



Ingenieurbüro Für Ic-Technologie  
Seit 1985 alles rund um ALTERA



## IFI Avalon CAN Module

---

### User Guide

Core Version: 2003.01  
Document Version: 2003.01 rev 1  
Document Date: march 2003



Ingenieurbüro Für Ic-Technologie  
Seit 1985 alles rund um ALTERA

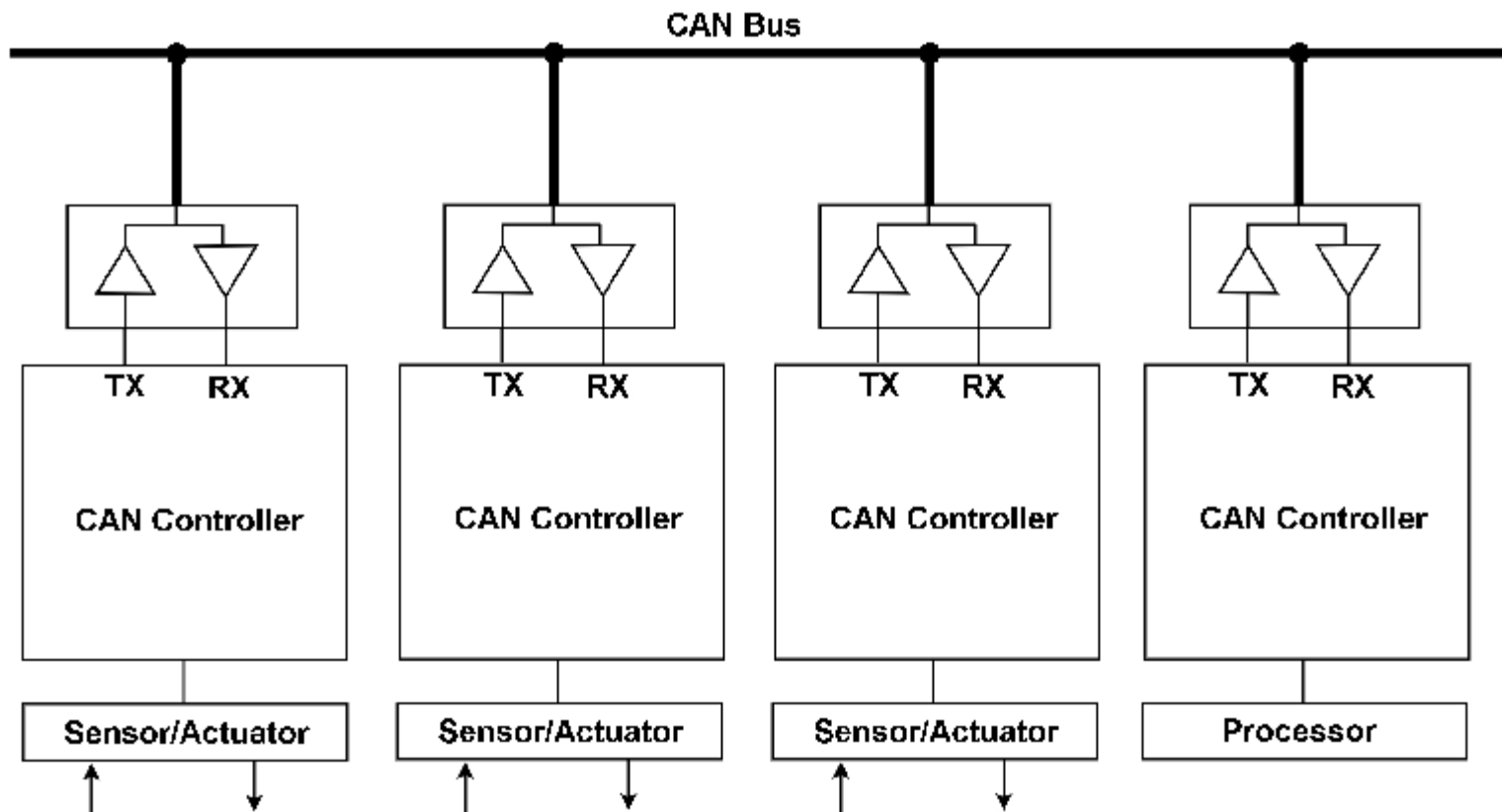
## Device Family Support:

