

Parallel Interface Design Guide Anybus[®]-S Slave & Master

Doc.Id. HMSI-27-275
Rev. 3.00



HALMSTAD • CHICAGO • KARLSRUHE • TOKYO • BEIJING • MILANO • MULHOUSE • COVENTRY • PUNE • COPENHAGEN

HMS Industrial Networks
Mailing address: Box 4126, 300 04 Halmstad, Sweden
Visiting address: Stationsgatan 37, Halmstad, Sweden

E-mail: info@hms-networks.com
www.anybus.com

Important User Information

This document is intended to provide a good understanding of the functionality offered by a module in the Anybus-S Slave & Master family.

The reader of this document is expected to be familiar with high level software design, and communication systems in general.

2.0.1 Liability

Every care has been taken in the preparation of this manual. 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.

2.0.2 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.

2.0.3 Trademark Acknowledgements

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

<p>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.</p> <p>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.</p>

Table of Contents

Preface	About This Manual
	Related Documentation 7
	Document History 7
	Conventions Used in This Manual 8
	Support 8
Chapter 1	Introduction
	Key Features 9
	Internals 10
	External View 11
Chapter 2	Application Connector
	Connector Pinout..... 13
	Control Signals..... 13
	Asynchronous Serial Interface 15
Chapter 3	Memory Map
Chapter 4	Control Register Area
	Registers..... 18
Chapter 5	Handshaking & Indication Registers
	Application Indication Register (7FEh, R/W) 26
	Anybus Indication Register (7FFh, RO)..... 27
	Collisions 28
	Area Allocation/De-allocation..... 29
	<i>Unsynchronized Data Exchange</i> 29
	<i>Synchronised Data Exchange</i> 30
	<i>Requesting/Releasing Multiple Areas Simultaneously</i> 31
	<i>Application Example, Cyclic Access Method</i> 32
Chapter 6	Interrupts
	Hardware Interrupt (IRQ) 34
	Event Notification (Software Interrupt)..... 35

Chapter 7	Fieldbus Data Exchange	
	Basics	36
	Dual Port Memory vs. Internal Memory	37
	Data types	37
	Data Composition	38
 Chapter 8	 Mailbox Interface	
	General	39
	Message Types	39
	Mailbox Notification Bits	40
	<i>Sending a Mailbox Message</i>	41
	<i>Receiving a Mailbox Message</i>	41
	Mailbox Message Structure	42
	Message Header	42
 Chapter 9	 Mailbox Messages	
	Application Messages	44
	<i>Start Initialization (START_INIT)</i>	45
	<i>Anybus Initialization (Anybus_INIT)</i>	46
	<i>End Initialization (END_INIT)</i>	48
	<i>Save to FLASH (SAVE_TO_FLASH)</i>	49
	<i>Load from FLASH (LOAD_FROM_FLASH)</i>	50
	<i>Hardware Check (HW_CHK)</i>	51
	Fieldbus Messages	52
	Internal Memory Messages	52
	<i>Read Internal Input Area (RD_INT_IN)</i>	53
	<i>Write Internal Input Area (WR_INT_IN)</i>	54
	<i>Clear Internal Input Area (CLR_INT_IN)</i>	55
	<i>Read Internal Output Area (RD_INT_OUT)</i>	56
	Reset Messages	57
	<i>Software Reset (SW_RESET)</i>	57

Chapter 10 Start Up and Initialization

Introduction	58
Hardware Initialization	59
<i>Startup Sequence</i>	59
<i>Dual Port Memory Check (Optional)</i>	59
<i>Hardware Check (Optional)</i>	60
Software Initialization	60
<i>Prepare Initialization Data</i>	60
<i>Start Initialization</i>	60
<i>Initialise Parameter Values</i>	61
<i>Set Initial Fieldbus Data</i>	61
<i>End Initialization</i>	61
<i>Basic Initialization Sequence Example 1</i>	62
<i>Basic Initialization Sequence Example 2</i>	62
<i>Advanced Initialization Example</i>	63

Chapter 11 Indication LEDs

Fieldbus Status Indicators	64
Anybus-S Watchdog LED	64

Chapter 12 Firmware Upgrade

Chapter 13 Driver Example

Interrupt Handler	67
Interface Handler	68
Mailbox Handler	69

Chapter 14 Mechanical Aspects

PCB Measurements	70
Height Restrictions	70
Mounting Holes	71
Application Connector	71
Fieldbus Connector(s)	71
Fieldbus Status Indication LED's	72

Appendix A Extended Memory Mode (4K DPRAM)

Appendix B Deviances

General	74
Locked Release Behavior	74
Application Interface Hardware Deviances	75

Appendix C Interrupt Line and Hardware Reset

Recommended Solution	77
----------------------------	----

Appendix D Electrical Specification

Power Supply Requirements.....	78
Signal Characteristics	79
PE & Shielding Recommendations	79

Appendix E Environmental Specification

Temperature	80
Relative Humidity.....	80
EMC compliance.....	80

Appendix F Conformance with Predefined Standards

Fieldbus Certification.....	81
CE-Mark	81
UL/cUL-Certificate	81

Appendix G Troubleshooting

P. About This Manual

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

P.1 Related Documentation

Document	Author
Anybus-S API Reference Manual	HMS (www.hms-networks.com)
Anybus-S Fieldbus Appendices (one for each fieldbus)	
Data sheet for dual port memory (CY7C136)	Cypress (www.cypress.com)
Understanding Asynchronous Dual-Port RAMs (application note)	

P.2 Document History

Summary of Recent Changes (2.08...3.00)

Change	Page(s)
Updated Firmware Upgrade chapter	65
Updated Electric Signal Characteristics	79

Revision List

Rev.	Date	Author	Chapter	Description
2.00	2004-02-19	PeP	All	Second major release
2.01	2004-06-17	ToT	4	Corrected online/offline indication for 'Module Status Register'
2.02	2005-07-19	PeP	9	Corrected Anybus_INIT response (Fault Information)
			D	Corrected signal levels (Reset signal)
			2	Corrected pull-up resistance & decoupling (Reset signal)
2.03	2008-10-14	HeS	2	Renamed /CS, /RD, /WR to CE, OE, R/W
			10	Updated exclusions during Dual Port Memory Check
			14	Corrected DCP measures in drawing
			1,9	Misc. minor updates
2.04	2009-09-10	KeL	6, 8, D, 12, 4, 5, B	Misc. minor updates
2.05	2009-11-13	KeL	4, 5, A	Misc. minor updates
2.06	2010-01-12	KeL	6, 7, D	Misc. minor updates
2.07	2010-04-16	KeL	4	Minor update
2.08	2010-12-17	KeL	P, 4	Minor update
3.00	2014-09-19	KeL	12, D	Updated template, misc. minor updates

P.3 Conventions Used in This Manual

The following conventions are used throughout this manual:

- Numbered lists provide sequential steps
- Bulleted lists provide information, not procedural steps
- The term ‘module’ is used when referring to the Anybus module
- The term ‘application’ is used when referring to the hardware that is connected to the Anybus Application Connector
- Hexadecimal values are written in the format NNNNh, where NNNN is the hexadecimal value.
- All measurements expressed in this document have a tolerance of $\pm 0.25\text{mm}$ unless otherwise stated.
- 16/32 bit values are generally stored in Motorola (big endian) format unless otherwise stated.

P.4 Support

For general contact information and support, please refer to the contact and support pages at www.anybus.com.

1. Introduction

The Anybus-S/Anybus-M is a series of interchangeable fieldbus communication modules featuring on board memory and processing power. All software and hardware functionality required to communicate on the fieldbus is incorporated in the module itself, allowing the application to focus on other tasks.

The interface towards the application is based on a dual port memory architecture, where the host application and the Anybus module exchange data via a shared memory area. This allows for very efficient data exchange, and generally produces very little overhead for the host application.

Standardisation of mechanical, electrical and software interfaces ensures that the different Anybus-S/Anybus-M models are fully interchangeable. This also means that the same PCB layout can be used for different fieldbus systems.

Typical applications are frequency inverters, HMI and visualization devices, instruments, scales, robotics, PLC's and intelligent measuring devices.

Note: The application interface of the Anybus-M is identical to that of the Anybus-S. Therefore, all further references in this manual will be made to the Anybus-S; The information does however apply equally to the Anybus-M.

1.1 Key Features

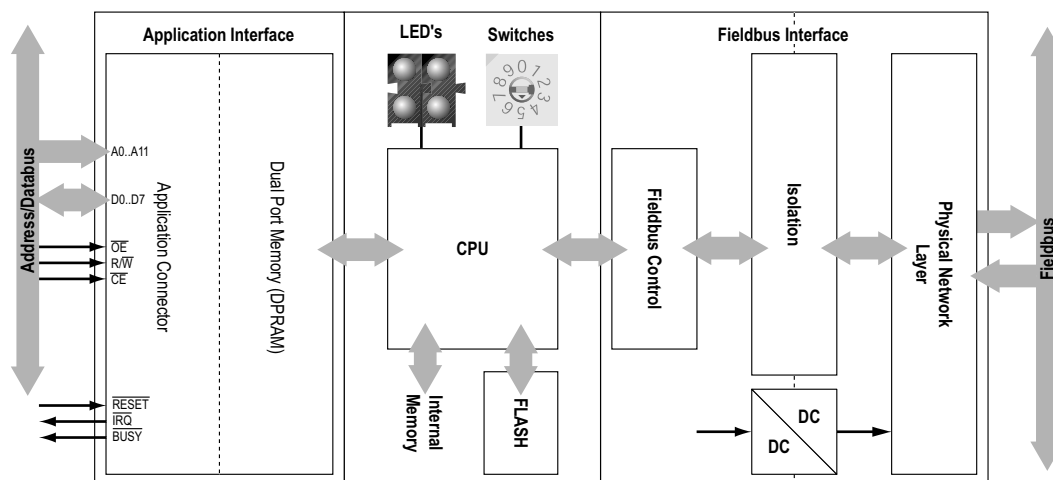
- Interchangeable (Uniform software interface regardless of fieldbus type)
- Slave and Master versions available
- All major fieldbus systems supported through a common application interface
- On board CPU relieves host system from time consuming network related tasks
- Pre-certified for all fieldbus networks (where applicable)¹
- Mailbox interface
- 2KB Dual Port Ram (DPRAM)² architecture
- Up to 2048 bytes of Input / Output data²
- On board configuration switches (where applicable)
- On board LED indications
- Galvanically isolated fieldbus interface (where applicable)
- CE, UL & cUL certified

1. See "Fieldbus Certification" on page 81.

2. Some Anybus-S versions offer more DPRAM and I/O data. For more information, see "Extended Memory Mode (4K DPRAM)" on page 73

1.2 Internals

Below is a schematic overview of a typical Anybus-S module; the application interface, the internal data path, and the fieldbus interface.



Application Interface

From an external point of view, the application interface is a common 8 bit parallel slave port interface that can easily be incorporated into any microprocessor based system that has an address/data type bus. Additionally, the application interface also features a reset pin, a busy signal, and an interrupt request signal.

Fieldbus Interface

The fieldbus interface of an Anybus-S module is galvanically isolated, and is designed according to each fieldbus standard. The fieldbus protocol is handled entirely by the Anybus-S and requires no interaction by the application. However, to utilize the full potential of the fieldbus, additional fieldbus specific support is included in all Anybus-S modules. It is then up to the application to exploit these features.

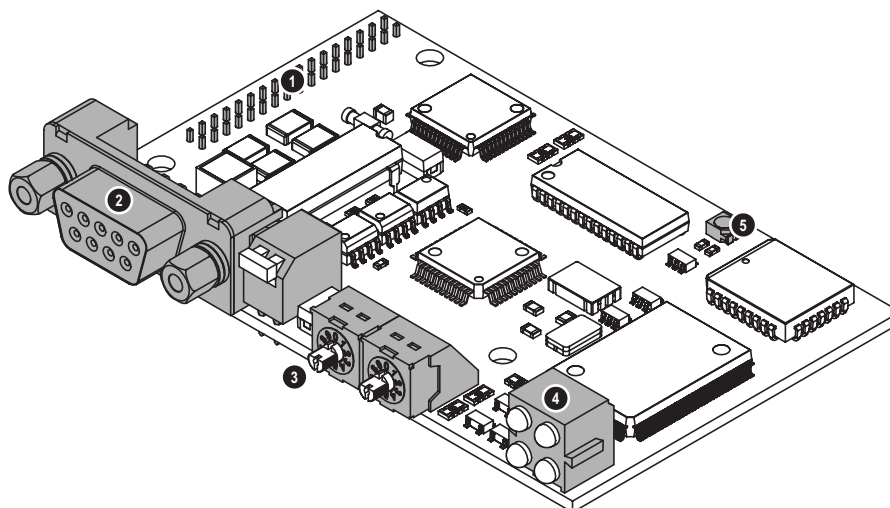
The Anybus-S is tested standalone and found to comply with each fieldbus standard. For more information, see "Fieldbus Certification" on page 81.

Data Exchange

Internally, the application interface is based on a dual port memory (DPRAM) architecture. This enables the application to exchange data with the Anybus-S module via a shared memory area. Basically, in order to exchange data on the fieldbus, all the application has to do is to read/write data from/to this area. The data is then forwarded from/to the fieldbus by the on board CPU, i.e. all fieldbus activity is handled completely by the Anybus-S and generally requires no interaction by the host application.

1.3 External View

The figure below shows a typical Anybus-S module. For more information about the mechanical aspects of the Anybus-S, see “Mechanical Aspects” on page 70.



1. Application Connector

The Anybus-S is accessed through a 34-pin connector (2mm strip header). This connector features various control signals, address/databus signals and power supply. For more information, see “Application Connector” on page 12 and Appendix “Application Connector” on page 71.

2. Fieldbus Connector(s)

The Anybus-S provides fieldbus connectors according to each fieldbus specification.¹

3. Configuration Switches

Some Anybus-S modules features on board configuration switches for fieldbus settings such as baud rate, node address, fieldbus termination etc.

4. Fieldbus Status Indication LED's

All Anybus-S modules features LED indications according to the fieldbus standard. For more information, see “Fieldbus Status Indicators” on page 64 and Appendix “Fieldbus Status Indication LED's” on page 72.

5. Anybus-S Watchdog Led

For more information, see “Anybus-S Watchdog LED” on page 64.

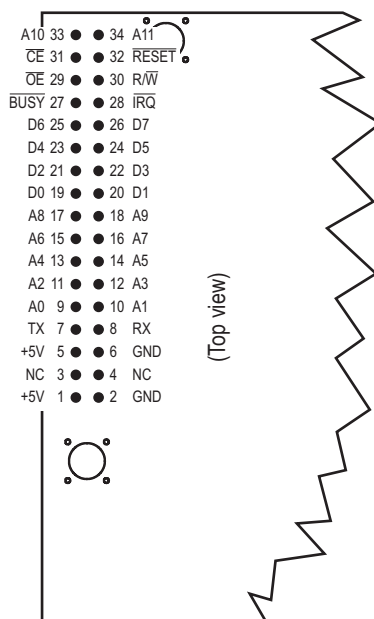
1. See “Fieldbus Certification” on page 81

2. Application Connector

The Anybus-S application connector features a parallel slave port type interface. Implementing this type of interface is comparable to implementing an 8 bit wide SRAM. This makes it easy to incorporate the module directly on the host application address/databus.

The application connector also features an asynchronous serial interface. Generally, this interface is used for firmware upgrades etc., but in some cases it may be used to enable external configuration / monitoring purposes.

For mechanical details, measurements etc. see “Application Connector” on page 71. For further information about signal levels, power requirements etc. see “Electrical Specification” on page 78.



2.1 Connector Pinout

Note: All signals are TTL level unless otherwise stated.

Pin	Name	Description	Direction	Note
1	+5V	VCC	Input	Power Supply, bus interface. See "Power Supply Requirements" on page 78.
2	GND	Ground	-	
3, 4	NC	Isolation distance	-	(Not connected)
5	+5V	VCC	Input	Power Supply, module electronics. See "Power Supply Requirements" on page 78.
6	GND	Ground	-	
7	TxD	Transmit Data ^a	Output	Asynchronous serial interface transmit ^a .
8	RxD	Receive Data ^a .	Input	Asynchronous serial interface receive ^a .
9 - 12	A ₀ - A ₃	Address Inputs	Input	Address lines 0 ... 3
13	A ₄	Address Inputs ^a .	Input	Address line 4 ^a .
14 - 18	A ₅ - A ₉	Address Inputs	Input	Address lines 5 ... 9
19 - 26	D ₀ - D ₇	Data Input / Output	Bidirectional	Databus, bits 0 ... 7
27	$\overline{\text{BUSY}}$	Busy Signal ^a .	Output	Active low open collector output ^a .
28	$\overline{\text{IRQ}}$	Interrupt Request ^a .	Output	Active low open collector output ^a .
29	$\overline{\text{OE}}$	Output Enable	Input	Active low input
30	R/ $\overline{\text{W}}$	Read/Write	Input	Active low input
31	$\overline{\text{CE}}$	Chip Enable ^a .	Input	Active low input ^a .
32	$\overline{\text{RESET}}$	Reset	Input	Active low input. Internally pulled up with 35 - 75k Ω .
33	A ₁₀	Address Input ^a .	Input	Address line 10 ^a .
34	A ₁₁	Address Input ^b	Input	Address line 11 ^b .

a. Internally pulled up with 10k Ω .

b. This signal is used on some Anybus modules to accommodate a larger dual port memory. On those modules, this pin is internally pulled up with 10k Ω . For more information, see "Extended Memory Mode (4K DPRAM)" on page 73. Note that the use of this pin is optional. If not used, it must be left unconnected or pulled to VCC.

