Command/ Response	Information
W	Control Register	0x80				Initial Handshake
R	Status Register	0x80				
W	Control Register	0x00				
R	Status Register	0x00				
W	Control Register	0x80				
R	Status Register	0xa0				
Ask for module type						
W	Message Write Area	0x01, 0x01, 0x01, 0x00, 0x41, 0x00, 0x01, 0x00	Anybus (0x01)	1	Get_Attribute	Attr = MODULE_TYPE (0x01)
W	Control Register	0x40				
R	Status Register	0x60				
R	Message Read Area	0x01, 0x01, 0x01, 0x00, 0x01, 0x02, 0x01, 0x00, 0x01, 0x04			ACK	ModuleType = ABCC (0x0401)
Map one byte to Read Process Data						
W	Message Write Area	0x02, 0x03, 0x01, 0x00, 0x51, 0x04, 0x01, 0x00, 0x04, 0x01, 0x01, 0x00	Network (0x03)	1	Map_ADI_Read_Area	ADI = 0x0001, DataType = UINT8 (0x04), NumElements = 0x01, OrderNum = 0x0001
W	Control Register	0xc0				
R	Status Register	0xa0				
W	Control Register	0x00				
R	Status Register	0x60				
R	Message Read Area	0x02, 0x03, 0x01, 0x00, 0x11, 0x01, 0x01, 0x00, 0x00			ACK	AreaOffset = 0x00
Set Setup Complete						
W	Message Write Area	0x03, 0x01, 0x01, 0x00, 0x42, 0x01, 0x05, 0x00, 0x01	Anybus (0x01)	1	Set_Attribute	Attr = SETUP_COMPLETE (0x05), Value = TRUE (≠0x00)
W	Control Register	0xc0				
R	Status Register	0xe0				
R	Message Read Area	0x03, 0x01, 0x01, 0x00, 0x02, 0x00, 0x05, 0x00			ACK	
W	Control Register	0x00				
R	Status Register	0x21				(Anybus State = NW_INIT)
W	Control Register	0x80				

R/W	DPRAM area	Data	Destination Object	Instance	Command/ Response	Information
R	Status Register	0xa1				
W	Control Register	0x00				
R	Status Register	0x21				
Accept commands from module and send error responses in return (CTLR_R = 1)						
W	Control Register	0xa0				
R	Status Register	0xe1				
R	Message Read Area	0x00, 0xfd, 0x01, 0x00, 0x41, 0x00, 0x01, 0x00	PROFIBUS DP-V1 (0xfd)	1	Get_Attribute	Attr = IDENT_NUMBER (0x01)
W	Message Write Area	0x00, 0xfd, 0x01, 0x00, 0x81, 0x01, 0x01, 0x00, 0x03			NAK	Error = UNSUP_OBJ (0x03)
W	Control Register	0x60				
R	Status Register	0x21				
W	Control Register	0xa0				
R	Status Register	0xe1				
R	Message Read Area	0x01, 0xfd, 0x01, 0x00, 0x41, 0x00, 0x06, 0x00	PROFIBUS DP-V1 (0xfd)	1	Get_Attribute	Attr = BUFFER_MODE (0x06)
W	Message Write Area	0x01, 0xfd, 0x01, 0x00, 0x81, 0x01, 0x06, 0x00, 0x03			NAK	Error = UNSUP_OBJ (0x03)
W	Control Register	0x60				
R	Status Register	0x21				
W	Control Register	0xa0				
R	Status Register	0xe1				
R	Message Read Area	0x00, 0xfd, 0x01, 0x00, 0x41, 0x00, 0x03, 0x00	PROFIBUS DP-V1 (0xfd)	1	Get_Attribute	Attr = EXPECTED_CFG_DATA (0x03)
W	Message Write Area	0x00, 0xfd, 0x01, 0x00, 0x81, 0x01, 0x03, 0x00, 0x03			NAK	Error = UNSUP_OBJ (0x03)
W	Control Register	0x60				
R	Status Register	0x21				
W	Control Register	0xa0				
R	Status Register	0xe1				
R	Message Read Area	0x01, 0xfd, 0x01, 0x00, 0x41, 0x00, 0x05, 0x00	PROFIBUS DP-V1 (0xfd)	1	Get_Attribute	Attr = SIZEOF_ID_REL_DIAG (0x05)
W	Message Write Area	0x01, 0xfd, 0x01, 0x00, 0x81, 0x01, 0x05, 0x00, 0x03			NAK	Error = UNSUP_OBJ (0x03)