**ACEX 1K™**  
**APEX™**  
**APEX™ II**  
**Cyclone™**  
**Stratix™**

## Features:

- n CAN 2.0B
- n 32 Messages Transmit Buffer (FIFO)
- n 32 Messages Receive Buffer (FIFO)
- n 64 Message Filters
- n NIOS Interface
- n SOPC-Builder ready
- n Optional: - Automatic BAUD-Rate Detection
- n OpenCore feature allows designers to instantiate and simulate designs prior to purchasing a license
- n OpenCorePlus feature allows additionally to test the design in hardware for a limited time (~ 1 hour)

**Application Example:**





Ingenieurbüro Für Ic-Technologie  
Seit 1985 alles rund um ALTERA

## CAN Messages:

Every CAN message consist of a certain number of bits that are divided into fields. There are fields like Arbitration Field, Data Field, CRC, End of Frame...

The Arbitration Field is different for CAN 2.0 A and CAN 2.0 B messages. It's a logical address with 11 bits for CAN 2.0 A and 29 bits for CAN 2.0 B. The lowest value is the highest priority = 0.

The Data Field contains the application data of the message with 0 to 64 bits (0 to 8 bytes).

With exception of the CRC delimiter, the ACK field and the EOF the bits are stuffed. That means, 5 consecutive bits with identical value are followed from a complementary bit.

Error Frame and the Overload Frame are of a fixed form and not coded with bit stuffing.

## Error Detection:

The error management unit is able to detect five different error types.

- n Bit Error
- n Bit Stuffing Error
- n CRC Error
- n Form Error
- n ACK Error

## Error Handling:

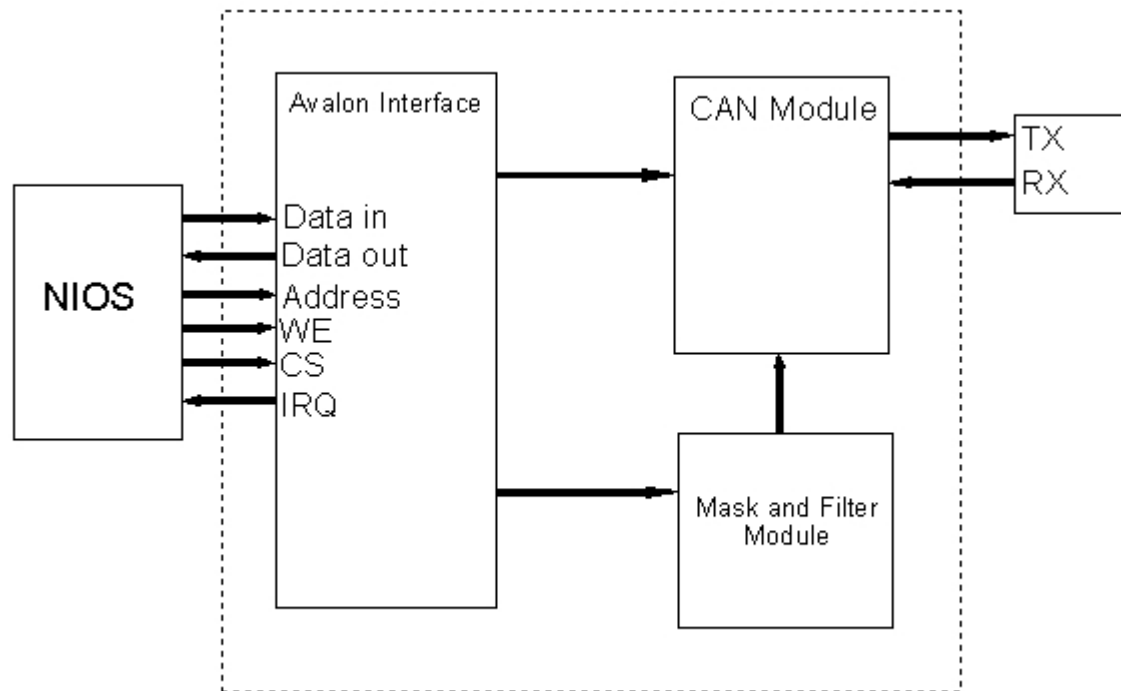
- n Error detected
- n Transmit of error frame
- n Message will be discarded
- n Error counters are incremented
- n Transmission will be repeated

### Error Limitation:

To prevent a permanently disturbed bus each CAN controller has three error states.