Note: Since the first release of the Anybus-S, several minor changes have been made to the application interface. The information in the table above applies only to the most recent Anybus-S versions. For more information, see "Application Interface Hardware Deviances" on page 75.

2.2 Control Signals

Address Inputs (A₀ ... A₁₁)

Address input pins. Selects the target location in the dual port memory. A₀ contains the least significant bit, A₁₁ contains the most significant bit. A₄, A₁₀ and A₁₁ is internally pulled up with 10k Ω .

The use of A₁₁ is optional, however it is recommended to implement it on the application as it is used on some Anybus modules for extended memory features. If not implemented, it must be left unconnected or pulled to VCC. For more information, see "Extended Memory Mode (4K DPRAM)" on page 73. See also "Application Interface Hardware Deviances" on page 75.

Data Input / Output ($D_0 \dots D_7$)

Data output pins during read operations, or data input pins during write operations. D_0 is the least significant bit, D_7 is the most significant.

The target memory location is specified on the Address Inputs ($A_0 \dots A_{11}$).

Busy Signal ($\overline{\text{BUSY}}$)

Active low open collector output, internally pulled up with $10\text{k}\Omega$. When low, this pin indicates that the desired address is currently in use by the Anybus module, and can be used to insert wait states to stall the current operation until the module is ready.

Interrupt Request ($\overline{\text{IRQ}}$)

Active low open collector output, internally pulled up with $10\text{k}\Omega$. When low, this pin indicates that new information is available in the Anybus Indication Register (7FFh). It is strongly recommended to implement this signal on the host application.

Output Enable ($\overline{\text{OE}}$)

Enables data output on $D_0 \dots D_7$ when low.

Read/Write ($\overline{\text{R}/\text{W}}$)

Enables data input on $D_0 \dots D_7$ when low. Internally pulled up with $10\text{k}\Omega$.

Chip Enable ($\overline{\text{CE}}$)

Active low input (though pulled up on most modules); enables communication with the application interface. $\overline{\text{CE}}$ must only be active during access of the DPRAM. Internally pulled up with $10\text{k}\Omega$ unless otherwise stated in section 'Application Interface Hardware Deviances'.

Reset ($\overline{\text{RESET}}$)

If low, a system reset is initiated.

Internally pulled up with $10\text{k} - 75\text{k}\Omega$ and decoupled to ground with a $10 - 100\text{nF}$ capacitor.

2.3 Asynchronous Serial Interface

These pins are generally used for firmware upgrades etc., see “Firmware Upgrade” on page 65.

For signal characteristics etc., see “Signal Characteristics” on page 79.

Transmit Data (TxD)

Asynchronous serial data transmit signal. Internally pulled up with 10k Ω . Anybus modules with 3,3 to 5V conversion of the Tx signal does not have the 10k resistor. The signal is driven high or low by the buffer circuit instead. See also “Application Interface Hardware Deviances” on page 75.

Receive Data (RxD)

Asynchronous serial data receive signal. Internally pulled up with 10k Ω .

3. Memory Map

The dual port memory is subdivided into several smaller areas based on their usage, see memory map below.

Address:	Area:	Access:	Notes:
000h - 1FFh	Input Data Area	R/W	See "Fieldbus Data Exchange" on page 36
200h - 3FFh	Output Data Area	RO	See "Fieldbus Data Exchange" on page 36
400h - 51Fh	Mailbox Input Area	R/W	See "Mailbox Interface" on page 39
520h - 63Fh	Mailbox Output Area	RO	See "Mailbox Interface" on page 39
640h - 7BFh	Fieldbus Specific Area	-	(Consult separate fieldbus appendix)
7C0h - 7FDh	Control Register Area	R/W	See "Control Register Area" on page 17
7FEh - 7FFh	Handshake Registers	R/W	See "Handshaking & Indication Registers" on page 25



These areas must be allocated before access. See "Handshaking & Indication Registers" on page 25.



These areas can be accessed directly.

Note: Implementing A11 in the application will affect the memory map. See "Extended Memory Mode (4K DPRAM)" on page 73 for further information.

4. Control Register Area

This area contains information about the Anybus module; revision, initialization parameters, fieldbus type and status etc. This area also contains registers for Watchdog handling and Event Notification handling.

Address:	Register:	Access:	Notes:
7C0h - 7C1h	Bootloader Version	RO	
7C2h - 7C3h	Application Interface Software Version ^a	RO	
7C4h - 7C5h	Fieldbus software version ^a	RO	
7C6h - 7C9h	Module Serial Number	RO	Unique serial number
7CAh - 7CBh	Vendor ID	RO	Manufacturer ID number (HMS, other)
7CCh - 7CDh	Fieldbus Type	RO	Fieldbus type identifier
7CEh - 7CFh	Module Software Version	RO	Software revision
7D0h - 7D1h	(reserved)	-	
7D2h - 7D3h	Watchdog Counter Input	R/W	Application controlled Watchdog counter
7D4h - 7D5h	Watchdog Counter Output	RO	Counter, incremented each 1ms
7D6h - 7D9h	(reserved)	-	
7DAh - 7DDh	LED Status	RO	Current status of each fieldbus status indicator
7DEh - 7DFh	(reserved)	-	
7E0h - 7E1h	Module Type	RO	Module type, master, slave, other.
7E2h - 7E3h	Module Status	RO	Bit information; freeze, clear etc.
7E4h - 7EBh	Changed Data Field	RO	Bit field, indicating changes in the Output Data Area in the Dual Port Memory
7ECh - 7EDh	Event Notification Cause	R/W	Event cause register
7EEh - 7EFh	Event Notification Source	RO	Configuration register for Event Notification
7F0h - 7F1h	Input I/O Length	RO	Input I/O size
7F2h - 7F3h	Input DPRAM Length	RO	Number of input I/O bytes in dual port memory
7F4h - 7F5h	Input Total Length	RO	Total Input Data size
7F6h - 7F7h	Output I/O Length	RO	Output I/O size
7F8h - 7F9h	Output DPRAM Length	RO	Number of output I/O bytes in dual port memory
7FAh - 7FBh	Output Total Length	RO	Total Output Data size
7FCh - 7FDh	(reserved)	-	

a. On modules with application interface versions prior to 2.00, this register is reserved and should be zero.

Note: Generally, the Control Register Area must be allocated by the application before access. However, during module initialization, it is allowed to read static data such as software revision, fieldbus type, module type etc. without handshaking.

4.1 Registers

Module Bootloader Version (7C0h - 7C1h, RO)

This register specifies the revision of the boot loader firmware within the module.

	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0	
7C0h	Major revision (BCD coded)								Minor revision (BCD coded)								7C1h

Application Interface Software Version (7C2h - 7C3h, RO)

This register specifies the revision of the application interface firmware in the module.

	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0	
7C2h	Major revision (BCD coded)								Minor revision (BCD coded)								7C3h

Note: On modules with application interface versions prior to 2.00, this register is reserved and set to 0.

Fieldbus Software Version (7C4h - 7C5h, RO)

This register specifies the revision of the fieldbus control firmware in the module.

	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0	
7C4h	Major revision (BCD coded)								Minor revision (BCD coded)								7C5h

Note: On modules with application interface versions prior to 2.00, this register is reserved and set to 0.

Module Serial Number (7C6h - 7C9h, RO)

This register contains a unique 32 bit serial number.

Vendor ID (7CAh - 7CBh, RO)

This register indicates the manufacturer of the module.

ID #	Vendor
0000h	(not used)
0001h	HMS
0002h - FFFFh	Reserved for OEM customers

Fieldbus Type (7CCh - 7CDh, RO)

This register indicates the type fieldbus interface that is featured on the module.

Type #	Fieldbus
0001h	PROFIBUS-DP
0005h	PROFIBUS-DPV1
0010h	Interbus-S
0011h	Interbus 2Mbps (Copper & Fibre Optic)
0015h	LonWorks
0020h	CANopen
0025h	DeviceNet
0035h	FIP IO
0040h	Modbus Plus
0045h	Modbus RTU
0065h	ControlNet
0082h	Ethernet (Modbus/TCP + IT)
0083h	Ethernet (EtherNet/IP + Modbus/TCP + IT)
0084h	PROFINET
0086h	FL-net
0087h	EtherCAT
0089h	PROFINET IRT
0090h	CC-Link
0091h	AS-Interface
0093h	Ethernet (Modbus/TCP + IT) 2-port
0094h	Ethernet (EtherNet/IP + Modbus/TCP + IT) 2-port
009Dh	PROFINET IRT FO

Module Software Version (7CEh - 7CFh, RO)

This register specifies the revision of the system firmware within the module.

	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0	
7CEh	Major revision (BCD coded)								Minor revision (BCD coded)								7CFh

Watchdog Counter Input (7D2h - 7D3h, R/W)

This register is used to indicate to the fieldbus that the application is working properly. To accomplish this, the application should read the contents of the Watchdog Counter Output and write it to this register.

If the difference between these registers exceeds the value specified in the Anybus_Init command during module initialization, the module will indicate that the application is not working properly to the fieldbus. This feature can be disabled during initialization by setting the difference value to zero.

	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0	
7D2h	Counter (high byte)								Counter (low byte)								7D3h

Note: The implementation and behavior of this bit depends on the fieldbus type. Consult each separate fieldbus appendix for more information. See also “Deviances” on page 74.

Watchdog Counter Output (7D4h - 7D5h, RO)

The Anybus-S firmware features an internal counter that is incremented each millisecond. This internal counter is continuously written to this register to indicate to the application that the Anybus module is working properly.

The maximum refresh time of this register is 50ms, i.e. the value can be updated by as much as 50 each refresh cycle.

	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0	
7D4h	Counter (high byte)								Counter (low byte)								7D5h

LED Status (7DAh - 7DDh, RO)

These registers contains the current status of each fieldbus status indicator LED, 1 byte per led.

	b7	b6	b5	b4	b3	b2	b1	b0	
7DAh	Led 1 (Top left)								
7DBh	Led 2 (Top right)								
7DCh	Led 4 (Bottom left)								
7DDh	Led 3 (Bottom right)								

Value	Description
00h	LED off or not used by the module
01h	LED green ^a
02h	LED red ^a

a. Due to the requirements of certain fieldbus systems, some versions of the Anybus-S may use other colours. Consult each fieldbus appendix for more information,

Module Type (7E0h - 7E1h, RO)

This register indicates the type of the currently used module.

Type #	Module Type
0001h	Anybus DT (Obsolete)
0101h	Anybus-S (a.k.a. Anybus-S Slave)
0102h	Anybus-S Drive Profile
0201h	Anybus-M (a.k.a Anybus-S Master)

Module Status (7E2h - 7E3h, RO)

This register contains various status bits. Most bits in this register corresponds to settings made in the ‘Anybus Init’ mailbox command., see “Anybus Initialization (Anybus_INIT)” on page 46.

	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0	
7E2h	(reserved)					APRS	CD	APFC	(reserved)			RDR	FBSPU	FBS	FBFC	FBRS	7E3h

- **APRS¹ (Application Run/Stop)**

This bit indicates if the difference between the Watchdog Counter Output and Watchdog Counter Input has exceeded the Watchdog Timeout Value specified in Anybus_INIT.

0: Application has stopped (Watchdog Timeout Value exceeded)

1: Application is running

- **CD (Changed Data Field Status)**

0: Changed Data Field disabled

1: Changed Data Field enabled

- **APFC¹ (Application Stopped Freeze/Clear)**

0: Input data is cleared if application has stopped

1: Input data is frozen if application has stopped

- **RDR¹ (Fieldbus Reset Device Request Notification)**

0: No fieldbus reset device request received (or the feature is not used)

1: Fieldbus reset device request received

- **FBS¹, FBSPU¹ & FBFC**

Bit value			Fieldbus Offline Action		Notes
FBSPU ¹	FBS ¹	FBFC	Output I/O Data	Parameter Data	
0	0	0	Clear	Clear	All data is cleared when the fieldbus goes off line.
		1	Freeze	Freeze	All data is frozen in it's current state when the fieldbus goes off line.
	1	0	Set	Set	All data is set when the fieldbus goes offline.
		1	(reserved)	(reserved)	-
1	0	0	Clear	Update	On some fieldbus systems, parameter data may still be updated via the fieldbus although the I/O data exchange is not running.
		1	Freeze	Update	
	1	0	Set	Update	Consult each separate fieldbus appendix for further information.
		1	(reserved)	(reserved)	

- **FBRS (Fieldbus On / Off Line)**

1: Fieldbus online

0: Fieldbus offline

1. The implementation and behavior of this bit depends on the fieldbus type. Consult each separate fieldbus appendix for more information. See also “Deviances” on page 74.

Changed Data Field (7E4h - 7EBh, R/W)

These registers form bit fields by which the application may determine what parts of the Output Data Area contains changed data. Each bit in these registers represents 8 bytes in the Output Data Area.

Note 1: The use of these registers will reduce the overall performance of the module.

Note 2: This register is not implemented in some versions of the Anybus-M (a.k.a. Anybus-S Master).

	b7	b6	b5	b4	b3	b2	b1	b0
7E4h	56-63	48-55	40-47	32-39	24-31	16-23	8-15	0-7
7E5h	120-127	112-119	104-111	96-103	88-95	80-87	72-79	64-71
7E6h	184-191	176-183	168-175	160-167	152-159	144-151	136-143	128-135
7E7h	248-255	240-247	232-239	224-231	216-223	208-215	200-207	192-199
7E8h	312-319	304-311	296-303	288-295	280-287	272-279	264-271	256-263
7E9h	376-383	368-375	360-367	352-359	344-351	336-343	328-335	320-327
7EAh	440-447	432-439	424-431	416-423	408-415	400-407	392-399	384-391
7EBh	504-511	496-503	488-495	480-487	472-479	464-471	456-463	448-455

Event Notification Cause (7ECh - 7EDh, R/W)

This register indicates the source of an Event Notification. The bits in this register are edge triggered, i.e. the bits are set by the module when the event occurs. Default value is 0. The application should clear the corresponding bit when the Event Notification has been handled.

For more information regarding interrupts and Event Notification, see “Interrupts” on page 34 and “Event Notification (Software Interrupt)” on page 35.

	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
7ECh	(reserved)												RST	FBON	FBOF	DC
																7EDh

- **RST¹**
 - 0: -
 - 1: A reset request from the fieldbus has been issued.
- **FBON**
 - 0: -
 - 1: Fieldbus has gone on line
- **FBOF**
 - 0: -
 - 1: Fieldbus has gone off line
- **DC**
 - 0: -
 - 1: Data has been updated, see “Changed Data Field (7E4h - 7EBh, R/W)” on page 22.

1. The implementation and behavior of this bit depends on the fieldbus type. Consult each separate fieldbus appendix for more information. See also “Deviances” on page 74.

Event Notification Source (7EEh - 7EFh, RO)

This register indicates which events that will trigger an Event Notification. The contents of this register is determined during module initialization in the mailbox command ‘Anybus Init’.

For more information regarding interrupts and Event Notification, see “Interrupts” on page 34 and “Event Notification (Software Interrupt)” on page 35.

	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0	
7EEh	(reserved)												RST	FBON	FBOF	DC	7EFh

- **RST¹**
 - 0: -
 - 1: An Event Notification will be issued each time “reset request” has been issued from the fieldbus
- **FBON**
 - 0: -
 - 1: An Event Notification will be issued when the fieldbus goes on line
- **FBOF**
 - 0: -
 - 1: An Event Notification will be issued when the fieldbus goes off line
- **DC**
 - 0: -
 - 1: An Event Notification will be issued each time the data in the Output Area has changed.
Note that this requires that the Changed Data Field has been enabled during module initialization.

Input I/O Length (7F0h - 7F1h, RO)

This holds the Input I/O Length specified during module initialization.

	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0	
7F0h	High byte								Low byte								7F1h

Input DPRAM Length (7F2h - 7F3h, RO)

This holds the Input DPRAM Length specified during module initialization.

	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0	
7F2h	High byte								Low byte								7F3h

1. The implementation and behavior of this bit depends on the fieldbus type. Consult each separate fieldbus appendix for more information. See also “Deviances” on page 74.

Input Total Length (7F4h - 7F5h, RO)

This holds the Input Total Length specified during module initialization.

	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0	
7F4h	High byte								Low byte								7F5h

Output I/O Length (7F6h - 7F7h, RO)

This holds the Output I/O Length specified during module initialization.

	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0	
7F6h	High byte								Low byte								7F7h

Output DPRAM Length (7F8h - 7F9h, RO)

This holds the Output DPRAM Length specified during module initialization.

	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0	
7F8h	High byte								Low byte								7F9h

Output Total Length (7FAh - 7FBh, RO)

This holds the Output Total Length specified during module initialization.

	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0	
7FAh	High byte								Low byte								7FBh

5. Handshaking & Indication Registers

Memory locations 7FEh and 7FFh holds two important registers; the Application Indication Register and the Anybus Indication Register. These registers serves three main purposes:

- **Area allocation and de-allocation**

This procedure is mandatory when accessing the Input-, Output-, Fieldbus Specific- and Control Register Area. For more information, see “Area Allocation/De-allocation” on page 29.