R/W	DPRAM area	Data	Destination Object	Instance	Command/ Response	Information
W	Control Register	0x60				
R	Status Register	0x21				
W	Control Register	0xa0				
R	Status Register	0xe1				
R	Message Read Area	0x00, 0xfd, 0x01, 0x00, 0x41, 0x00, 0x07, 0x00	PROFIBUS DP-V1 (0xfd)	1	Get_Attribute	Attr = ALARM_SETTINGS (0x07)
W	Message Write Area	0x00, 0xfd, 0x01, 0x00, 0x81, 0x01, 0x07, 0x00, 0x03			NAK	Error = UNSUP_OBJ (0x03)
W	Control Register	0x60				
R	Status Register	0x21				
W	Control Register	0xa0				
R	Status Register	0xe1				
R	Message Read Area	0x01, 0xfd, 0x01, 0x00, 0x41, 0x00, 0x08, 0x00	PROFIBUS DP-V1 (0xfd)	1	Get_Attribute	Attr = MANUFACTURER_ID (0x08)
W	Message Write Area	0x01, 0xfd, 0x01, 0x00, 0x81, 0x01, 0x08, 0x00, 0x03			NAK	Error = UNSUP_OBJ (0x03)
W	Control Register	0x60				
R	Status Register	0x21				
W	Control Register	0xa0				
R	Status Register	0xe1				
R	Message Read Area	0x00, 0xfd, 0x01, 0x00, 0x41, 0x00, 0x09, 0x00	PROFIBUS DP-V1 (0xfd)	1	Get_Attribute	Attr = ORDER_ID (0x09)
W	Message Write Area	0x00, 0xfd, 0x01, 0x00, 0x81, 0x01, 0x09, 0x00, 0x03			NAK	Error = UNSUP_OBJ (0x03)
W	Control Register	0x60				
R	Status Register	0x21				
W	Control Register	0xa0				
R	Status Register	0xe1				
R	Message Read Area	0x01, 0xfd, 0x01, 0x00, 0x41, 0x00, 0x0a, 0x00	PROFIBUS DP-V1 (0xfd)	1	Get_Attribute	Attr = SERIAL_NO (0x0a)
W	Message Write Area	0x01, 0xfd, 0x01, 0x00, 0x81, 0x01, 0x0a, 0x00, 0x03			NAK	Error = UNSUP_OBJ (0x03)
W	Control Register	0x60				
R	Status Register	0x21				

R/W	DPRAM area	Data	Destination Object	Instance	Command/ Response	Information
W	Control Register	0xa0				
R	Status Register	0xe1				
R	Message Read Area	0x00, 0xfd, 0x01, 0x00, 0x41, 0x00, 0x0b, 0x00	PROFIBUS DP-V1 (0xfd)	1	Get_Attribute	Attr = HW_REV (0x0b)
W	Message Write Area	0x00, 0xfd, 0x01, 0x00, 0x81, 0x01, 0x0b, 0x00, 0x03			NAK	Error = UNSUP_OBJ (0x03)
W	Control Register	0x60				
R	Status Register	0x21				
W	Control Register	0xa0				
R	Status Register	0xe1				
R	Message Read Area	0x01, 0xfd, 0x01, 0x00, 0x41, 0x00, 0x0c, 0x00	PROFIBUS DP-V1 (0xfd)	1	Get_Attribute	Attr = SW_REV (0x0c)
W	Message Write Area	0x01, 0xfd, 0x01, 0x00, 0x81, 0x01, 0x0c, 0x00, 0x03			NAK	Error = UNSUP_OBJ (0x03)
W	Control Register	0x60				
R	Status Register	0x21				
W	Control Register	0xa0				
R	Status Register	0xe1				
R	Message Read Area	0x00, 0xfd, 0x01, 0x00, 0x41, 0x00, 0x0e, 0x00	PROFIBUS DP-V1 (0xfd)	1	Get_Attribute	Attr = PROFILE_ID (0x0e)
W	Message Write Area	0x00, 0xfd, 0x01, 0x00, 0x81, 0x01, 0x0e, 0x00, 0x03			NAK	Error = UNSUP_OBJ (0x03)
W	Control Register	0x60				
R	Status Register	0x21				
W	Control Register	0xa0				
R	Status Register	0xe1				
R	Message Read Area	0x01, 0xfd, 0x01, 0x00, 0x41, 0x00, 0x0f, 0x00	PROFIBUS DP-V1 (0xfd)	1	Get_Attribute	Attr = PROFILE_SPEC_TYPE (0x0f)
W	Message Write Area	0x01, 0xfd, 0x01, 0x00, 0x81, 0x01, 0x0f, 0x00, 0x03			NAK	Error = UNSUP_OBJ (0x03)
W	Control Register	0x60				
R	Status Register	0x21				
W	Control Register	0xa0				
R	Status Register	0xe1				