- n Error Active
- n Error Passive
- n Bus Off

### Block Diagram:





Ingenieurbüro Für Ic-Technologie  
Seit 1985 alles rund um ALTERA

The SOPC Builder Ready OpenCore Package contains all files required for plug-and-play integration of this core into Altera's SOPC Builder tool, allowing the user to easily evaluate the core within his Avalon-based system.

## Package Contents

The SOPC Builder Ready OpenCore package contains the following directories:

- ü Documentation:
  - This file
- ü ifi\_avalon\_can\_module:
  - SOPC Builder component directory
- ü License:
  - OpenCore evaluation license file
- ü reference\_design:
  - Reference design demonstrating the use of the core with the SOPC Builder design flow
- ü Symbol:
  - Quartus II symbol file for the core
- ü VHDL-simulation:
  - Precompiled simulation models for the core and testbench demonstrating the communication between 2 CAN-nodes, for use in the ModelSim software



Ingenieurbüro **F**ür **I**c-Technologie  
Seit 1985 alles rund um **ALTERA**

## Core Installation

1. Extract the contents of the zipped file to your hard drive.
2. Exit the SOPC Builder application if it is running.
3. Copy the directory <Core installation directory>\ifi\_avalon\_can\_module\ to <SOPC Builder installation directory>\components\.

You will now be able to add the core to your Nios system using the SOPC Builder.

## Licensing

Two different types of licenses are available for the core: an OpenCore evaluation license, and a full license.

### OpenCore Evaluation License

This package is shipped with an OpenCorePlus time limited evaluation license, <Core installation directory>\license\_eval.dat. When the FEATURE line from this license is appended to the user's Quartus II license file, the encrypted EDIF netlist file can be read into Quartus II and place and route can be performed. Evaluation licenses do not permit generation of programming files or gate-level simulation netlists. For more information on OpenCore evaluation, please see Altera Application Note 125: Evaluating AMPP and MegaCore Functions.

### Full License

A full license enabling the generation of programming files and gate-level simulation netlists will be shipped when the core is purchased by the user.



Ingenieurbüro Für Ic-Technologie  
Seit 1985 alles rund um ALTERA

## Integrating the Core with your System using SOPC Builder

This section contains instructions on the following:

- ü Adding the Core to your System
- ü Using ModelSim to Simulate the Core within your System
- ü Running the Reference Design

These instructions assume that the user is familiar with the Altera OpenCore evaluation process, Altera's Quartus II development software, and the Altera SOPC Builder tool. For more information on these prerequisites, please visit [www.altera.com](http://www.altera.com).

### Adding the Core to your System

1. Start Quartus II, version 2.2 SP1 or higher.
2. Create a new project using the New Project Wizard (File menu).
3. Launch SOPC Builder version 2.8 or higher from Quartus II (Tools menu).
4. Choose desired HDL and output file name. Please note that simulation models for the CAN Module core are available in VHDL only; therefore, you must choose VHDL for your SOPC Builder system if you wish to simulate in ModelSim Altera Edition.
5. Click "Next" to launch the SOPC Builder. The System Contents screen will appear. If the core was installed properly, you will see the core name under "Avalon Modules/Communication" on the left hand side of the screen. Next to the core name, you will also see a yellow icon, signifying that the OpenCore evaluation is installed.
6. Select the core by clicking on the core name, then click "Add" to add the core to your system. You should see the core instance appear in the components list on the right hand side of the screen.
7. Specify desired instance name, base address, and IRQ, if applicable.
8. Add additional components as required by your design.
9. Complete system generation as described in the Altera SOPC Builder documentation.