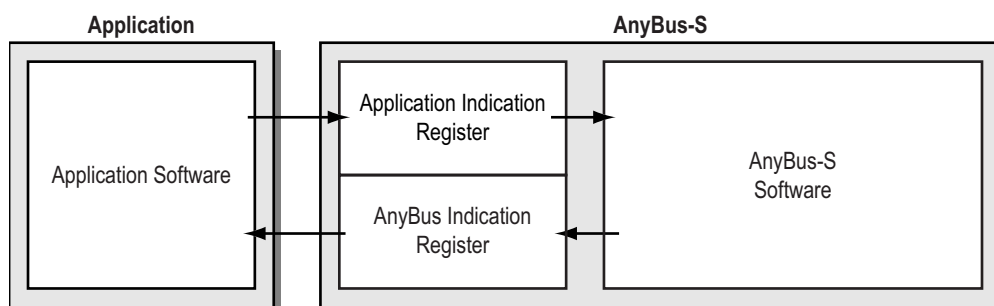
- **Event Notification**

See “Event Notification (Software Interrupt)” on page 35.

- **Sending & Receiving Mailbox Messages**

See “Mailbox Interface” on page 39.

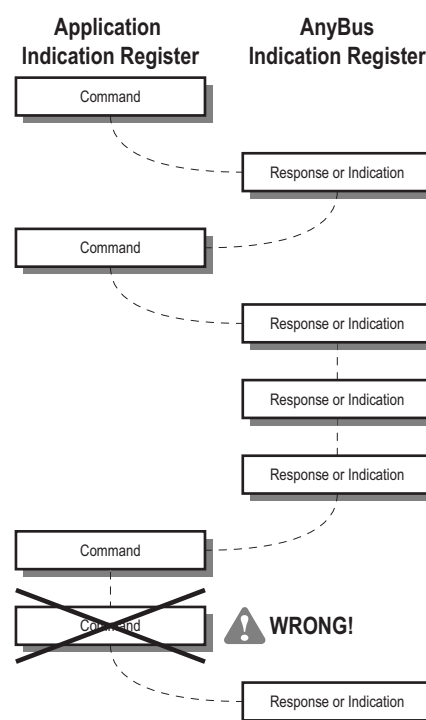
Generally, the Application Indication Register is used to instruct (command) the module to perform a specific low level action. The Anybus Indication Register contains responses to previously issued commands and indicates the overall status of the module.



Although these registers are often used in a command-response like fashion, the registers are independent from each other and should be treated accordingly. However, the application must not issue a new command until the module has responded to the previous one., see figure on the right.

Each time a response or indication is sent by the module, the UPDATED bit (bit 3 of the Anybus Indication Register) is toggled and a hardware interrupt is triggered. The application can then examine the Anybus Indication Register to detect the cause of the hardware interrupt.

If the hardware interrupt feature is not implemented, the application has to cyclically poll the Anybus Indication Register in order to detect any changes. It is however strongly recommended to utilize the interrupt feature, as polling suffers from unnecessary overhead and is generally harder to implement.



5.1 Application Indication Register (7FEh, R/W)

This register is used to perform the following tasks:

- Area allocation and de-allocation (a.k.a. Request / Release)
- Acknowledge Events (Event Notification)
- Send / Receive mailbox messages

The register consists of a bit field with the following layout:

b7	b6	b5	b4	b3	b2	b1	b0
AP_M _{IN}	AP_M _{OUT}	AP_EVNT	ACTION	LOCK	AP_IN	AP_OUT	AP_FBCTRL

Bit	Function	Description	Default
AP_M _{IN}	Mailbox Notification	Used to send a mailbox message. See "Mailbox Notification Bits" on page 40.	0
AP_M _{OUT}		Used to acknowledge a received mailbox message. See "Mailbox Notification Bits" on page 40.	0
AP_EVNT	Event Acknowledge	This bit should be toggled to acknowledge that an Event Notification event has been handled. See "Event Notification (Software Interrupt)" on page 35.	0
ACTION	Area Request/Release. These bits are used to request or release one or several areas within the dual port memory.	This bit indicates if the current action is a request or release: 1: Request 0: Release	0
LOCK		This bit indicates if the action is locked or unlocked. 1: Locked 0: Unlocked	0
AP_IN		This bit represents the Input Data Area 1: Action is affective for this area 0: Action is not affective for this area	0
AP_OUT	For more information, see "Area Allocation/De-allocation" on page 29.	This bit represents the Output Data Area 1: Action is affective for this area 0: Action is not affective for this area	0
AP_FBCTRL		This bit represents the Fieldbus Specific Area and the Control Registers 1: Action is affective for this area 0: Action is not affective for this area	0

Note 1: It is recommended not to access this register cyclically. It should only be used to respond to incoming events and to make requests.

Note 2: When accessing this register, it is important to modify all bits related to the desired operation using a single memory access, otherwise the operation might be misinterpreted as several operations by the module.

5.2 Anybus Indication Register (7FFh, RO)

This register contains responses to previously issued commands and indicates the status of different functions/areas in the module. The following information can be extracted from this register:

- Ownership of the different areas within the dual port memory
- Event Notification status
- Mailbox Input & Output status
- initialization status

Each change in this register will trigger a hardware interrupt. The application can then examine the contents of the register to detect the cause of the interrupt.

It is also possible to cyclically poll the contents of this register to detect any changes, however it is strongly recommended to utilize the interrupt feature, as polling suffers from unnecessary overhead and is generally harder to implement. (For more information about interrupts, see “Hardware Interrupt (IRQ)” on page 34).

b7	b6	b5	b4	b3	b2	b1	b0
AB_M _{IN}	AB_M _{OUT}	AB_EVNT	INIT	UPDATED	AB_IN	AB_OUT	AB_FBCTRL

Bit	Function	Description	Default
AB_M _{IN}	Mailbox Notification	The module toggles this bit to acknowledge that it has received a mailbox message. See “Mailbox Notification Bits” on page 40.	0
AB_M _{OUT}		The module toggles this bit to indicate that a new message is available in the Mailbox Output area. See “Mailbox Notification Bits” on page 40.	0
AB_EVNT	Event Notification	The module toggles this bit when a new Event Notification has occurred	0
INIT	Module initialised	This bit indicates if the module has been initialised. 0: Module not initialised 1: Module initialised ^a	0
UPDATED	Register updated	This bit is toggled by the module each time the contents of this register has been updated.	0
AB_IN	Area ownership. These bits indicates the owner of each area within the dual port memory.	This bit represents the Input Data Area 1: Area owned by application 0: Area owned by Anybus	0
AB_OUT		This bit represents the Output Data Area 1: Area owned by application 0: Area owned by Anybus	0
AB_FBCTRL		This bit represents the Fieldbus Specific Area and the Control Registers 1: Area owned by application 0: Area owned by Anybus	0

a. Please note that due to the nature of certain fieldbus systems, this does not necessarily mean that the fieldbus is fully initialised and exchanging data. Consult each separate fieldbus appendix for further information.

Note: This register is read only. Do not attempt to write to this register as doing so may produce unpredictable results.

5.3 Collisions

As mentioned earlier, the application interface is based on a dual port memory architecture, where both sides are able to access the same memory location simultaneously.

An area allocation scheme (see “Area Allocation/De-allocation” on page 29) is used in order to prevent collisions during runtime, however this does not cover the event of a collision occurring in the Anybus and Application Indication Registers. If this is not handled correctly, the module might not get data written to the Application Indication Register, and data read from the Application Indication Register may be out of date and/or incorrect.

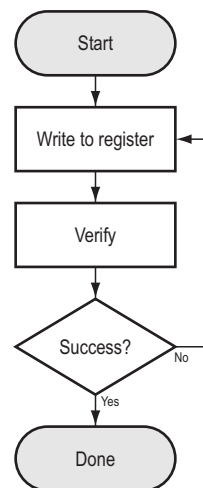
There are three ways of dealing with collisions: (Numbered in order of importance)

1. Multiple Write / Read (Mandatory)

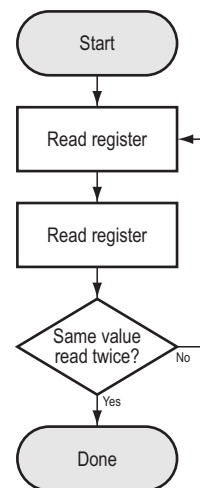
While the $\overline{\text{IRQ}}$ and $\overline{\text{BUSY}}$ signals are used to *prevent* collisions (see below), this is a way of actually *dealing* with them.

- When writing to the Application Indication Register...
... keep writing and verifying until the written data is known to be correct.
- When reading from the Anybus Indication Register...
... keep reading and comparing until two consecutive reads match.

Application Indication Register Access:



AnyBus Indication Register Access:



Note: At first sight, it may appear as if this violates what is stated earlier on page 25 and in Note 1 on page 26, however as the accesses are carried out in rapid sequence the module will interpret them as a single access.

2. Implement the $\overline{\text{IRQ}}$ signal in the application (Optional, but recommended)

This pin is drawn low each time the Anybus Indication Register has been updated. The application can utilize this by accessing this register instantly upon receiving the interrupt, thus avoiding a collision.

3. Implement the $\overline{\text{BUSY}}$ signal in the application (Optional, but recommended)

If this signal goes low, the application should stall the current memory access until the signal goes high again e.g. by inserting wait states, thus avoiding a collision. (This may however not be possible with all architectures).

General Recommendation

Preferably, all three options should be implemented in the application. Either way, option 1 should be considered mandatory, and it is strongly recommended to implement at least one of the other two.

5.4 Area Allocation/De-allocation

As described earlier, the dual port memory is sub divided in several areas based on their function. In order to avoid collisions and to ensure data consistency, the application has to allocate each area before access. If the area is free to use, the module will indicate this to the application by setting the corresponding bit in the Anybus Indication Register. The area is then considered to be “owned” by the application, and can be accessed freely. When finished, the area must be returned, i.e. released, to the Anybus module. The area is then considered to be “owned” by the Anybus module.

This allocation procedure is mandatory when accessing the following areas:

- Input Data Area
- Output Data Area
- Fieldbus Specific Area
- Control Register Area

The application can own an area for a maximum of 1000ms. If this time is exceeded, i.e. the application does not release the area in time, the allocation will be terminated automatically, i.e. the ownership of the area(s) will be handed back to the Anybus module. It is important that the software routines within the application have the capability to recognize this and terminate the access in a safe manner.

5.4.1 Unsynchronized Data Exchange

In it's simplest form, an access sequence towards the module consists of the following steps: (See also figure on the right)

1. Request access to an area

To accomplish this, the application should set the corresponding bit for that area as well as the ACTION bit in the Application Indication Register.

2. Wait for response¹

3. Check response to see if access is granted

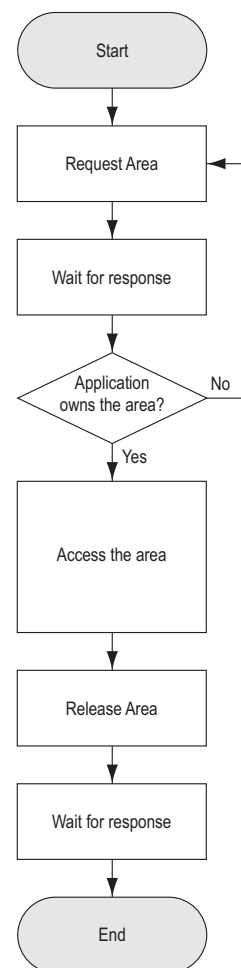
To know if the request was successful, the application should check the ownership of the area in the Anybus Indication Register.

If the desired area is owned by the application, the application is free to access that area.

4. Release the area

To do this, the application should set the corresponding bit for that area as well as clearing the ACTION bit in the Application Indication Register.

5. Wait for response¹



1. This can either be waiting for an interrupt or polling the Anybus Indication Register, depending on how the application is implemented. For more information, see “Anybus Indication Register (7FFh, RO)” on page 27 and “Interrupts” on page 34.

5.4.2 Synchronised Data Exchange

The procedure described earlier will result in an unsynchronised data exchange. However, in some cases it is desirable to synchronise the data exchange between the application and the Anybus-S module.

The LOCK bit in the Application Indication Register is used for this purpose, see table below.

Action	LOCK	Result
Request	0	If the requested area is currently in use, the application will have to repeat the request until access is granted.
	1	If the requested area is currently in use, the module will first send a response indicating that the area is still in use by the Anybus module. However, as soon as the area is free, the ownership of the area will be handed over to the application (i.e. Area owned by Application).
Release	0	The area is released.
	1	The area is released, and is reserved for the Anybus module. The application will not be granted access again before the module has accessed the area.

Locked Request

A locked request ensures that the application will gain access to an area as soon as it is free.

1. Request access to the area (LOCK = 1)

2. Wait for the initial response¹

3. Check the ownership of the area

Area owned by Application - the area is free to use. Proceed with step 6.

Area owned by Anybus - the area is currently in use. Proceed with step 4.

4. Wait for an additional response¹

5. Check the ownership of the area

The ownership of the area is handed over to the application.²

6. Done

-
1. This can either be waiting for an interrupt or polling the Anybus Indication Register, depending on how the application is implemented. For more information, see “Anybus Indication Register (7FFh, RO)” on page 27 and “Interrupts” on page 34.
 2. If multiple areas are requested simultaneously, the module may send up to 3 additional responses, one for each area. In these cases, the sequence of events may be slightly different from what is described above. For more information, see “Requesting/Releasing Multiple Areas Simultaneously” on page 31.

Locked Release

In some cases, it makes no sense to gain access to an area unless the Anybus has accessed it first. For example, there is no gain in polling the Output Data Area cyclically unless the application can be sure that the data has been updated by the Anybus module between each poll.

By using the LOCK bit when releasing an area, the application can reserve the area for the Anybus module, i.e. the application will not gain access to the area until the module has updated its contents.¹

1. **Release the area (LOCK = 1)**

2. **Wait for response²**

3. **Done**

The area is now reserved for the Anybus module, i.e. the application will not gain access to the area until the module has updated its contents.

5.4.3 Requesting/Releasing Multiple Areas Simultaneously

Multiple areas can be requested or released simultaneously. However, when doing this, it is important to monitor the response in the Anybus Indication Register to see which areas that are actually free to use (i.e. it is possible that one or more of the areas is in use at the time of the request). This is pretty straight forward for unlocked requests, however special care has to be taken when performing locked requests for multiple areas.

- **Requesting multiple areas (LOCK = 0)**

The module sends a single response.

- **Requesting multiple areas (LOCK = 1)**

The module may in theory send up to 4 responses depending on the situation:

- The initial response.
- Up to three additional responses will be sent as the ownership of the requested areas are handed over to the application when an area is free to use.

- **Releasing multiple areas (LOCK = 0)**

The specified areas are released instantly. The module sends a single response to acknowledge the release. “Locked Release Behavior” on page 74

- **Releasing multiple areas (LOCK = 1)**

The specified areas are released instantly. The module sends a single response to acknowledge the release. The areas are then reserved for the Anybus module, i.e. the application cannot gain ownership of them until the Anybus has updated their contents.

-
1. Please note that some modules update the Output Data Area only when there are changes to the output data. This may cause the Output Data Area to stay locked, until an external event has caused changes in the data. For further information see “Locked Release Behavior” on page 74
 2. This can either be waiting for an interrupt or polling the Anybus Indication Register, depending on how the application is implemented. For more information, see “Anybus Indication Register (7FFh, RO)” on page 27 and “Interrupts” on page 34.

5.4.4 Application Example, Cyclic Access Method

This example describes one method for an application that requires cyclic access to the DPRAM input and output data areas.

Background Information

To better understand the example, here follows a description of what happens inside the module.

When the application releases the DPRAM input data area (data to fieldbus), it triggers the module to take control of that area. The area will be owned by the module until it has finished its tasks. How long time this will take varies according to the configuration. Once finished, the module will not require the area until the next time the area is released by the application. The application can thus, immediately after it has released the area (with a locked release), perform a locked request of the area. It will then gain access to the area at once when the module releases it the next time. While waiting, it can execute other tasks, as long as the total time does not exceed 1000 ms.

