

PCAN MicroMod Library

V1.3.6

Table of Contents

1 Symbol Reference 1

1.1 Functions 1

1.1.1 AD_1000Hz	1
1.1.2 AD_Init	1
1.1.3 AD_Read()	2
1.1.4 AD_SetTimeConstant()	2
1.1.5 CAN_Init()	2
1.1.6 CAN_Offline	3
1.1.7 CAN_Online	3
1.1.8 CAN_Read	4
1.1.9 CAN_RegisterMsg()	4
1.1.10 CAN_RxQueueReset	4
1.1.11 CAN_Status	4
1.1.12 CAN_TxQueueReset	5
1.1.13 CAN_Write	5
1.1.14 CANRxISRCallBack()	5
1.1.15 EEprom_CloseWrite	5
1.1.16 EEprom_CloseWrite	6
1.1.17 EEprom_isvalid	6
1.1.18 EEprom_OpenRead	6
1.1.19 EEprom_OpenWrite	7
1.1.20 EEprom_Read()	7
1.1.21 EEprom_Write()	7
1.1.22 EnableExtendedID()	8
1.1.23 Get_DIN()	8
1.1.24 Get_DINByte	8
1.1.25 Get_DOUT()	9
1.1.26 Get_DOUTByte	9
1.1.27 Getch	9
1.1.28 GetFrequency()	10
1.1.29 GetFrequencyInit	10
1.1.30 Getch	10
1.1.31 GetTimer32	10
1.1.32 Hardware_Init	11
1.1.33 I2C_2_Init	11
1.1.34 I2C_2_MasterRead	11
1.1.35 I2C_2_MasterWrite	11
1.1.36 I2C_2_Start	12
1.1.37 I2C_2_Stop	12
1.1.38 I2C_Delay	12

1.1.39 I2C_Init	12
1.1.40 I2C_MasterRead()	13
1.1.41 I2C_MasterWrite()	13
1.1.42 I2C_Start	13
1.1.43 I2C_Stop	14
1.1.44 itoa()	14
1.1.45 itoa_hex()	14
1.1.46 LED_1000Hz	15
1.1.47 LED_SetSpeed()	15
1.1.48 Putch()	15
1.1.49 puts()	16
1.1.50 PWM_Init	16
1.1.51 Set_DOUT()	16
1.1.52 Set_DOUTByte	17
1.1.53 Set_DOUTByteMask	17
1.1.54 SetFrequency()	17
1.1.55 SetPWM()	18
1.1.56 Timer()	18
1.1.57 Timer_Init	18
1.1.58 Timer_Set()	19
1.1.59 Timer_SystemReset	19
1.1.60 Timer_SystemTime	19
1.1.61 UART_Init()	19
1.1.62 uitoa()	20

1.2 Types 20

1.2.1 CAN1_qentrytyp	20
1.2.2 uchar	21
1.2.3 ulong	21
1.2.4 word	21

1.3 Variables 21

1.3.1 Variables	21
1.3.2 CAN_BUFF_DATA	22
1.3.3 CAN_BUFF_FORMAT	22
1.3.4 CAN_BUFF_ID	22
1.3.5 CAN_BUFF_LEN	22

1.4 Macros 22

1.4.1 __I2C2H__	23
1.4.2 CAN_BAUD_100K	23
1.4.3 CAN_BAUD_10K	23
1.4.4 CAN_BAUD_125K	23
1.4.5 CAN_BAUD_1M	23
1.4.6 CAN_BAUD_20K	24

1.4.7 CAN_BAUD_250K	24
1.4.8 CAN_BAUD_2M	24
1.4.9 CAN_BAUD_33K3	24
1.4.10 CAN_BAUD_47K6	25
1.4.11 CAN_BAUD_500K	25
1.4.12 CAN_BAUD_50K	25
1.4.13 CAN_BAUD_83K3	25
1.4.14 CAN_BAUD_95K2	25
1.4.15 CAN_ERR_BUSOFF	26
1.4.16 CAN_ERR_OK	26
1.4.17 CAN_ERR_OVERRUN	26
1.4.18 CAN_ERR_QOVERRUN	26
1.4.19 CAN_ERR_QRCVEMPTY	27
1.4.20 CAN_ERR_QXMTFULL	27
1.4.21 CAN_ERR_XMTFULL	27
1.4.22 CAN_EXT	27
1.4.23 CAN_RTR	27
1.4.24 CAN_SETBUSOFF	28
1.4.25 CAN_SETBUSON	28
1.4.26 NULL	28
1.4.27 SER_BAUD_19200	28
1.4.28 SER_BAUD_38400	29
1.4.29 SER_BAUD_76800	29
1.4.30 SER_BAUD_9600	29

1.5 Files 29

1.5.1 ADI.H	29
1.5.2 can.h	30
1.5.3 compiler.h	31
1.5.4 eeprom.h	31
1.5.5 hardware.h	31
1.5.6 I2C.H	32
1.5.7 I2C_2.H	32
1.5.8 pwm.h	32
1.5.9 timer.h	32
1.5.10 uart.h	33

2 Index 34

PCAN MicroMod Library V1.3.6

1 Symbol Reference

1.1 Functions

1.1.1 AD_1000Hz

```
void AD_1000Hz(void);
```

File

ADI.H (see page 29)

Description

This function should be called every 1ms to read in all analogue channels and perform a lowpass function, if enabled.