Ingenieurbüro Für Ic-Technologie  
Seit 1985 alles rund um ALTERA

## Using ModelSim to Simulate the Core within your System

1. Generate your system using the SOPC Builder.
2. Launch ModelSim or ModelSim Altera Edition, version 5.6a or higher, via the "Run ModelSim" button in SOPC Builder.
3. Type 's' to load the design files.
4. Type 'w' to add the appropriate waveforms to the wave window.
5. Type 'run 5 ms' to start the simulation.
6. For more details on simulation, please see Altera Application Note 189: Simulating Nios Embedded Processor Designs.

## Running the Reference Design

1. Start Quartus II, version 2.2 SP1 or higher.
2. Open the Quartus II project <Core installation directory>\reference\_design\standard\_32.quartus.
3. Launch SOPC Builder version 2.8 or higher from Quartus II (Tools menu).
4. The Nios CPU has been parameterized and added to the system for you, as have the program and data memories, UART, PIO, and the core itself.
5. Double-click on ext\_ram, click on the Simulation tab, and verify that the contents of the program memory will be compiled from <Core installation directory>\cpu\_sdk\src\hello\_canuart.c, which is a C file that performs writes and reads to and from the core.
6. Click on the "System Generation" tab and select the desired target device.
7. Click "Generate" to generate the SDK, HDL, and simulation files.
8. Launch ModelSim or ModelSim Altera Edition, version 5.6a or higher.
9. Type 's' to load the design files.
10. Type 'w' to add the appropriate waveforms to the wave window.
11. Type 'run 5 ms' to start the simulation.
12. Switch to the wave window. Note the presence of the Avalon bus signals in the window. The Avalon bus is the interface that Nios uses to communicate with the core. You should see transitions on the CHIPSELECT and READN/WRITEN lines, matching the read and write commands given in the hello\_canuart.c file.



Ingenieurbüro Für Ic-Technologie  
Seit 1985 alles rund um ALTERA

## Using ModelSim to Simulate the Communication between 2 CAN-Modules

1. Launch ModelSim or ModelSim Altera Edition, version 5.6a or higher.
2. Use FILE - CHANGE DIRECTORY to switch to the <Core installation directory>\VHDL-Simulation directory.
3. Use TOOL - EXECUTE MACRO to run SETUP\_SIM.do.
4. Type 's' to load the design files.
5. Type 'w' to add the appropriate waveforms to the wave window.
6. Type 'run 1 ms' to start the simulation.

This simulation uses a testbench named "nioscan\_tb.vhd". This file can be edited and modified.

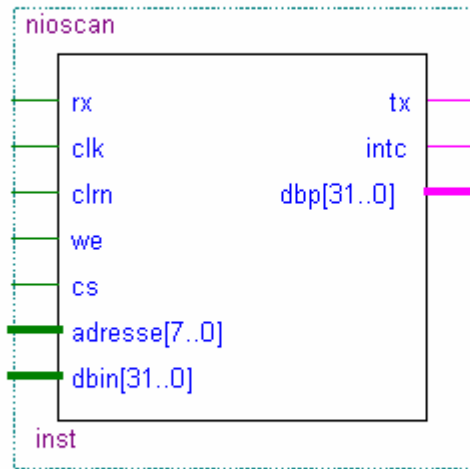
### Resource usage:

ACEX 1k	1550 LEs	6 ESBs
APEX	1440 LEs	6 ESBs
APEX II	1440 LEs	6 ESBs
CYCLONE	1250 LEs	3 * 4k Block
STRATIX	1290 LEs	3 * 4k Block



# Ingenieurbüro Für Ic-Technologie

Seit 1985 alles rund um ALTERA



Quartus Symbol



Ingenieurbüro Für Ic-Technologie  
Seit 1985 alles rund um ALTERA

Portname	Direction	Usage	Description
tx	output	External	Transmit from CAN
rx	input	External	Receive to CAN
clk	input	Internal	System clock
clrn	input	Internal	System reset (low active)
we	input	Internal	Write request (high active)
cs	input	Internal	Chip select (high active)
Adresse[7..0]	input	Internal	Address for read/write requests
Dbin[31..0]	input	Internal	Write data bus
Dbp[31..0]	output	Internal	Read data bus
intc	output	Internal	Interrupt request

## Contacting Technical Support

Although we have made every effort to ensure that this SOPC Builder Ready OpenCore Package works correctly, there might be problems that we have not encountered. If you have a question or problem that is not answered by the information provided in this README file, please contact the IP Vendor or Altera at one of the addresses below.