R/W	DPRAM area	Data	Destination Object	Instance	Command/ Response	Information
R	Message Read Area	0x00, 0xfd, 0x01, 0x00, 0x41, 0x00, 0x10, 0x00	PROFIBUS DP-V1 (0xfd)	1	Get_Attribute	Attr = IM_VERSION (0x10)
W	Message Write Area	0x00, 0xfd, 0x01, 0x00, 0x81, 0x01, 0x10, 0x00, 0x03			NAK	Error = UNSUP_OBJ (0x03)
W	Control Register	0x60				
R	Status Register	0x21				
W	Control Register	0xa0				
R	Status Register	0xe1				
R	Message Read Area	0x01, 0xfd, 0x01, 0x00, 0x41, 0x00, 0x11, 0x00	PROFIBUS DP-V1 (0xfd)	1	Get_Attribute	Attr = IM_SUPPORTED (0x11)
W	Message Write Area	0x01, 0xfd, 0x01, 0x00, 0x81, 0x01, 0x11, 0x00, 0x03			NAK	Error = UNSUP_OBJ (0x03)
W	Control Register	0x60				
R	Status Register	0x21				
W	Control Register	0xa0				
R	Status Register	0xe1				
R	Message Read Area	0x00, 0xfd, 0x01, 0x00, 0x41, 0x00, 0x12, 0x00	PROFIBUS DP-V1 (0xfd)	1	Get_Attribute	Attr = IM_HEADER (0x12)
W	Message Write Area	0x00, 0xfd, 0x01, 0x00, 0x81, 0x01, 0x12, 0x00, 0x03			NAK	Error = UNSUP_OBJ (0x03)
W	Control Register	0x60				
R	Status Register	0x21				
W	Control Register	0xa0				
R	Status Register	0xa2				(Anybus State = WAIT_PROCESS)

Support

HMS Sweden (Head Office)

E-mail: support@hms-networks.com
Phone: +46 (0) 35 - 17 29 20
Fax: +46 (0) 35 - 17 29 09
Online: www.anybus.com

HMS North America

E-mail: us-support@hms-networks.com
Phone: +1-312-829-0601
Toll Free: +1-888-8-Anybus
Fax: +1-312-738-5873
Online: www.anybus.com

HMS Germany

E-mail: ge-support@hms-networks.com
Phone: +49-721-96472-0
Fax: +49-721-964-7210
Online: www.anybus.com

HMS Japan

E-mail: jp-support@hms-networks.com
Phone: +81-45-478-5340
Fax: +81-45-476-0315
Online: www.anybus.com

HMS China

E-mail: cn-support@hms-networks.com
Phone: +86 10 8532 3023
Online: www.anybus.com

HMS Italy

E-mail: it-support@hms-networks.com
Phone: +39 039 59662 27
Fax: +39 039 59662 31
Online: www.anybus.com

HMS France

E-mail: mta@hms-networks.com
Phone: +33 (0) 3 89 32 76 41
Fax: +33 (0) 3 89 32 76 31
Online: www.anybus.com