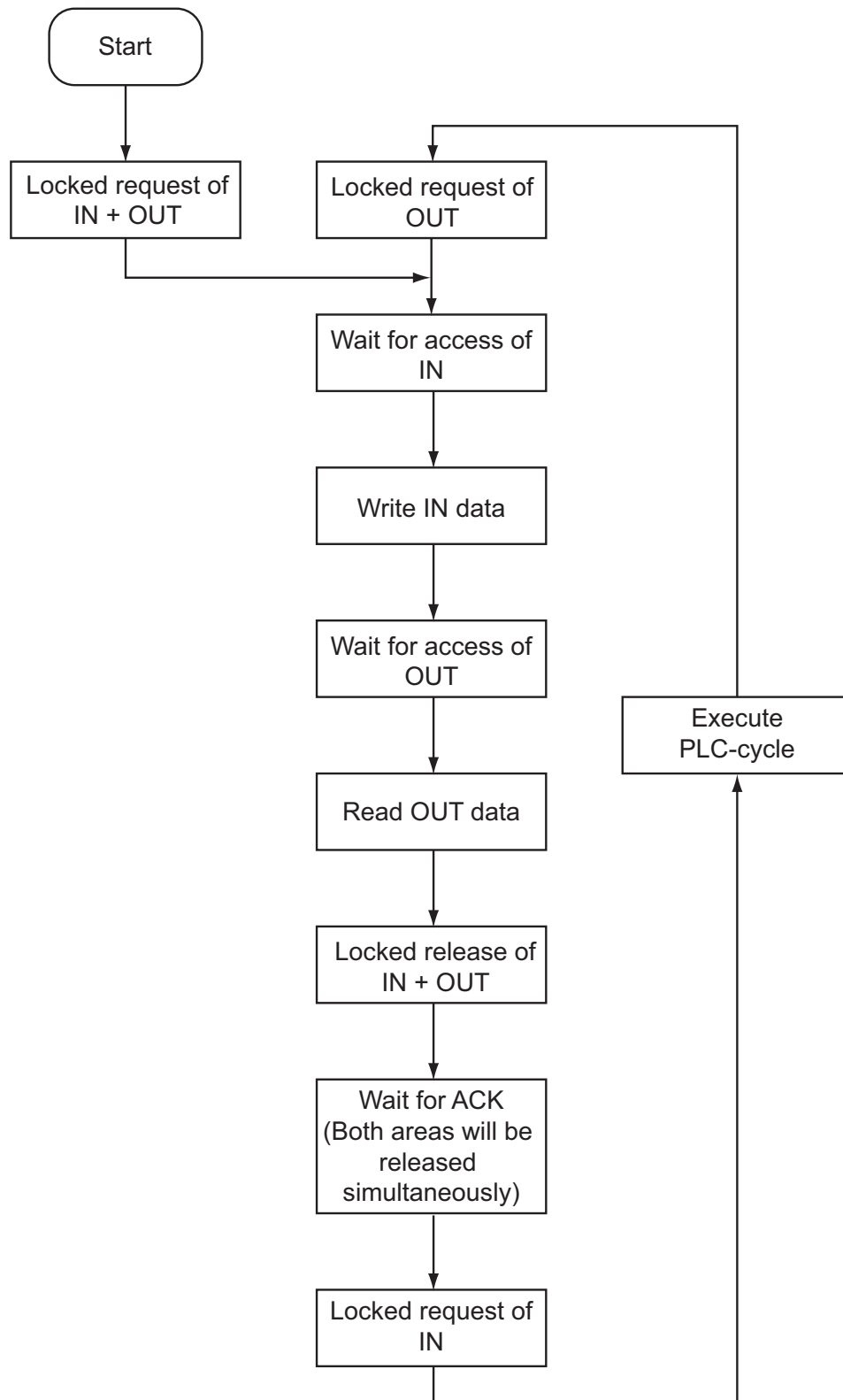
An internal OUT I/O data buffer is continuously updated by the module. When data from all slaves has been updated, the module requests access to the DPRAM output data area (data from fieldbus). It copies the data from the buffer to the DPRAM and then releases the output data area. The process is then repeated. The module's access to the DPRAM output data area is thus not related to the application's access of the area. To make sure to read the latest data, the application must perform a request of the DPRAM output data area only when it actually needs to read the data.

Suggested Access Method for a Cyclic Application

1. Locked request of OUT
2. Wait for access of IN (requested in previous cycle)
3. Write IN data
4. Wait for access of OUT¹
5. Read OUT data
6. Locked release of IN + OUT
7. Wait for ACK (both areas will be released simultaneously)
8. Locked request of IN
9. Execute PLC-cycle
10. Repeat from 1

Note: This loop needs to be entered at step number 2 after an initial locked request of the DPRAM input and output data areas, see flowchart next page.

1. Please note that some modules update the Output Data Area only when there are changes to the output data. This may cause the Output Data Area to stay locked, until an external event has caused changes in the data. For further information see "Locked Release Behavior" on page 74



IN : DPRAM input data area (data to fieldbus)

OUT : DPRAM output data area (data from fieldbus)

Cyclic access method flowchart

6. Interrupts

6.1 Hardware Interrupt (IRQ)

The module features an interrupt request pin ($\overline{\text{IRQ}}$, pin #28). If implemented, it can be used to notify the application of any changes in the Anybus Indication Register. Generally, it is recommended to use this feature as it can significantly reduce overhead compared to polling the register cyclically.

The following events will generate a hardware interrupt:

- Event Notification (Software interrupt, see below)
- Mailbox notification (See “Mailbox Notification Bits” on page 40)
- Module initialised (See “Anybus Indication Register (7FFh, RO)” on page 27)
- Startup interrupt (See “Startup Sequence” on page 59)
- Area allocation responses (See “Area Allocation/De-allocation” on page 29)

Once the application has read the contents of the Anybus Indication Register, the interrupt is automatically cleared.

6.2 Event Notification (Software Interrupt)

Event Notification is a mechanism for signalling important events to the application. The following events can generate an Event Notification:

- Fieldbus On/Off line
- Fieldbus reset requests
- Data changed¹

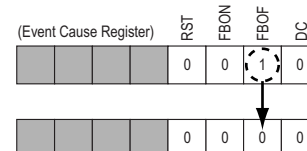
Which of the above events that should cause an Event Notification is configured during module initialization in the Event Notification Config parameter in the 'Anybus Init' mailbox message.

To signal that a new Event has occurred, the module will toggle bit 5 (AB_EVNT) in the Anybus Indication Register. The cause of the event can then be read in the Event Notification Cause Register, see "Event Notification Cause (7ECh - 7EDh, R/W)" on page 22. When the event has been handled, the application should clear the corresponding bits in this register and confirm the event by toggling bit 5 (AP_EVNT) in the Application indication Register.

While an event is unconfirmed by the application, all new events are queued within the module. This eliminates the risk of an event being missed. The module will only toggle AB_EVNT, to indicate a new event, if it is equal to AP_EVNT. Once the application returns a confirmation and toggles AP_EVNT to be equal to AB_EVNT, the module can indicate the next event in the queue. It's important that the application does not toggle bit 5 (AP_EVNT) in the Application Indication Register, unless for confirming an event. If AB_EVNT and AP_EVNT are not equal, the module will be prohibited from indicating a new event.

The following example describes how to check if a new event has occurred, and how to handle it.

1. AB_EVNT | = AP_EVNT?
2. If yes, an event has occurred.(If no, skip the remaining steps)
3. Examine the Event Cause register to find out what caused the event. In this case, the fieldbus has gone off line (FBOF).
4. Clear the FBOF bit in the Event Cause Register²
5. Perform tasks associated with fieldbus off line events².
6. Toggle AP_EVNT in the Application Indication Register to confirm the event.



Note: Event Notification is a software interrupt and should not be confused with the hardware interrupt described earlier. However, as it uses the Anybus Indication Register, it will trigger a hardware interrupt.

1. Data change event notification does not work for I/O data mapped to Internal Memory.
2. This requires that the Fieldbus Specific / Control Register Area is owned by the application.

7. Fieldbus Data Exchange

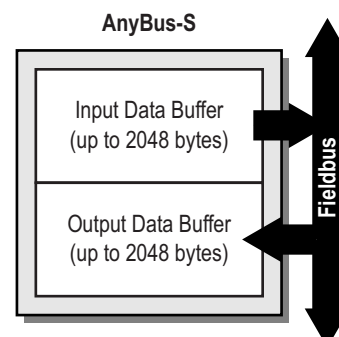
7.1 Basics

The module exchanges data on the fieldbus via two data buffers:

- **Input Data Buffer**
Data written to this buffer will be sent to the fieldbus.
- **Output Data Buffer**
This buffer contains data received from the fieldbus.

Basically, in order to exchange data on the fieldbus, all the application has to do is to read/write data from/to these two buffers.

Note: The size and composition of the data buffers is determined during module initialization. Therefore, the module will not be able to exchange data on the fieldbus unless it has been properly initialised first. For more information, see “Start Up and Initialization” on page 58.



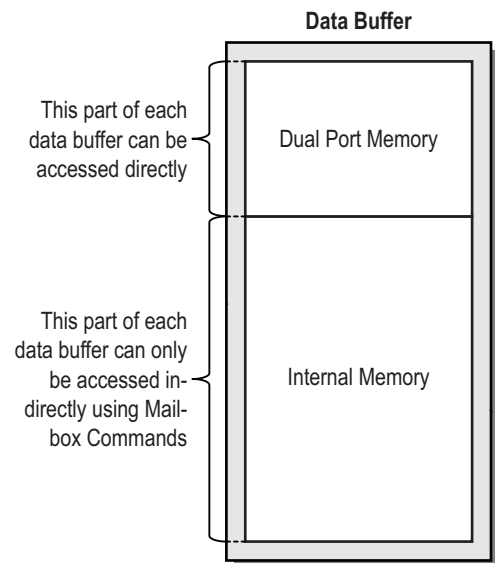
7.2 Dual Port Memory vs. Internal Memory

Each of the two data buffers can have a portion of their data situated in dual port memory. The remainder is located in Internal Memory. The advantage of having data situated in dual port memory is that it can be accessed much faster than data situated in the Internal Memory.

Internal Memory can only be accessed indirectly via mailbox commands, and is thus better suited for less time critical data. Data buffer data that is situated in dual port memory can not be accessed using mailbox messages in parallel.

It is possible to configure how much data that should be reside in dual port memory, and how much that should be located in Internal Memory. The maximum size of each data buffer is 2kbytes, out of which up to 512¹ bytes can be configured to reside in dual port memory.

Note: Data change event notification (software interrupt) can not be used for I/O data mapped to the Internal Memory.



7.3 Data types

Most fieldbus systems makes a distinction between fast cyclical data and slower parameter data. This is reflected in the way data is treated by the Anybus-S module:

- **I/O Data**

This type of data is usually associated with fast fieldbus data (a.k.a. cyclic data).

- **Parameter Data**

This type of data is usually associated with slow fieldbus data (a.k.a. acyclic data).

How this data is treated for each fieldbus type is described in each separate fieldbus appendix.

1. Future Anybus versions may allow a larger amount of data to reside in dual port memory, see “Extended Memory Mode (4K DPRAM)” on page 73.

7.4 Data Composition

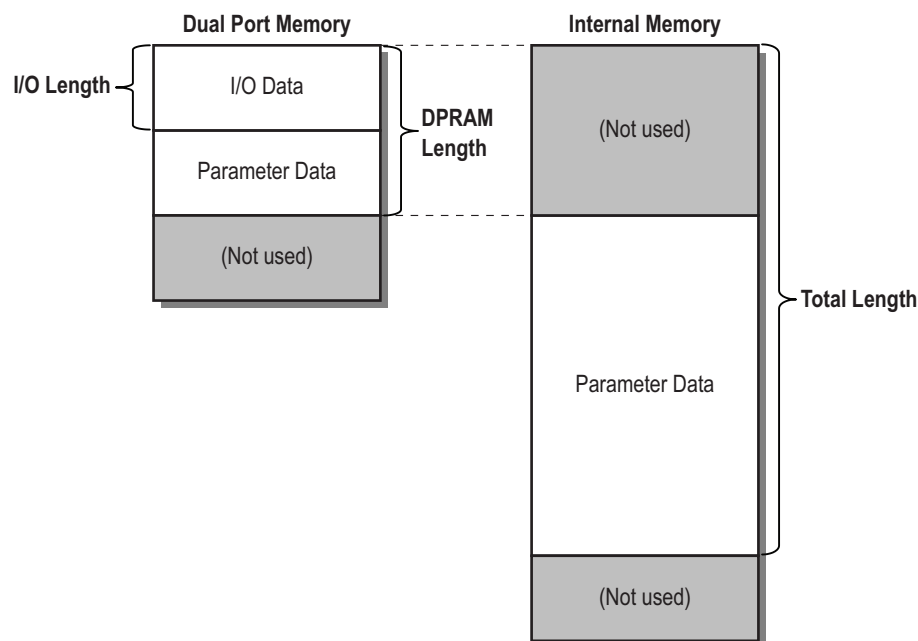
As mentioned earlier, the maximum total size of each data buffer is 2048 bytes. Up to 512¹ of these bytes can reside in dual port memory, the remainder is located in Internal Memory and can only be accessed indirectly using mailbox commands.

The composition of I/O and Parameter Data is determined during the module initialization phase in the mailbox command ‘Anybus Init’, see “Anybus Initialization (Anybus_INIT)” on page 46.

The following parameters must be set for each (Input and Output) data buffer:

- Total Length²**
This parameter defines the total amount of fieldbus data (I/O Data + Parameter Data) for the data buffer. The maximum Total Length is 2048 bytes regardless of DPRAM and I/O Length settings.
- DPRAM Length²**
This parameter defines how much of data that should be located in dual port memory. The DPRAM Length cannot exceed 512¹ bytes.
- I/O Length²**
This parameter defines how much of the data that should be treated as I/O Data. The remaining data will be treated as Parameter Data.

The figure below illustrates the relationship between the parameters described above.



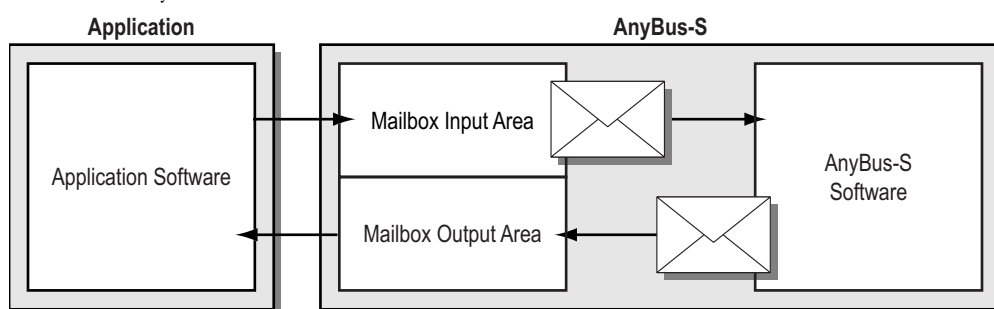
- Future Anybus-S versions may allow a larger amount of data to reside in dual port memory, see “Extended Memory Mode (4K DPRAM)” on page 73.
- The maximum range of these parameters may be limited by the fieldbus. Consult each separate fieldbus appendix for more information.

8. Mailbox Interface

8.1 General

The mailbox interface is a message interface used to instruct the module to perform a specific task, or to request data. It is also used by the module to indicate certain events and to respond to requests sent by the application.

Mailbox communication is handled through the Mailbox Input and Output Areas (See figure below) and generally does not interfere with fieldbus data exchange unless the mailbox message itself is related to fieldbus activity.



The handshaking procedure for the Mailbox Input/Output Areas is slightly different than the one used for the other areas. For more information, see “Mailbox Notification Bits” on page 40.

The mailbox interface supports the following types of communication:

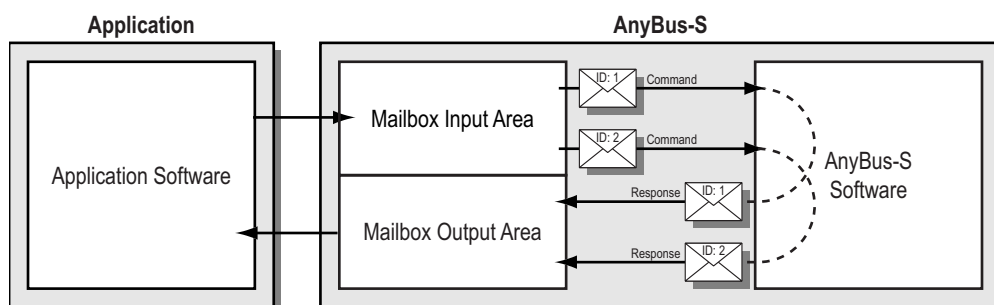
- **Command - Response**

A message is sent by the message initiator, and the message recipient is required to respond. The message initiator can be either the application or the Anybus-S.

- **Indication**

A message is sent by the message initiator, and no response is required. The message initiator can be either the application or the Anybus-S.

The mailbox interface is designed to allow multiple messages to be sent to the module before receiving a response (if applicable). To be able to distinguish which mailbox response that belong to which command, a unique Message ID is used for each message/response, see figure below.



8.2 Message Types

The mailbox messages can be grouped into five categories based on their functionality, see below.

- **Application Messages**
This category includes commands for accessing and controlling internal Anybus-S functions
- **Fieldbus Specific Messages**
This category includes commands for accessing fieldbus specific data and functions. For more information, consult each separate fieldbus appendix.
- **Internal Memory Messages**
This category includes functions for accessing the Internal Memory.
- **Reset Messages**
This category includes functions to effect the module operation.

8.3 Mailbox Notification Bits

The Mailbox Notification bits in the Anybus- and Application Indication Registers are used to control the mailbox interface. Both registers contains bits that are used to send and receive mailbox messages, and to monitor the current mailbox status.

Bit	Function
AP_M _{IN}	Toggle this bit to post a message previously written to the Mailbox Input Area.
AP_M _{OUT}	Toggle this bit to acknowledge that a mailbox message has been read
AB_M _{IN}	This bit is toggled by the Anybus module when it has read a mailbox message
AB_M _{OUT}	This bit is toggled each time a new message is waiting in the Mailbox Output Area.

Before entering a new mailbox message in the Mailbox Input Area, or attempting to read a message from the Mailbox Output Area, it is necessary to know the current status of the mailbox interface. This is done by comparing the corresponding Mailbox Notification bits in the Anybus- and Application Indication Registers, see table below.

Expression	Result	Meaning
AP_M _{IN} = AB_M _{IN}	True	Mailbox Input Area is free
	False	Mailbox Input Area is currently in use
AP_M _{OUT} = AB_M _{OUT}	True	No message available in the Mailbox Output Area
	False	New message available in the Mailbox Output Area

8.3.1 Sending a Mailbox Message

To send a mailbox message to the module, follow the procedure below.

- **Start**
 1. Check if the Mailbox Input Area is free (If not, retry again later)
 2. Write the message to the Mailbox Input Area
 3. Toggle the AP_M_{IN} bit in the Application Indication Register (7FEh)
- **Done**

8.3.2 Receiving a Mailbox Message

To receive a mailbox message, follow the procedure below.