For questions about the core's features, functionality, and parameter settings please contact:

IFI Ingenieurbüro Für Ic-Technologie  
P. Riekert & F. Sprenger  
Kleiner Weg 3 -- 97877 Wertheim – Germany  
Phone: (+49)9342/96080  
E-Mail: ifi@ifi-pld.de  
<http://www.ifi-pld.de>



**Addressmap:**

Address 0 to 3 are acting like a FIFO. You can write up to 32 Transmitobjects into this FIFO

Address 0 must be written as the last word of a transmitobject

Bit	31	30	29	28	27	26	25	24
Byte 3								

Not used

Bit	23	22	21	20	19	18	17	16
Byte 2								

Not used

Bit	15	14	13	12	11	10	9	8
Byte 1 read			OBJ5	OBJ4	OBJ3	OBJ2	OBJ1	OBJ0

**OBJ[5..0] Object Number from Filter and Mask Module**

Bit	7	6	5	4	3	2	1	0
Byte 0 read				RTR	DLC3	DLC2	DLC1	DLC0
write				RTR	DLC3	DLC2	DLC1	DLC0

**Remote Transmit and Datalengthcode**



# Ingenieurbüro Für Ic-Technologie

Seit 1985 alles rund um ALTERA

Address 1 write = Transmitobject ---- read = Receiveobject

Bit	31	30	29	28	27	26	25	24
Byte 3 R/W			IDE	IDX 28	IDX 27	IDX 26	IDX 25	IDX 24

Bit	23	22	21	20	19	18	17	16
Byte 2 R/W	IDX 23	IDX 22	IDX 21	IDX 20	IDX 19	IDX 18	IDX 17	IDX 16

Bit	15	14	13	12	11	10	9	8
Byte 1 R/W	IDX 15	IDX 14	IDX 13	IDX 12	IDX 11	ID 10	ID 9	ID 8

Bit	7	6	5	4	3	2	1	0
Byte 0 R/W	ID 7	ID 6	ID 5	ID 4	ID 3	ID 2	ID 1	ID 0

**IDE : Enable extended Identifier**

**IDX : Extended Identifier**

**ID: Standard Identifier**

Address 2 Databyte4, Databyte3, Databyte2, Databyte1

Address 2 Databyte4, Databyte3, Databyte2, Databyte1

Address 3 Databyte8, Databyte7, Databyte6, Databyte5

Address 3 Databyte8, Databyte7, Databyte6, Databyte5

– write = Transmitter

– read = Receiver

– write = Transmitter

– read = Receiver



Address 4 read/write

Bit	31	30	29	28	27	26	25	24
Byte 3 read								
Byte 3 write	1= prescale set							Normal mode

Bit	23	22	21	20	19	18	17	16
Byte 2 read	Prescale7	Prescale6	Prescale5	Prescale4	Prescale3	Prescale2	Prescale1	Prescale0
Byte 2 write	Prescale7	Prescale6	Prescale5	Prescale4	Prescale3	Prescale2	Prescale1	Prescale0