- **Start**
 1. Check if a message is waiting in the Mailbox Output Area (If not, retry again later)
 2. Read the message from the Mailbox Output area
 3. Toggle the AP_M_{OUT} bit in the Application Indication Register (7FEh)
- **Done**

8.4 Mailbox Message Structure

A mailbox message consists of a message header and message data, see below.

Offset:	Contents:	Description:
000h - 01Fh	Message Header (32 bytes)	See "Message Header" on page 42
020h - 11Fh	Message Data (up to 256 bytes)	This section contains the data associated with the mailbox message.

8.5 Message Header

The header consists of a series of 16bit registers that specifies the type of message and the length of the message data.

Offset:	Register:
000h	Message ID
002h	Message Information
004h	Command Number
006h	Data Size
008h	(reserved, set to 0001h)
00Ah	(reserved, set to 0001h)
00Ch	(reserved, set to 0000h)
00Eh	(reserved, set to 0000h)
010h	Extended Word 1
012h	Extended Word 2
014h	Extended Word 3
016h	Extended Word 4
018h	Extended Word 5
01Ah	Extended Word 6
01Ch	Extended Word 7
01Eh	Extended Word 8

Message ID

The Message ID register contains a 16-bit integer identifier for the command. When a response is sent back to the message initiator, the same message ID is used in that message. Message ID:s can be selected arbitrary, but messages currently being processed must all have unique ID's.

Message Information

This register contains bit and code information about the mailbox message. The register is divided into five areas according to the figure below:

b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
ERR	C/R	(reserved)					Error Code								Message Type

Bit / Field	Description	Contents
ERR	This bit indicates if the received command contained any errors.	0:Message OK 1:Error (See also 'Error Code' below)
C/R	This bit specifies whether the message is a command or a response.	0:Response Message 1:Command Message
Error Code	If the ERR bit is set, this field contains additional information about the error.	0h:Invalid Message ID 1h:Invalid Message Type 2h:Invalid Command 3h:Invalid Data Size 4h:Message header malformed (offset 008h) 5h:Message header malformed (offset 00Ah) 6h:Message header malformed (offset 00Ch - 00Dh) 7h: Invalid address 8h:Invalid Response 9h:Flash Config Error Fh:Invalid Other (All other values are reserved)
Message Type	This field specifies the type of the message.	1h:Application Message 2h:Fieldbus Specific Message 3h:Memory Message 5h:Reset Message (All other values are reserved)

Command Number

This register contains a 16 bit command identifier.

Data Size

This register specifies the size of the Message Data in bytes. The maximum Message Data size is 256 bytes.

Extended Words 1 ... 8

These registers are specific for each command. Consult the specification for each command for further information.

9. Mailbox Messages

9.1 Application Messages

General

This category includes commands for accessing and controlling internal Anybus-S functions

Messages in This Category

Message	Abbreviation	Description	Page
Start initialization	START_INIT	Initiates the initialization process.	45
Anybus initialization	Anybus_INIT	Used to set up basic operational parameters.	46
End initialization	END_INIT	Ends the initialization process.	48
Save to FLASH	SAVE_TO_FLASH	Records a mailbox initialization sequence to flash.	49
Load from FLASH	LOAD_FROM_FLASH	Replays a previously recorded mailbox initialization sequence from flash.	50
Hardware Check	HW_CHK	Performs a diagnostic test on the Anybus-S hardware.	51

9.1.1 Start Initialization (START_INIT)

This command initiates the initialization process.

Message Initiator	Application
Message Name	START_INIT
Message Type	1. (Application Message)
Command Number	0001h
Extended Header	-
Command Data	-
Response Data	-

Command and response layout:

	Command	Expected Response	
	(ID)	(ID)	
Message ID	4001h	0001h	<i>Application Message</i>
Message information	0001h	0001h	<i>START_INIT</i>
Command	0000h	0000h	<i>No message data</i>
Data size	0001h	0001h	
	0001h	0001h	
	0000h	0000h	
	0000h	0000h	
Extended word 1	-	-	
Extended word 2	-	-	
Extended word 3	-	-	
Extended word 4	-	-	
Extended word 5	-	-	
Extended word 6	-	-	
Extended word 7	-	-	
Extended word 8	-	-	

9.1.2 Anybus Initialization (Anybus_INIT)

This command is used to configure the data composition of the data exchange buffers, and the way the module should operate on the network. Sending this mailbox is mandatory, either directly or indirectly using the mailbox command 'Load from FLASH'.

Note that the application must monitor the response from the module and verify that the command was accepted.

The initialization parameters passed in the command are parsed by the module. If any parameter exceeds its limits, the response message will contain recommended values. The application will then have to re-send the message with the corrected values.

Note: This command can only be send during module initialization, i.e. between START_INIT and END_INIT.

Message Initiator	Application
Message Name	Anybus_INIT
Message Type	1. (Application Message)
Command Number	0002h
Extended Header	The response header may contain fault information.
Command Data	Initialization parameter data
Response Data	Copy of command data, or suggested maximum values.

Command and response layout:

	Command	Expected Response	
Message ID	(ID)	(ID)	
Message information	4001h	0001h	Application Message
Command	0002h	0002h	Anybus_INIT
Data size	0012h	0012h	9 words of data (18 bytes)
	0001h	0001h	
	0001h	0001h	
	0000h	0000h	
	0000h	0000h	
Extended word 1	-	-	
Extended word 2	-	-	
Extended word 3	-	-	
Extended word 4	-	-	
Extended word 5	-	-	
Extended word 6	-	-	
Extended word 7	-	-	
Extended word 8	-	Fault Information	
Command data word 1	Input I/O Length	Input I/O Length	
Command data word 2	Input DPRAM Length	Input DPRAM Length	
Command data word 3	Input Total Length	Input Total Length	
Command data word 4	Output I/O Length	Output I/O Length	
Command data word 5	Output DPRAM Length	Output DPRAM Length	
Command data word 6	Output Total Length	Output Total Length	
Command data word 7	Operation Mode	Operation Mode	
Command data word 8	Event Notification Config.	Event Notification Config.	
Command data word 9	Watchdog Timeout Value	Watchdog Timeout Value	

- **Fault Information**

If the error code is 'Invalid Other' (Fh), extended error information is presented in this register as follows:

Bit	Fault	Description
0	Input I/O Length	Incorrect length of input I/O
1	Input DPRAM Length	Incorrect length of DPRAM input
2	Input Total Length	Incorrect length of total input
3	(reserved)	
4	Output I/O Length	Incorrect length of output I/O
5	Output DPRAM Length	Incorrect length of DPRAM output
6	Output Total Length	Incorrect length of total output
7	(reserved)	
8	Module Status	Incorrect configuration of bits in Module Status register
9	Event Notification	Incorrect configuration of bits in Event Notification register
10	Incorrect Watchdog	Incorrect Watchdog Counter difference value
11 - 15	(reserved)	

- **Input I/O Length, Input DPRAM Length & Input Total Length**

These parameters determine the composition of the Input Data Buffer. For more information, see "Data Composition" on page 38

- **Output I/O Length, Output DPRAM Length & Output Total Length**

These parameters determine the composition of the Output Data Buffer. For more information, see "Data Composition" on page 38

- **Operation Mode**

This parameter is used to set various options in the module. The contents of this parameter is reflected in the Module Status register in the Control Register Area.

b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
						CD	APFC				RDR	FBSPU	FBS	FBFC	

Bit	Description	State
FBFC	These bits defines the behaviour of the module when the fieldbus goes off line. For more information, see "Module Status (7E2h - 7E3h, RO)" on page 21	For more information, see "Module Status (7E2h - 7E3h, RO)" on page 21
FBS ¹		
FBSPU ¹		
RDR ¹	Fieldbus Reset Device Request notification	0:Disable 1:Enable
APFC ¹	This bit defines how the module should behave when the application has stopped (i.e. a Watchdog timeout)	0:Clear Input Data 1:Freeze Input Data
CD	This bit enables/disables the Changed Data Field registers in the Control Register Area.	0:Disable 1:Enable

- **Event Notification Config.**

This parameter is used to set which events that should trigger an Event Notification. For more information about these bits, "Event Notification Source (7EEh - 7EFh, RO)" on page 23.

- **Watchdog Timeout Value¹**

This parameter is used to set the maximum allowed difference between the Watchdog Counter Input/Output registers (See "Watchdog Counter Input (7D2h - 7D3h, R/W)" on page 19 and

1. The implementation and behavior of this bit depends on the fieldbus type. Consult each separate fieldbus appendix for more information. See also "Deviances" on page 74.

“Watchdog Counter Output (7D4h - 7D5h, RO)” on page 20). When this value is exceeded, the module will signal to the fieldbus that the application is not functioning properly. The range of this parameter is 100 to 30000, which corresponds to a 0.1 - 30 second timeout period. A value of zero will disable this feature.

9.1.3 End Initialization (END_INIT)

This command ends the initialization process.

Note: It is not possible to re-initialise the module without making a software or hardware reset.

Message Initiator	Application
Message Name	END_INIT
Message Type	1. (Application Message)
Command Number	0003h
Extended Header	-
Command Data	-
Response Data	-

Command and response layout:

	Command	Expected Response	
Message ID	(ID)	(ID)	
Message information	4001h	0001h	Application Message
Command	0003h	0003h	END_INIT
Data size	0000h	0000h	No message data
	0001h	0001h	
	0001h	0001h	
	0000h	0000h	
	0000h	0000h	
Extended word 1	-	-	
Extended word 2	-	-	
Extended word 3	-	-	
Extended word 4	-	-	
Extended word 5	-	-	
Extended word 6	-	-	
Extended word 7	-	Secondary Fault Information	
Extended word 8	-	Primary Fault Information	

- **Primary & Secondary Fault Information**
Some modules can return fieldbus-specific error codes via these words during the modules internal initialization. Since these error codes are not generic, please refer to the applicable fieldbus appendix for more information.

9.1.4 Save to FLASH (SAVE_TO_FLASH)

This command is sent to save the Anybus-S initialization in the FLASH. This command can only be sent directly after the START_INIT command. Please observe that issuing this command will erase any previously stored configuration.

Message Initiator	Application
Message Name	SAVE_TO_FLASH
Message Type	1. (Application Message)
Command Number	0004h
Extended Header	The response header may contain fault information.
Command Data	-
Response Data	-

Command and response layout:

	Command	Expected Response	
Message ID	(ID)	(ID)	
Message information	4001h	0001h	<i>Application Message</i>
Command	0004h	0004h	<i>SAVE_TO_FLASH</i>
Data size	0000h	0000h	<i>No message data</i>
	0001h	0001h	
	0001h	0001h	
	0000h	0000h	
	0000h	0000h	
Extended word 1	-	-	
Extended word 2	-	-	
Extended word 3	-	-	
Extended word 4	-	-	
Extended word 5	-	-	
Extended word 6	-	-	
Extended word 7	-	-	
Extended word 8	-	-	
		Fault Information	

- Fault Information**

If 'Flash Config Error' is returned in the Message Information word in the header of the response, information about the fault can be found here.

0001h: Flash is full - the mailbox that responded with this message will be unable to save.

0002h: Store operation error - will not be able to load the configuration.

9.1.5 Load from FLASH (LOAD_FROM_FLASH)

This command is sent to load the Anybus-S initialization from the flash. This command can only be sent directly after the START_INIT command, and a previously saved configuration must be present in the FLASH.

Message Initiator	Application
Message Name	LOAD_FROM_FLASH
Message Type	1. (Application Message)
Command Number	0005h
Extended Header	The response header may contain fault information.
Command Data	-
Response Data	-

Command and response layout:

	Command	Expected Response	
Message ID	(ID)	(ID)	
Message information	4001h	0001h	<i>Application Message</i>
Command	0005h	0005h	<i>LOAD_FROM_FLASH</i>
Data size	0000h	0000h	<i>No message data</i>
	0001h	0001h	
	0001h	0001h	
	0000h	0000h	
	0000h	0000h	
Extended word 1	-	-	
Extended word 2	-	-	
Extended word 3	-	-	
Extended word 4	-	-	
Extended word 5	-	-	
Extended word 6	-	-	
Extended word 7	-	-	
Extended word 8	-	-	
		Fault Information	

- Fault Information**

If 'Flash Config Error' is returned in the Message Information word in the header of the response, information about the fault can be found here.

0003h: CRC mismatch or FLASH empty - responds directly and will not load the configuration.

0004h: LOAD failed - module cannot load the configuration, the module will be reset.

9.1.6 Hardware Check (HW_CHK)

This command instructs the module to perform a diagnostic test on the hardware. This includes the RAM, DPRAM, FLASH (CRC test) and possibly the ASIC depending on fieldbus type.

If any errors are detected, the module will not respond to any command until a hardware reset is performed. The Anybus-S Watchdog will indicate the type of error, for more information see “Anybus-S Watchdog LED” on page 64.

Note: The command can only be sent before the START_INIT command.

Message Initiator	Application
Message Name	HW_CHK
Message Type	1. (Application Message)
Command Number	0006h
Extended Header	-
Command Data	-
Response Data	-

Command and response layout:

	Command	Expected Response	
Message ID	(ID)	(ID)	
Message information	4001h	0001h	<i>Application Message</i>
Command	0006h	0006h	<i>HW_CHK</i>
Data size	0000h	0000h	<i>No message data</i>
	0001h	0001h	
	0001h	0001h	
	0000h	0000h	
	0000h	0000h	
Extended word 1	-	-	
Extended word 2	-	-	
Extended word 3	-	-	
Extended word 4	-	-	
Extended word 5	-	-	
Extended word 6	-	-	
Extended word 7	-	-	
Extended word 8	-	-	

9.2 Fieldbus Messages

This category includes commands for accessing fieldbus specific data and functions, and are described in each separate fieldbus appendix.

9.3 Internal Memory Messages

General

This category includes functions for accessing the Internal Memory.

Messages in This Category

Message	Abbreviation	Description	Page
Read Internal Input Area	RD_INT_IN	Reads a block of data from the internal input area.	53
Write Internal Input Area	WR_INT_IN	Writes a block of data to the internal input area.	54
Clear Internal Input Area	CLR_INT_IN	Clears a block of data in the internal input area.	55
Read Internal Output Area	RD_INT_OUT	Reads a block of data from the internal output area.	56

9.3.1 Read Internal Input Area (RD_INT_IN)

This command is used to read a block of data from the Internal Input Area. It is possible to read up to 256 bytes of data with each command.

Message Initiator	Application
Message Name	RD_INT_IN
Message Type	3. (Internal Memory Message)
Command Number	0001h
Extended Header	Contains the source offset address and size of the data block
Command Data	-
Response Data	Contents of read data block.

Command and response layout:

Command		Expected Response	
Message ID	(ID)	(ID)	
Message information	4003h	0003h	Internal Memory Message
Command	0001h	0001h	RD_INT_IN
Data size	0000h	(data size)	(same as Block Size)
	0001h	0001h	
	0001h	0001h	
	0000h	0000h	
	0000h	0000h	
Extended word 1	Block Offset	Block Offset	
Extended word 2	Block Size	Block Size	
Extended word 3	-	-	
Extended word 4	-	-	
Extended word 5	-	-	
Extended word 6	-	-	
Extended word 7	-	-	
Extended word 8	-	-	
			Response data word 1
		Data Block	...
			Response data word n

- **Block Offset**
Address offset from the start of the Input Data Buffer.
- **Block Size**
Size of the block that should be read (in bytes).
- **Data Block**
The actual data block.

9.3.2 Write Internal Input Area (WR_INT_IN)

This command is used to write a data block to the Internal Input Area. It is possible to write up to 256 bytes of data with each command.

Message Initiator	Application
Message Name	WR_INT_IN
Message Type	3. (Internal Memory Message)
Command Number	0002h
Extended Header	Contains destination offset address and size of the data block
Command Data	Data that should be written.
Response Data	Copy of the written data.

Command and response layout:

Command		Expected Response	
Message ID	(ID)	(ID)	
Message information	4003h	0003h	Internal Memory Message
Command	0002h	0002h	WR_INT_IN
Data size	(data size)	(data size)	(same as Block Size)
	0001h	0001h	
	0001h	0001h	
	0000h	0000h	
	0000h	0000h	
Extended word 1	Block Offset	Block Offset	
Extended word 2	Block Size	Block Size	
Extended word 3	-	-	
Extended word 4	-	-	
Extended word 5	-	-	
Extended word 6	-	-	
Extended word 7	-	-	
Extended word 8	-	-	
Command data word 1	Data Block		Response data word 1
...		Data Block	...
Command data word n			Response data word n

- **Block Offset**
Address offset from the start of the Input Data Buffer.
- **Block Size**
Size of the block that should be written (in bytes).
- **Data Block**
The actual data block.

9.3.3 Clear Internal Input Area (CLR_INT_IN)

This command is used to clear blocks of data in the Internal Input Area. It is possible to clear up to 256 bytes of data with each command. Several commands are required to clear the whole area.

Message Initiator	Application
Message Name	CLR_INT_IN
Message Type	3. (Internal Memory Message)
Command Number	0003h
Extended Header	Contains destination offset address and size of the data block that should be cleared
Command Data	-
Response Data	-

Command and response layout:

	Command	Expected Response	
Message ID	(ID)	(ID)	
Message information	4003h	0003h	<i>Internal Memory Message</i>
Command	0003h	0003h	<i>CLR_INT_IN</i>
Data size	0000h	0000h	<i>No message data</i>
	0001h	0001h	
	0001h	0001h	
	0000h	0000h	
	0000h	0000h	
Extended word 1	Block Offset	Block Offset	
Extended word 2	Block Size	Block Size	
Extended word 3	-	-	
Extended word 4	-	-	
Extended word 5	-	-	
Extended word 6	-	-	
Extended word 7	-	-	
Extended word 8	-	-	

- **Block Offset**

Address offset from the start of the Input Data Buffer.

- **Block Size**

Size of the block that should be cleared (in bytes).

9.3.4 Read Internal Output Area (RD_INT_OUT)

This command is used to read a block of data from the Internal Output Area. It is possible to read up to 256 bytes of data with each command.

Message Initiator	Application
Message Name	RD_INT_OUT
Message Type	3. (Internal Memory Message)
Command Number	0004h
Extended Header	Contains the source offset address and size of the data block
Command Data	-
Response Data	Contents of read data block.

Command and response layout:

Command		Expected Response	
Message ID	(ID)	(ID)	
Message information	4003h	0003h	Internal Memory Message
Command	0004h	0004h	RD_INT_OUT
Data size	0000h	(data size)	(same as Block Size)
	0001h	0001h	
	0001h	0001h	
	0000h	0000h	
	0000h	0000h	
Extended word 1	Block Offset	Block Offset	
Extended word 2	Block Size	Block Size	
Extended word 3	-	-	
Extended word 4	-	-	
Extended word 5	-	-	
Extended word 6	-	-	
Extended word 7	-	-	
Extended word 8	-	-	
			Response data word 1
		Data Block	...
			Response data word n

- **Block Offset**
Address offset from the start of the Output Data Buffer.
- **Block Size**
Size of the block that should be read (in bytes).
- **Data Block**
The actual data block.

9.4 Reset Messages

General

This category includes reset related functions.

Messages in This Category

Message	Abbreviation	Description	Page
Software Reset	SW_RESET	Performs a software reset of the module.	57

9.4.1 Software Reset (SW_RESET)

This command is used if a restart of the Anybus-S module for some reason is required, e.g. if some initialization data has to be changed. The application has 1 second to read the response before the reset command is activated.

Note: This command only makes a software reset of the module, not a hardware reset.

Message Initiator	Application
Message Name	SW_RESET
Message Type	5. (Reset Message)
Command Number	0001h
Extended Header	-
Command Data	-
Response Data	-

Command and response layout:

	Command	Expected Response	
Message ID	(ID)	(ID)	
Message information	4005h	0005h	<i>Reset Message</i>
Command	0001h	0001h	<i>SW_RESET</i>
Data size	0000h	0000h	<i>No message data</i>
	0001h	0001h	
	0001h	0001h	
	0000h	0000h	
	0000h	0000h	
Extended word 1	-	-	
Extended word 2	-	-	
Extended word 3	-	-	
Extended word 4	-	-	
Extended word 5	-	-	
Extended word 6	-	-	
Extended word 7	-	-	
Extended word 8	-	-	

10. Start Up and Initialization

10.1 Introduction

At any given time, the Anybus-S module will be in one out of three possible states:

1. Hardware Initialization State

This is the initial state of the module after power on or reset. To get to the next state, various diagnostic tests on the hardware should be performed, see “Hardware Initialization” on page 59.

- No data exchange is possible in this state.

2. Software Initialization State

In this state, the basic operating parameters are set. In order to proceed to the next state, the module must first successfully pass the software initialization sequence, see “Software Initialization” on page 60.

- No data exchange is possible in this state.

3. Data Exchange State

In this state, the module is able to exchange data between the fieldbus and two I/O data buffers. The only way to get to this state is to successfully go through the other states.

This chapter describes the steps involved in state 1 and 2. For more information about state 3 (Data Exchange State), see “Fieldbus Data Exchange” on page 36.

10.2 Hardware Initialization

This procedure is mandatory and ensures that the module is working properly before the software initialization sequence. The procedure consists of the following steps:

- Startup Sequence
- Dual Port Memory Check (Optional)
- The Hardware Check mailbox message (Optional)

10.2.1 Startup Sequence

Depending on how the application has been implemented, the startup procedure differs slightly:

- **Hardware Interrupt feature not implemented**
After power on/reset, the application should poll the Watchdog Counter Out register (7D4h - 7D5h) approximately every 10ms to detect if it has been properly updated by the module. When this register has been updated by the module at least 10 times, the module can be considered to be up and running.
- **Hardware Interrupt feature implemented**
After power on/reset, the Anybus module generates an interrupt to indicate that it is ready. Before this interrupt, the application is not allowed to write any data in the dual port memory. The interrupt line is managed entirely by the internal logic of the DPRAM; this logic is affected by a power on reset but not by a hardware reset. Therefore, certain precautions are needed to ensure proper functionality. For more information, see “Interrupt Line and Hardware Reset” on page 76.

10.2.2 Dual Port Memory Check (Optional)

It is recommended to perform a read/write test of the dual port memory. This can be done in two ways, depending on the nature of the application:

- **Hardware Reset not implemented (Or not controlled by the application software)**
The test should be carried out directly after the Startup Sequence described above. The test must be non-destructive, i.e. the data in the dual port memory must be restored after the test. Also, it is important to exclude the following from these tests as this would interfere with the operation of the module:
 - Watchdog Counter In register (7D2h - 7D3h)
 - Watchdog Counter Out register (7D4h - 7D5h)
 - Handshake Registers (7FEh - 7FFh)
 - LED indication status (7DAh-7DFh)
 - Module Status (7E2h-7E3h)
 - Fieldbus Specific Area (640h - 7BFh)
- **Hardware Reset implemented (And controlled by the application software)**
The dual port memory test should be carried out while the reset line is held low by the application, prior to the Startup Sequence described above. All memory locations can be tested, and the test can be either destructive or non-destructive.

10.2.3 Hardware Check (Optional)

Note: This step can only be performed after the Startup Sequence and the Dual Port Memory Check.

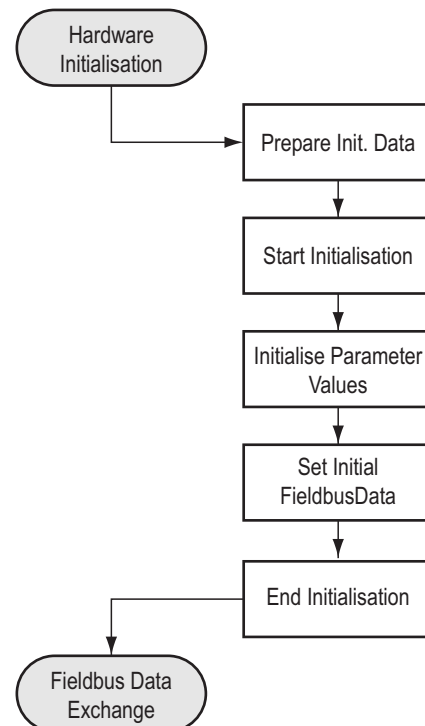
By sending the mailbox command 'Hardware Check', the module will be instructed to perform a hardware self-test. If the test is successful, the module will respond. If the module does not respond, the test failed. For more information, see "Hardware Check (HW_CHK)" on page 51) and "Anybus-S Watch-dog LED" on page 64.

10.3 Software Initialization

Before any fieldbus communication can take place, the software in the Anybus-S module must be initialised. This process is mandatory and decides how the module should operate on the network.

The software initialization process basically consists of the following steps:

1. Prepare Initialization Data (Optional)
2. Start Initialization
3. Initialise Parameter Values
4. Set Initial Fieldbus Data (Optional)
5. End Initialization



10.3.1 Prepare Initialization Data

This step is optional, but will enable the application to take advantage of advanced fieldbus specific features without requiring multiple software versions. To accomplish this, the application needs to know the type of Anybus module that is connected, and possibly other information such as software revisions etc. During initialization, this information can be read directly from the Control Register Area without handshaking.

The application can then modify the software initialization parameters to better exploit a specific Anybus model.

10.3.2 Start Initialization

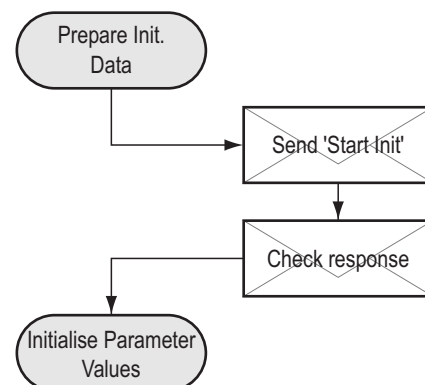
This step is mandatory, and instructs the module to start the software initialization sequence.

1. Send the mailbox message 'Start Init'

This will instruct the module to start the software initialization process.

2. Check response

The module is now ready to accept further configuration mailbox messages.



10.3.3 Initialise Parameter Values

This step is mandatory, however the exact procedure may vary from case to case. The following steps are involved.

- **Send mailbox command 'Save to FLASH' (Optional)**

This mailbox command works very much like a tape recorder that records the following mailbox commands. The recording stops when the module receives the 'End Init' mailbox command.

- **Send mailbox command 'Load from FLASH' (Optional)**

This mailbox command replays a previously recorded mailbox sequence made using the 'Save to FLASH' mailbox command.

- **Send mailbox command 'Anybus Init'**

This mailbox command is used to configure the data composition of the data exchange buffers, and the way the module should operate. Sending this mailbox is mandatory, either directly or indirectly using the mailbox command 'Load from FLASH'.

Note that the application must monitor the response from the module and verify that the command was accepted.

- **Send fieldbus specific initialization commands (Optional)**

This procedure usually involves sending fieldbus specific mailbox commands to the module, consult each separate fieldbus appendix for more information. Note that some fieldbus specific initialization commands must be sent before Anybus Init. This has to be accounted for in the application software.

Note: Please note that using fieldbus specific initialization commands may void the fieldbus pre-certification, see "Fieldbus Certification" on page 81.

10.3.4 Set Initial Fieldbus Data

This step is optional, but allows the application to have control over the contents of the Input Data Buffer before the first bus cycle. Depending on the location of the data that should be written, the application must either write the data to the Input Data Area in the DPRAM or send it to the module using Internal Memory commands, see "Internal Memory Messages" on page 52.

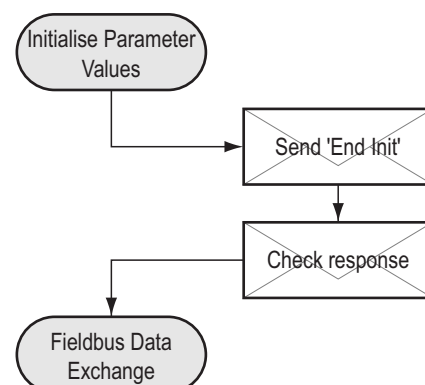
10.3.5 End Initialization

This step signals to the module that the initialization is done, and that the module should start exchanging data on the fieldbus.

1. **Send the mailbox message 'End Init'**

2. **Check response**

If the response is ok, the module will start exchanging data on the fieldbus.



10.3.6 Basic Initialization Sequence Example 1

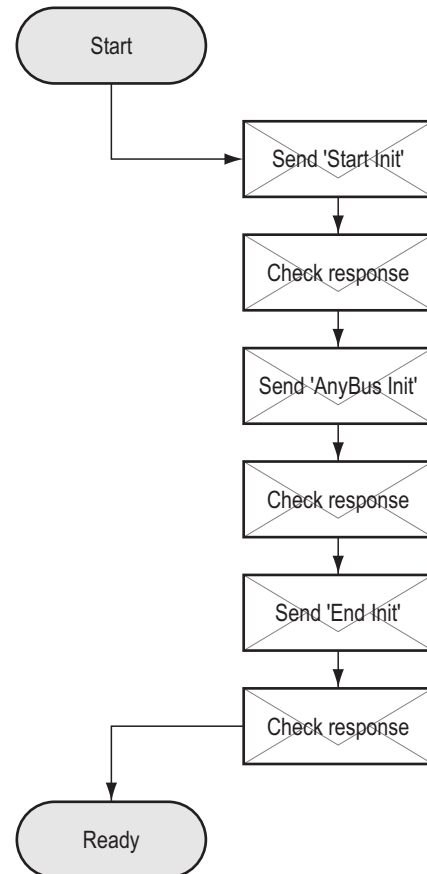
(Note that procedure below assumes that the parameters sent with the 'Anybus Init' command are valid.)

If only basic fieldbus functionality is required, a very basic initialization sequence like the one below can be used.

■ Power On (Reset)

1. Send mailbox command 'Start Init'
2. Check response
3. Send mailbox command 'Anybus Init'
4. Check response
5. Send mailbox command 'End Init'
6. Check response

■ Ready



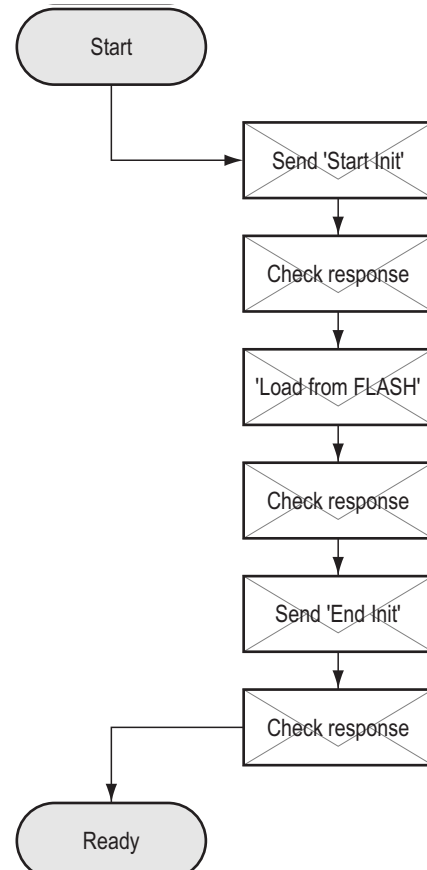
10.3.7 Basic Initialization Sequence Example 2

In this example, the initialization mailbox sequence is loaded from flash. Note that this requires that an initialization sequence has previously been stored in flash using the 'Save to FLASH' mailbox command.

■ Power On (Reset)

1. Send mailbox command 'Start Init'
2. Check response
3. Send mailbox command 'Load from flash'
4. Check response
5. Send mailbox command 'End Init'
6. Check response

■ Ready



10.3.8 Advanced Initialization Example

In this example, the initialization sequence is adapted for the currently used Module/Fieldbus type, and the Input Data Buffer is updated before the first bus cycle.

■ Power On (Reset)

1. Check Module/Fieldbus Type

This information can be read without handshaking from the Control Register Area.

2. Prepare Initialization Data

Based on the information from the Control Register Area, the application can decide what initialization parameters to use in the remainder of the initialization process.

3. Send mailbox command 'Start Init'

This step starts the initialization sequence.

4. Check response

5. Fieldbus Specific Initialization

The procedure for this step is different for each fieldbus and is described in each separate fieldbus appendix.

6. Send mailbox command 'Anybus Init'

'Anybus Init' is used to set up I/O sizes and various configuration bits.

7. Check response

8. Response OK?

If not, modify the parameters for the Anybus Init mailbox message and go to step 4.

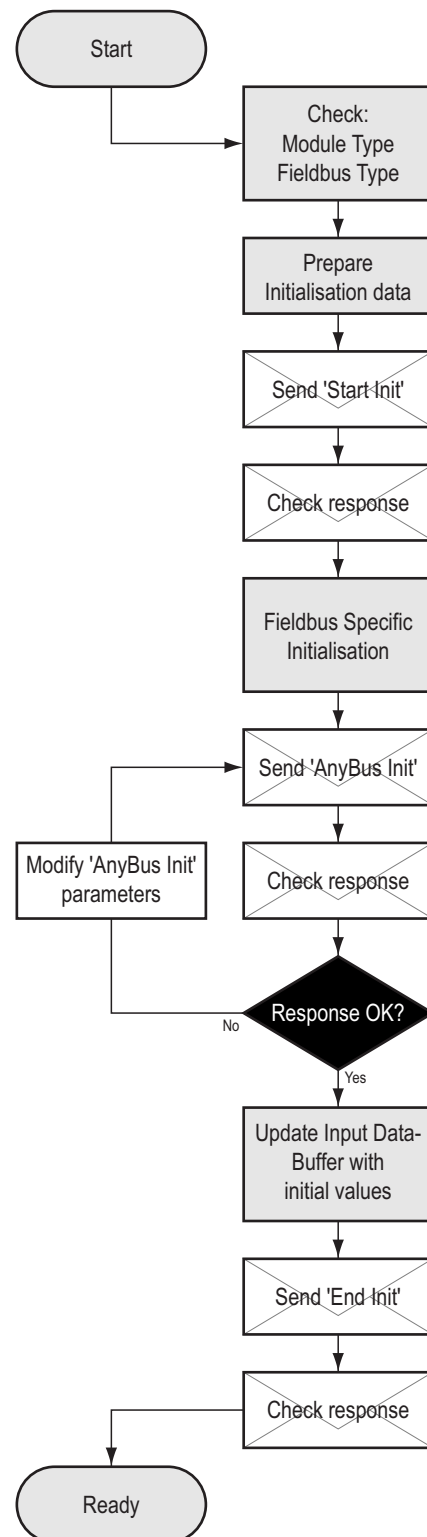
9. Update the Input Area with initial values

10. Send mailbox command 'End Init'

This step ends the initialization sequence.

11. Check response

■ Ready



11. Indication LEDs

11.1 Fieldbus Status Indicators

The module features four front mounted status LED's implemented in accordance with each fieldbus standard. The function of these LED's is fieldbus dependant and is described in each fieldbus appendix.