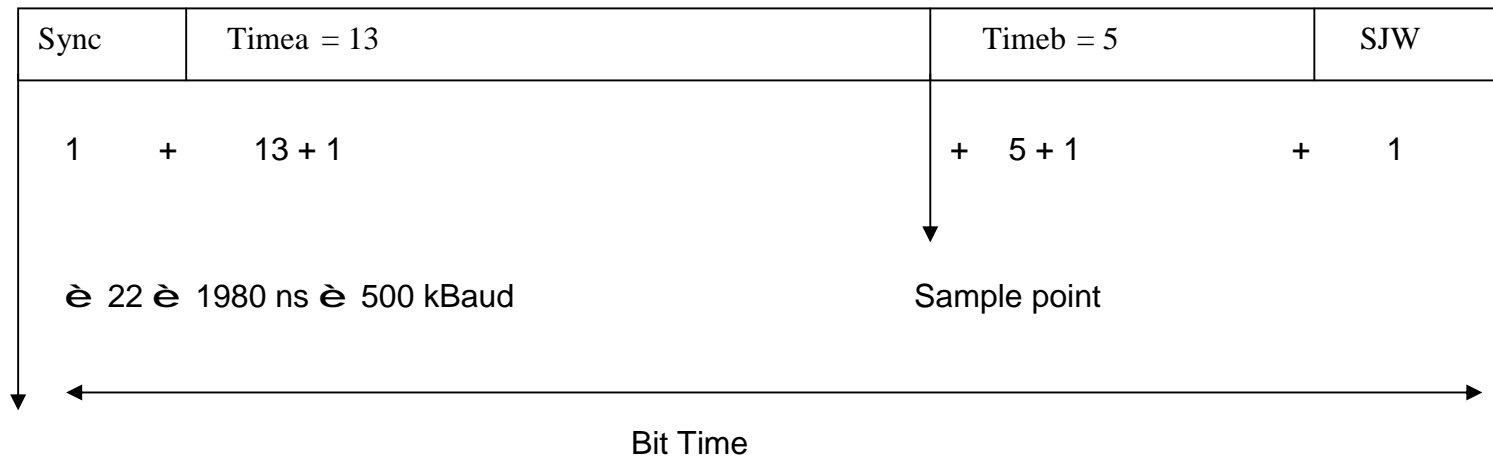
Bit	15	14	13	12	11	10	9	8
Byte 1 read				Timea4	Timea3	Timea2	Timea1	Timea0
Byte 1 write	1= timea set			Timea4	Timea3	Timea2	Timea1	Timea0

Bit	7	6	5	4	3	2	1	0
Byte 0 read				Timeb4	Timeb3	Timeb2	Timeb1	Timeb0
Byte 0 write	1= timeb set			Timeb4	Timeb3	Timeb2	Timeb1	Timeb0



The prescale can be adjusted from 1 to 255. For computing the resulting divisionfactor you have to add one.

Example: 33.33 MHz Clockfrequency and for the prescale the value 2 results in division by 3  $\Rightarrow$  90 ns for the Timesegments.



**Normal Mode has to be activated once to start the system or to restart the system after a busoff.**



Address 5 read/write

Bit	31	30	29	28	27	26	25	24
Byte 3 read			Int mask Rec Buffer full	Int mask Rec Buffer overflow	Int mask Rec_Buffer not empty	Int mask Transmit Buffer full	Int mask Transmit overflow	Int mask Transmit Bufferempty
Byte 3 write	1 = Intmask set		Int mask Rec Buffer full	Int_mask Rec Buffer overflow	Int_mask Rec_Buffer not empty	Int_mask Transmit Buffer full	Int_mask Transmit overflow	Int_mask Transmit Bufferempty

Interrupt mask bits

Bit	23	22	21	20	19	18	17	16
Byte 2 read								
Byte 2 write								

Bit	15	14	13	12	11	10	9	8
Byte 1 read								
Byte 1 write								

Bit	7	6	5	4	3	2	1	0
Byte 0 read							Int mask Error warn	Int mask busoff
Byte 0 write	1 = Error Intmask set						Int mask Error warn	Int mask busoff



Address 6 read/write

Bit	31	30	29	28	27	26	25	24
Byte 3 read			Rec Buffer full	Rec Buffer overflow	Rec_Buffer not empty	Transmit Buffer full	Transmit overflow	Transmit Bufferempty
Byte 3 write			reset Rec Buffer full	reset Rec Buffer overflow	reset Rec_Buffer not empty	reset Transmit Buffer full	reset Transmit overflow	reset Transmit Bufferempty