11.2 Anybus-S Watchdog LED

All Anybus-S modules features a surface mounted watchdog LED, indicating the status of the module.

Colour	Frequency	Indication
Red	-	Unspecified internal error, or running in bootloader mode
	1Hz	RAM failure
	2Hz	ASIC or FLASH failure
	4Hz	DPRAM failure
Green	2Hz	Module not initialised
	1Hz	Module initialised and running OK

12. Firmware Upgrade

To be able to keep the continuous product development on the Anybus-S module and to be able to give upgrades to our customers with more improved features, a FLASH memory is used in all Anybus-S modules. This means that the firmware in the module can be updated in the field after production.

The module supports serial firmware download:

- **Serial Download via Asynchronous Serial Interface**

This method requires that the asynchronous serial interface pins on the Anybus application connector is connected to a standard PC via RS232 line drivers. The firmware can then be downloaded using a Microsoft Windows application supplied by HMS.

This method may require access to the circuit board, depending on module. Please contact HMS for more information.

A simple but fully functional implementation can be made by adding a four-pin connector that will connect the VCC, GND, TX and RX pins in the application connector to an external RS232-to-TTL level converter. The TX/RX pins are used for the communication and the VCC/GND supply the converter with power, eliminating the need for an external supply.

The application software also needs some preparations since the module will only accept download commands immediately after a reset. Once the application has begun any initialization of the module it must be reset or power-cycled in order for a download to be possible.

On newer modules it may also be required to close a jumper on the module circuit board in order to be able to download new firmware, contact HMS for more information.

13. Driver Example

The example driver is composed of three routines, called ‘Handlers’:

- **Interrupt Handler**

This routine handles interrupts received from the Anybus-S. For more information, see “Interrupt Handler” on page 67.

- **Interface Handler**

This routine should be called cyclically from the main program. For more information, see “Interface Handler” on page 68.

- **Mailbox Handler**

This routine can be called cyclically from the main program, or when the need arises. For more information, see “Mailbox Handler” on page 69.

Implementation Notes

Also, it is assumed that the code in these examples is executed in a multitasking environment. If the application does not feature multitasking capability, the code should be implemented as a state machine or similar rather than linear code.

Important!

These example routines are far from optimal and is included for guidance only. Some essential functions for Initialization, timeout, Event Notification and error handling etc. are left out and has to be implemented in order to be able to operate the module properly.

Global Variables

The following global variables are used in the example routines:

Variable	Type	Description
INIT	Boolean	These variables corresponds to individual bits in the Anybus Indication Register, see “Anybus Indication Register (7FFh, RO)” on page 27
AB_IN	Boolean	
AB_OUT	Boolean	
AB_FBCTRL	Boolean	
AB_MOUT	Boolean	
AB_MIN	Boolean	
AB_EVNT	Boolean	
AP_MIN	Boolean	These variables corresponds to individual bits in the Application Indication Register, see “Application Indication Register (7FEh, R/W)” on page 26
AP_MOUT	Boolean	
AP_EVNT	Boolean	
ABS_INITIALISED	Boolean	This variable indicates that the module has been initialised.
IN_AREA_FREE	Boolean	These variables indicates if an area in the DPRAM is free to use. (Set = free)
OUT_AREA_FREE	Boolean	
FBCTRL_AREA_FREE	Boolean	
MBX_OUT_NEW	Boolean	This variable indicates that the Mailbox Output Area contains a new message.
MBX_IN_FREE	Boolean	This variable indicates that the Mailbox Input Area is free to use.

13.1 Interrupt Handler

The purpose of this routine is to interpret the contents of the Anybus Indication Register upon receiving an interrupt, and to make this information available to the Interface/Mailbox Handlers and the main application program.

To provide fast interrupt handling, no further processing is performed in this routine; i.e. all processing is performed in the Interface Handler instead.

Note: Although this routine is called 'Interrupt Handler' it can be used even if the interrupt pin is not implemented in the application. In this case, the routine must be called periodically and/or when needed in order to refresh its status variables. Exactly how this should be implemented is not the scope of this chapter. Again, it is strongly recommended to use the interrupt feature whenever possible.

Step by Step

■ Start

1. Read the Anybus Indication Register

2. Check if the module is initialised

(Store status in ABS_INITIALISED)

3. Check if the Input Area is free

(Store status in IN_AREA_FREE)

4. Check if the Output Area is free

(Store status in OUT_AREA_FREE)

5. Check if the Fieldbus Specific Area & Control Register Area is free

(Store status in FBCTRL_AREA_FREE)

6. Check if the Mailbox Output Area contains a new message

(Store status in MBX_OUT_NEW)

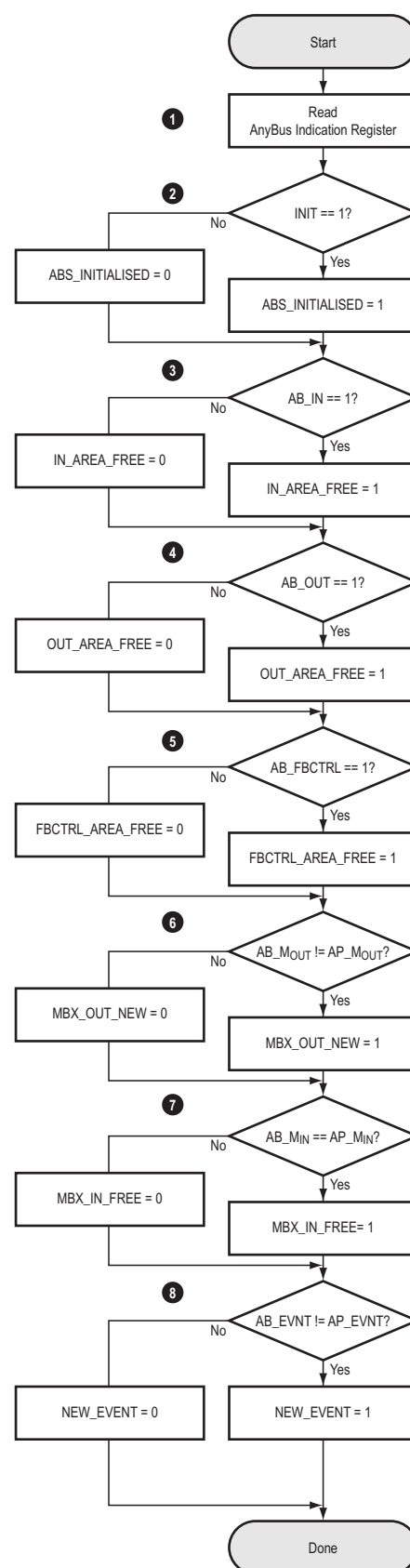
7. Check if the Mailbox Input Area is free

(Store status in MBX_IN_FREE)

8. Check if an Event Notification has occurred

(Store status in NEW_EVENT)

■ Done



13.2 Interface Handler

The Interface Handler should be called cyclically from the main program.

The purpose of this routine is to transfer data and to perform all necessary hand-shaking for the Input/Output Data Areas and the Fieldbus Specific/Control Register Areas.

If an area within the dual port memory is currently in use, a request will be made so that the area can be accessed the next time the routine is called.

Step by Step

■ Start

1. New data for Input Area?

If yes, access the data, release the area and wait for response.

If no, request and if possible access the area. If the area is still in use, a new attempt will be made the next time the routine is called.

3. Time to read Output Area?

If yes, access the data, release the area and wait for response.

If no, request and if possible access the area. If the area is still in use, a new attempt will be made the next time the routine is called.

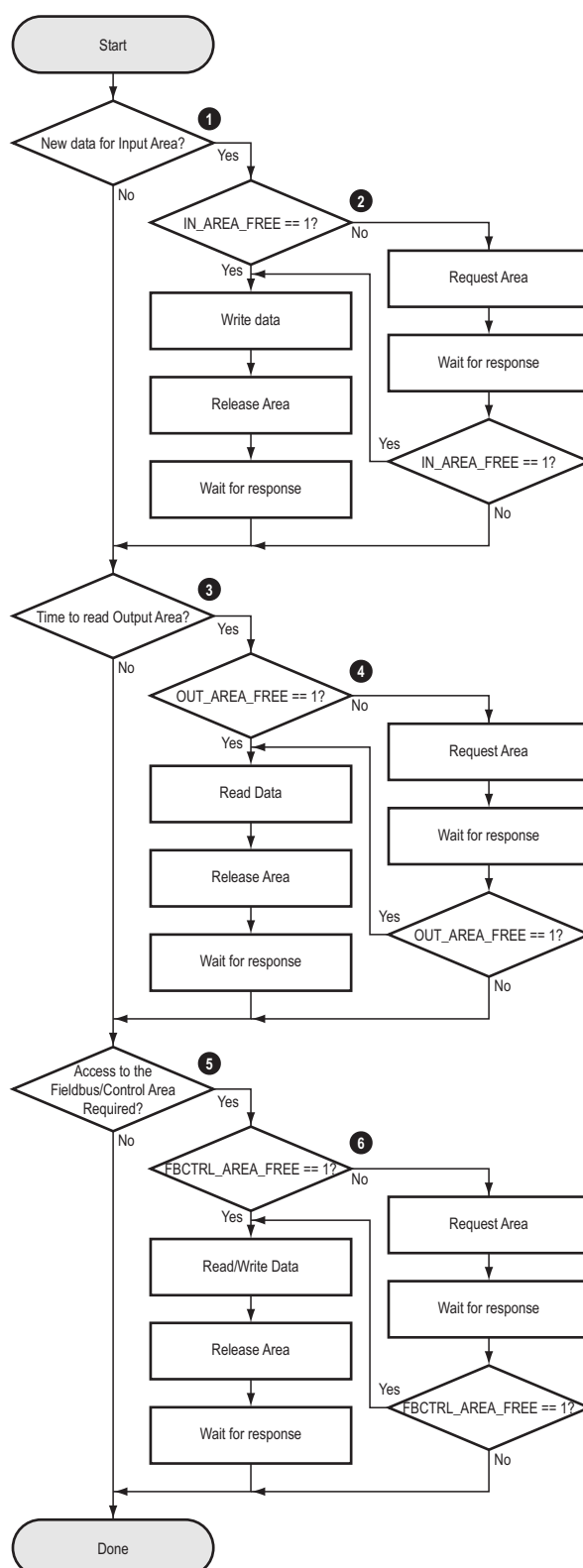
5. Access to the Fieldbus Specific Area & Control Register Area required?

6. Is the area free to use?

If yes, access the data, release the area and wait for response.

If no, request and if possible access the area. If the area is still in use, a new attempt will be made the next time the routine is called.

■ Done



13.3 Mailbox Handler

The Mailbox Handler should be called cyclically from the main program.

The purpose of this routine is to exchange mailbox messages with the module. The routine handles both incoming and outgoing mailbox traffic:

- **Application to Anybus module**

If the application wishes to send a mailbox message to the module, the routine will first check if the Mailbox Input Area is free (i.e. `MBX_IN_FREE == 1`), and if possible, transfer the message. The routine will then toggle the `AP_MIN` bit to signal to the module that a message has been entered in the Mailbox Input Area.

If the Mailbox Input Area is currently in use (i.e. `MBX_IN_FREE == 0`), the routine will attempt to send the message the next time the routine is called.

- **Module to Application**

If the application is able to receive a new mailbox message and a new message is detected (i.e. `MBX_OUT_NEW == 1`), the routine will transfer the message and acknowledge this to the module by toggling the `AP_MOUT` bit in the Application Indication Register.

Step by Step

- **Start**

1. **Should a new mailbox message be sent to the Anybus module?**

(If not, the routine jumps to step 3)

2. **Is the Mailbox Input Area free?**

If yes, write the message to the Mailbox Input Area. When done, toggle the `AP_MIN` bit to signal to the module that a new message has been entered.

If no, do nothing; a new attempt will be made the next time the routine is called.

3. **Is the application ready to receive a new mailbox message?**

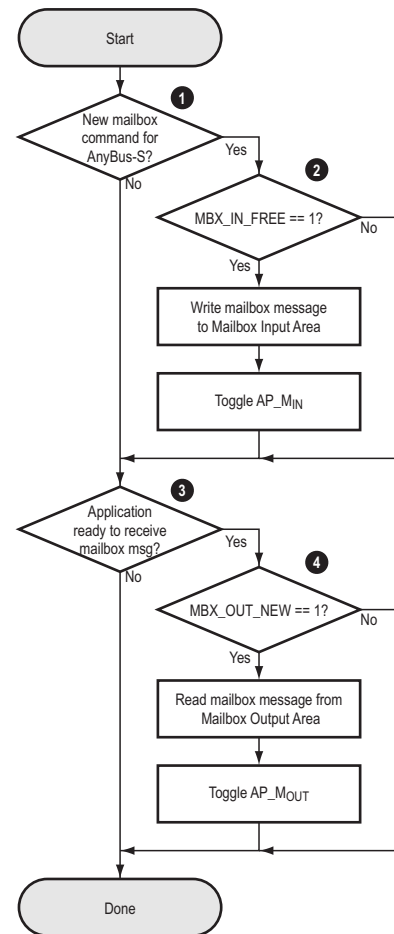
(If not, the routine exits)

4. **Does the Mailbox Output Area contain a new message?**

If yes, read the message from the Mailbox Output Area. When done, toggle the `AP_MOUT` bit to acknowledge that the message has been read.

If no, do nothing; a new attempt will be made the next time the routine is called.

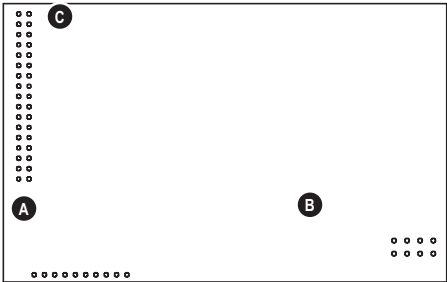
- **Done**



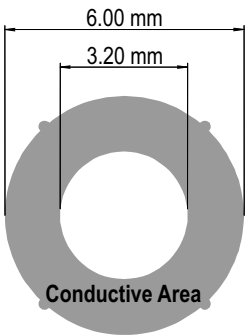
14.3 Mounting Holes

All Anybus-S modules features three metal plated mounting holes for mechanical fastening. Two of these holes are used for improved electrical connection between the application and the module, see table below.

(For measurements, see “PCB Measurements” on page 70.)

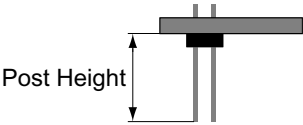
#	Description	Position
A	Conductive hole, used to connect PE (Protective Earth) with fieldbus electronics (Shield). (This hole is connected to the fieldbus cable shield via a filter according to the fieldbus specification).	
B	Electrically isolated hole for use with conductive or non conductive screws.	
C	Conductive hole, used for improved GND connection.	

Both on the conductive and non-conductive holes there is an area around the hole where no PCB wires are placed on the Anybus module. This area has a diameter of 8mm.



14.4 Application Connector

The application connector is a 2mm low profile strip header that can be mounted on either side of the module. If required, alternative solutions are available for an additional charge, e.g. the module can be equipped with other standard or non-standard headers with other post heights.

Standard Post Heights	
6.4 mm	
8.1 mm	
10.1 mm	
12.2 mm	

(For measurements, see “PCB Measurements” on page 70.)

14.5 Fieldbus Connector(s)

To be able to fulfil the connector specification for all major fieldbus systems, most Anybus-S modules can be equipped with several different connector types. A 2mm strip header is also available for applications where the fieldbus connector should be relocated to the application board.

The positions of the first fieldbus connector and the 2mm strip headers are fixed. Some fieldbus systems require a second fieldbus connector, however the position of this connector is not fixed.

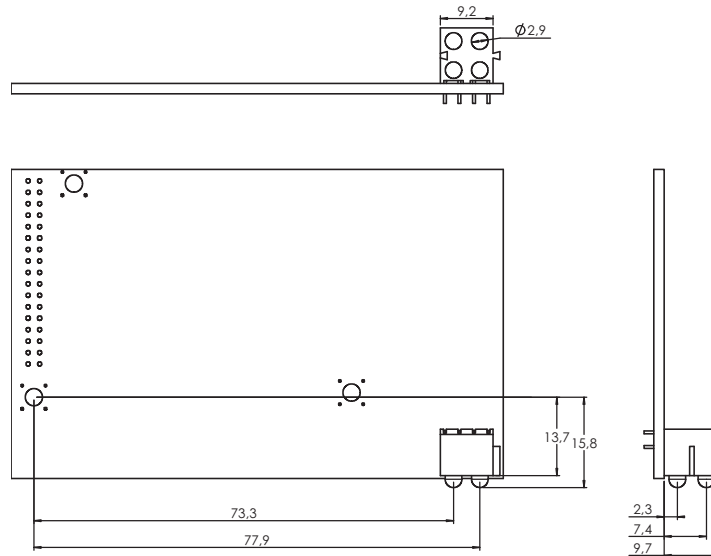
(For measurements, see “PCB Measurements” on page 70.)

14.6 Fieldbus Status Indication LED's

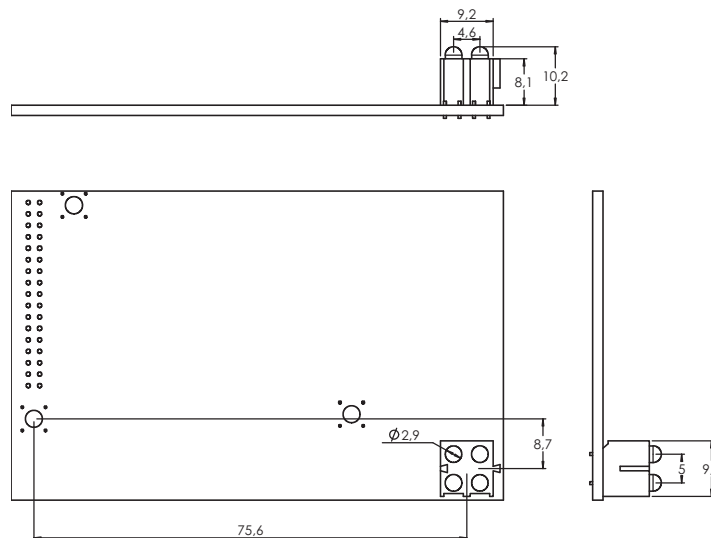
All Anybus-S modules features four fieldbus status indication LED's, available in both straight (180°) and right-angled (90°) configurations. The position of these LED's is standardised on all modules to facilitate the design of LED description panels etc.

The module can also be supplied with a 2.54 2x4 header if the LED's are to be relocated to the application board (for measurements, see "PCB Measurements" on page 70).

Angled LED's



Straight LED's



A. Extended Memory Mode (4K DPRAM)

The Anybus-S platform has been extended to allow faster access to fieldbus data. This is accomplished by using pin 34 of the application connector as address line 11 (A11), giving an effective address range of 4kbyte instead of the standard 2kbyte.

At the time of writing, this feature is only used by the Anybus-M Profibus DPV1 Master and the Anybus-M Devicenet Master/Scanner, but it may be used in future versions of the Anybus-S. Regardless of DPRAM size, all future Anybus-S versions are backwards compatible with applications where A11 is not implemented.

Implementing this feature will affect the memory map,. see below.

A11 not implemented:	Area:	A11 Implemented:
000h - 1FFh	Input Data Area	800h - 9FFh (Standard Mode) 000h - 5FFh (Extended Mode)
200h - 3FFh	Output Data Area	A00h - BFFh (Standard Mode) 600h - BFFh (Extended Mode)
400h - 51Fh	Mailbox Input Area	C00h - D1Fh
520h - 63Fh	Mailbox Output Area	D20h - E3Fh
640h - 7BFh	Fieldbus Specific Area	E40h - FBFh
7C0h - 7FDh	Control Register Area	FC0h - FFDh
7FEh - 7FFh	Handshake Registers	FFEh - FFFh

In order to be able to support future this feature, address line 11 must be implemented in the application, and the address offset used in the software *must* be recalculated relative to the memory map above.

B. Deviances

B.1 General

The Anybus-S is designed to support virtually any type of network, and to present network events and functions in a uniformed manner to the application in the widest extent possible.

To accomplish this, the Anybus-S specification includes a vast amount of bit coded status information. Which of these bits that are actually used, and their exact interpretation, are fieldbus dependant.

For this reason, an implementation that relies heavily on a specific status bit or function may require some slight modifications when changing fieldbus system, i.e. a completely transparent Anybus-S implementation cannot be done unless these slight differences are accounted for.

Also, in order to support fieldbus specific features, such as the socket interface of an Anybus-S Ethernet module, special support has to be included in the application software.

Information about these deviances are presented in each separate fieldbus appendix.

B.2 Locked Release Behavior

When a locked release of the Output Data Area is requested, the area cannot be accessed again until the Anybus module has updated it. Most modules update this area each cycle, whether there are changes to the data or not, but there are exceptions. There are some modules that update the Output Data Area only when there are changes to the output data, implying that the area can stay locked for several cycles. This has to be taken into account when configuring an application, as you may have to rely on external events for the area to be released.

The following modules update the Output Data Area only when the data has changed:

Module	Comment
ABS PROFIBUS DP-V0	-
ABS PROFIBUS DP-V1	-
ABS PROFIBUS DP-V1 I&M	-
ABS EtherNet/IP	-
ABS CANopen	When only COS ^a -data is used
ABS DeviceNet	When only COS-data is used

a. COS = Change of state

B.3 Application Interface Hardware Deviances

During the product's life cycle, certain minor changes has been made to the application interface hardware in order to improve signal characteristics etc. The table below lists these changes. Generally, all new products are implemented according to what is stated in “Application Connector” on page 12.

At the time of writing, products not listed below or products with a higher revision numbers than the ones listed below can be assumed to be implemented in accordance with what is stated earlier in “Application Connector” on page 12. (Products with lower revision numbers are considered obsolete and may have slight variations in resistor values etc. If this is the case with your product, contact HMS for further information.)

Anybus Module	PCB		A4/DE	\overline{CE}	\overline{IRQ}	\overline{BUSY}	A10	A11
	ID #	Revision						
Anybus-M DeviceNet	2102	2.3.1	-	-	10k	10k	10k	10k
Anybus-M DPV	2101	1.1.1	-	10k	10k	10k	-	10k
Anybus-S Interbus (500k)	3265	1.20 - 1.30	-	-	10k	10k	10k	x
Anybus-S Interbus Fibre Optic (500k)	3273	1.01 - 1.10	-	-	100k	100k	100k	x
Anybus-S Modbus Plus	4028	1.11	-	-	10k	10k	10k	x

-	no pull up resistor
10k	10k pull up resistor
x	Signal not implemented

C. Interrupt Line and Hardware Reset

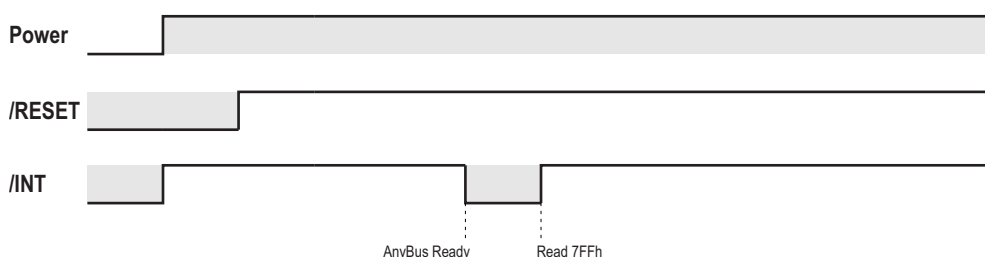
As described earlier, the Anybus-S generates an interrupt to indicate that it is ready after a power on reset. It is furthermore mentioned that the application is not allowed to write any data in the DPRAM before this interrupt is generated.

The interrupt line is however managed entirely by the internal logic of the dual port ram; this logic is affected by a power on reset but not by a hardware reset. If the application utilizes hardware reset, for example via a manual reset button, there is a risk of the interrupt line being held at a low level from the start. Therefore, certain precautions are needed to ensure proper functionality.

The following diagrams describe the different situations¹.

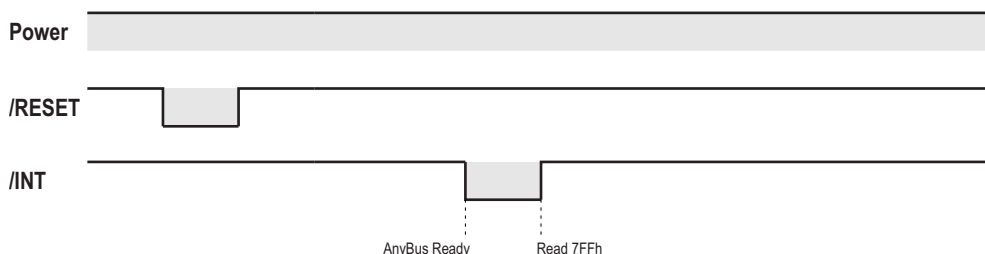
Normal Power On-Reset

The interrupt line (/INT) is initialized to a high level after a power-on reset.



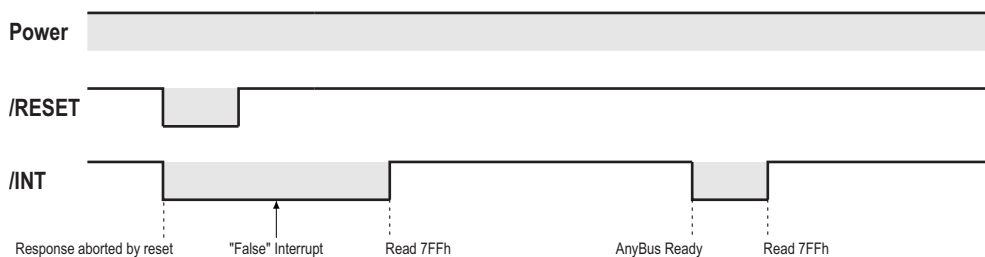
Hardware Reset – Case 1

A hardware reset occurs while the interrupt line is high, resulting in a normal start-up.



Hardware Reset – Case 2

A hardware reset occurs after the Anybus-S has responded to a request but before the application has read the handshake register. In this case there is a potential risk that the application starts the handshake procedures before the Anybus-S is ready.

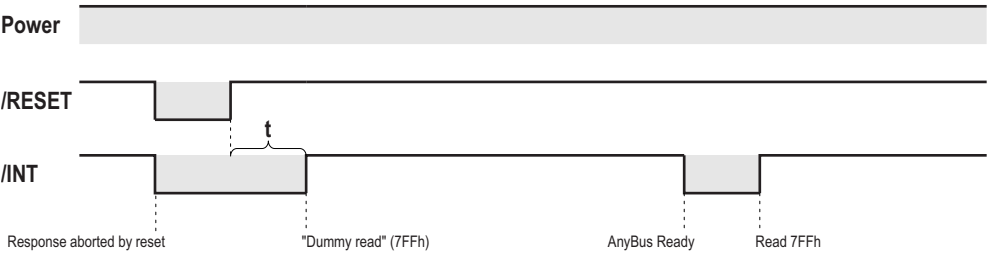


1. These diagrams do not show exact timing, but rather the relationship and sequence of events.

C.1 Recommended Solution

For a safe initialization in any of the situations described previously, it is recommended that the application perform an extra “dummy” read of the Anybus Indication Register (7FFh), while the reset line is held low or within $t < 120\text{ ms}$ after the reset line is released.

Hardware Reset – Case 2 Handled Correctly



D. Electrical Specification

D.1 Power Supply Requirements

The module features dual power supply pins. These can either be tied together or powered separately, depending on the power supply in the host application. Either way, both must be powered in order for the module to operate properly. Generally, it is recommended to tie both power supplies together.

Symbol	Item		Min.	Typ.	Max.	Unit
I_{IN}	Current Consumption	Bus Interface (Pin 1)	-	-	300	mA
		Module Electronics (Pin 5)	-	-	300	mA
		Both (Pins 1 and 5)	-	-	450 ^a	mA
V_{CC}	Supply Voltage (DC)	Bus Interface (Pin 1)	4.75	5.00	5.25	V
		Module Electronics (Pin 5)	4.75	5.00	5.25	V
	Maximum Ripple (AC)	Bus Interface (Pin 1)	-	-	± 100 pp	mV
		Module Electronics (Pin 5)	-	-	± 100 pp	mV
t_A	Power supply rise time (0.1 V_{CC} to 0.9 V_{CC})		-	-	50	ms

a. The maximum input current on both the bus interface and the module electronics summed together.

Note: The values in the table above are valid for most modules. At the time of writing there is one exception, the Anybus-S Profinet IRT FO. Please contact HMS for further information.

As an extra precaution, a bulk capacitor can be added close to the module power supply:

Capacitor Type	Value
Cheramic	22uF / 6.3V
Electrolythe	100uF / 16V

D.2 Signal Characteristics

Symbol	Item		Min.	Typ.	Max.	Unit	Test Conditions
V_{OH}	Output High Voltage	D_0-D_7	2.4	-	-	V	$I_{OH} = -4.0 \text{ mA}$
		Tx	3.5	-	-	V	$I_{OH} = -1.0 \text{ mA}$
V_{OL}	Output Low Voltage	D_0-D_7	-	-	0.4	V	$I_{OL} = 4.0 \text{ mA}$
		BUSY, \overline{IRQ}	-	-	0.5	V	$I_{OL} = 16.0 \text{ mA}$
		Tx	-	-	0.4	V	$I_{OL} = 1.60 \text{ mA}$
V_{IH}	Input High Voltage	$A_0-A_{11}, D_0-D_7, \overline{CE}, \overline{OE}, R/\overline{W}$	2.2	-	$V_{CC} + 0.2$	V	-
		Rx	2.0	-	$V_{CC} + 0.2$	V	-
		RES	3.5	-	$V_{CC} + 0.2$	V	-
V_{IL}	Input Low Voltage	$A_0-A_{11}, D_0-D_7, \overline{CE}, \overline{OE}, R/\overline{W}$	-0.2	-	0.8	V	-
		Rx	-0.2	-	0.8	V	-
		RES	-0.2	-	0.8	V	-
I_{IX}	Input Load Current	$A_0-A_{11}, D_0-D_7, \overline{CE}, \overline{OE}, R/\overline{W}$	-5	-	+5	μA	$GND \leq V_I \leq V_{CC}$
I_{OZ}	Output Leakage Current	D_0-D_7	-5	-	+5	μA	$GND \leq V_O \leq V_{CC}$ (Output Disabled)
-	Pulse Width	RES	1.0	-	-	μs	-

Note: Pull up resistors are not considered in these characteristics.

D.3 PE & Shielding Recommendations

In order to achieve proper EMC behaviour, the module must be properly connected to protective earth in accordance with the fieldbus requirements.

The Anybus-S features three metal plated mounting holes, out of which two are used for extended electrical connection between the application and the module. One of these two holes are used for connection to protective earth (PE). This hole will from now on be referred to as 'the PE-hole'

If the housing of the application is of non conductive type, it is recommended to use PE-hole. However, if the housing of the application is conductive, there are a two options:

- **Protective Earth connection via PE-hole**

In this case, the fieldbus connector must be completely isolated from the application housing in order to avoid ground loops etc.

- **Protective Earth connection via fieldbus connector / application housing**

In this case, the PE-hole must be isolated from the application in order to avoid ground loops etc.

The application must provide support for these different options in order to ensure compatibility with all fieldbus systems. This issue becomes a bit more complex when using a fieldbus system that uses more than one fieldbus connector as these connectors may need to be treated differently. In these cases, it is recommended to isolate the fieldbus connectors from the application housing and use the PE-hole.

Contact HMS and/or consult each fieldbus specification for further information regarding PE/shielding requirements.

E. Environmental Specification

E.1 Temperature

Operating

+0 to +70 degrees Celsius

(Test performed according to IEC-68-2-1 and IEC 68-2-2.)

Non Operating

-15 to +85 degrees Celsius

(Test performed according to IEC-68-2-1 and IEC 68-2-2.)

E.2 Relative Humidity

The product is designed for a relative humidity of 5 to 95% non-condensing.

Test performed according to IEC 68-2-30.

E.3 EMC compliance

EMC pre-compliance testing has been conducted according to the *Electromagnetic Compatibility Directive 2004/108/EC*.

Details about what standards that have been used, can be found in the separate EMC pre-compliance documents, available for download for each product at www.anybus.com.

F. Conformance with Predefined Standards

F.1 Fieldbus Certification

All Anybus-S modules are pre-certified and found to comply with each fieldbus standard. Please note that although the module itself has been pre-certified, the final product may still require re-certification depending on the fieldbus standard.

This pre-certification is valid under the following conditions:

- Standard fieldbus connectors
- No fieldbus specific initialization parameters
- Non-modified device description file (i.e. ‘GSD’ or ‘EDS’)

If any of these conditions change, the module is not considered to be pre-certified and requires re-certification.

For more information, consult the fieldbus standard and/or contact HMS.

F.2 CE-Mark

Generally, most Anybus-S modules are certified according to the European CE standard unless otherwise stated. It is however important to note that although the Anybus-S itself is certified, the final product may still require re-certification depending on the application.

F.3 UL/cUL-Certificate

The Anybus-S modules are UL/cUL recognized for the US (NRAQ2) and Canada (NRAQ8) according to UL508, “Programmable Controller”.

G. Troubleshooting

The module does not exchange data

- Check the Anybus-S Watchdog LED. If the module does not flash green at 1hz, this means that the module is not initialised and therefore cannot exchange data.

Initialization troubles

- Verify that the responses to the initialization mailbox messages does not contain any error indications.
- The maximum I/O/Parameter data sizes may differ between different fieldbus systems. Verify that the initialization parameters suit the currently used Anybus-S version.
- If using the HW_CHK mailbox message - does the module respond? If not, the module has detected a hardware problem. The reason for the problem is indicated on the Anybus-S Watchdog LED.
- If using the 'Load from FLASH' mailbox command - ensure that the flash actually contains a valid mailbox sequence.

The driver example in this document does not work

- In most cases, the example can't work without modifications as it would require very specific conditions to be met. Instead, we have chosen to illustrate how the code should work generally rather than going into details on how to implement the code in a particular application.
- The example is provided for educational purposes and is far from complete. Several essential functions for initialization, error handling etc. has intentionally been left out and must be implemented by the user.

Common Handshaking Troubles, solutions

- Never write new commands to the Application Indication Register unless the module has responded to a previous command in the Anybus Indication Register first.
- Write to the Application Indication Register only when required. Unnecessary accesses to this register will only result in loss of processing power. Use locked requests if an area should be accessed as soon as possible.

Reset Related Problems

- If the application utilizes hardware reset, for example via a manual reset button, there is a risk of the interrupt line being held at a low level from the start. Therefore, certain precautions are needed to ensure proper functionality. See "Interrupt Line and Hardware Reset" on page 76.