**Status**

Bit	23	22	21	20	19	18	17	16
Byte 2 read			Rec Buffer Pointer5	Rec Buffer Pointer4	Rec Buffer Pointer3	Rec Buffer Pointer2	Rec Buffer Pointer1	Rec Buffer Pointer0
Byte 2 write	Reset rec pointer							

**Receive Buffer Pointer**

Bit	15	14	13	12	11	10	9	8
Byte 1 read			trans Buffer Pointer5	trans Buffer Pointer4	trans Buffer Pointer3	trans Buffer Pointer2	trans Buffer Pointer1	trans Buffer Pointer0
Byte 1 write	Reset trans pointer							

**Transmit Buffer Pointer**

Bit	7	6	5	4	3	2	1	0
Byte 0 read					Error passive	Error active	Error warning	busoff
Byte 0 write	RecFifo next value							

**Error Status**



Address 7 read

Bit	31	30	29	28	27	26	25	24
Byte 3 read								
Byte 3 write								

Bit	23	22	21	20	19	18	17	16
Byte 2 read	RecError7	RecError6	RecError5	RecError4	RecError3	RecError2	RecError1	RecError0
Byte 2 write								

**Receive Error Counter value**

Bit	15	14	13	12	11	10	9	8
Byte 1 read								TransError8
Byte 1 write								

Bit	7	6	5	4	3	2	1	0
Byte 0 read	TransError7	TransError6	TransError5	TransError4	TransError3	TransError2	TransError1	TransError0
Byte 0 write								

**Transmit Error Counter value**



Address 8 to 127 not used

Address 128 to 255 are reserved for filter

Address 128

Bit	31	30	29	28	27	26	25	24
Byte 3 read write	1 = Valid			Maske28	Maske27	Maske26	Maske25	Maske24

Bit	23	22	21	20	19	18	17	16
Byte 2 read write	Maske22	Maske22	Maske21	Maske20	Maske19	Maske18	Maske17	Maske16

Bit	15	14	13	12	11	10	9	8
Byte 1 read write	Maske15	Maske14	Maske13	Maske12	Maske11	Maske10	Maske9	Maske8

Bit	7	6	5	4	3	2	1	0
Byte 0 read write	Maske7	Maske6	Maske5	Maske4	Maske3	Maske2	Maske1	Maske0

Valid = 1 ⇒ Use this mask

Valid = 0 ⇒ don't use this mask

Maskbit = 1 ⇒ ID-Bit compare

Maskbit = 0 ⇒ ID-Bit ignore



Address 129

Bit	31	30	29	28	27	26	25	24
Byte 3 read write	1 = Valid			ID-Bit28	ID-Bit27	ID-Bit26	ID-Bit25	ID-Bit24

Bit	23	22	21	20	19	18	17	16
Byte 2 read write	ID-Bit22	ID-Bit22	ID-Bit21	ID-Bit20	ID-Bit19	ID-Bit18	ID-Bit17	ID-Bit16

Bit	15	14	13	12	11	10	9	8
Byte 1 read write	ID-Bit15	ID-Bit14	ID-Bit13	ID-Bit12	ID-Bit11	ID-Bit10	ID-Bit9	ID-Bit8

Bit	7	6	5	4	3	2	1	0
Byte 0 read write	ID-Bit7	ID-Bit6	ID-Bit5	ID-Bit4	ID-Bit3	ID-Bit2	ID-Bit1	ID-Bit0

Valid = 1 ⇒ Use this ID

Valid = 0 ⇒ don't use this ID

ID-Bit 28 – 11 ⇒ extended Identifier

ID-Bit 10 – 0 ⇒ standard Identifier

Same for address 130 to 255



# Ingenieurbüro Für Ic-Technologie

Seit 1985 alles rund um ALTERA

Address	Contents		Object	Address
128	Valid	Mask0	1	Valid can switch on or off any object The object number with a match between the filter and a received message is readable on address 0
129	Valid	ID0		
130	Valid	Mask1	2	
131	Valid	ID1		
		.		
		.		
		.		
254	Valid	Mask63	64	
255	Valid	ID63		

maskbit	0	1	1	1	1
ldbit-filter	X	0	1	0	1
ldbit-received message	X	0	1	1	0
result	match	match	match	No match	No match