1.1.2 AD_Init

```
void AD_Init(void);
```

File

ADI.H (see page 29)

Description

Initializes AD-converter.

Example

```
void main()
{
    word (see page 21) value;
    AD_Init();
    AD_SetTimeConstant (see page 2)(1,30);
    value=AD_Read (see page 2)(1);
};
```

1.1.3 AD_Read()

```
word AD_Read(uchar chan);
```

File

ADI.H (see page 29)

Parameter

Parameter	Description
uchar chan	uchar (see page 21) channel-number, range is 0..7

Returns

word (see page 21): Value 0..1023

Description

Reads an analog channel.

1.1.4 AD_SetTimeConstant()

```
void AD_SetTimeConstant(uchar channel, word tau);
```

File

ADI.H (see page 29)

Parameter

Parameter	Description
word tau	word (see page 21) time constant in ms, range is 0(off)..32767
chan	uchar (see page 21) channel-number, range is 0..7

Description

Sets a time for lowpass function of the selected channel.

1.1.5 CAN_Init()

```
void CAN_Init(word baudrate);
```

File

can.h (see page 30)

Parameter

Parameter	Description
Baudrate	common Baudrates are defined in the header file

Returns

none

Description

Initializes the CAN-Bus with a given Baudrate

Example

```
void main()
```

```
{
CAN_Init(CAN_BAUD_500k);
CAN_RegisterMsg (☐ see page 4)(0);
CAN_BUFF_ID (☐ see page 22)=0x100;
CAN_BUFF_LEN (☐ see page 22)=4;
CAN_BUFF_DATA (☐ see page 22)[0]=0x12;
CAN_BUFF_DATA (☐ see page 22)[1]=0x34;
CAN_BUFF_DATA (☐ see page 22)[2]=0x45;
CAN_BUFF_DATA (☐ see page 22)[3]=0x67;
CAN_BUFF_FORMAT (☐ see page 22)=0;
CAN_Write (☐ see page 5()); // Send a CAN-Message
while(1)
{
if(CAN_Read (☐ see page 4)()==CAN_ERR_OK (☐ see page 26)) // Wait for a CAN-Message with any ID
{
CAN_BUFF_ID (☐ see page 22)++;
CAN_Write (☐ see page 5()); // echo Message
};
}
```

1.1.6 CAN_Offline

```
void CAN_Offline();
```

File

can.h (☐ see page 30)

Returns

none

Description

Sets CAN-Controller offline. No Messages are received or transmitted anymore.

1.1.7 CAN_Online

```
void CAN_Online();
```

File

can.h (☐ see page 30)

Returns

none

Description

Sets CAN-Controller online. CAN-Controller can transmit and receive Messages.

1.1.8 CAN_Read

```
uchar CAN_Read(void);
```

File

can.h (see page 30)

Returns

CAN_ERR_OK (see page 26): Global Variables (see page 21) CAN_BUFF_ID (see page 22), CAN_BUFF_LEN (see page 22) and CAN_BUFF_DATA (see page 22)[8] contain the received message

CAN_ERR_QRCVEMPTY (see page 27) : no message

Description

Reads a received CAN-Message from the internal Receive-Queue.

1.1.9 CAN_RegisterMsg()

```
void CAN_RegisterMsg(word id);
```

File

can.h (see page 30)

Returns

none

Description

Sets the Acceptance filter for the given Identifier

1.1.10 CAN_RxQueueReset

```
void CAN_RxQueueReset();
```

File

can.h (see page 30)

Description

Clears the CAN Receive Queue.

1.1.11 CAN_Status

```
byte CAN_Status(void);
```

File

can.h (see page 30)

Returns

CAN_ERR_OK (see page 26), CAN_ERR_BUSLIGHT, CAN_ERR_BUSHEAVY, CAN_ERR_BUSOFF (see page 26)

Description

Returns Status of CAN-Controller

1.1.12 CAN_TxQueueReset

```
void CAN_TxQueueReset();
```

File

can.h ([↗](#) see page 30)

Description

Clears the CAN Transmit Queue.

1.1.13 CAN_Write

```
int CAN_Write(void);
```

File

can.h ([↗](#) see page 30)

Returns

CAN_ERR_OK ([↗](#) see page 26), CAN_ERR_QXMTFULL ([↗](#) see page 27)

Description

Writes a CAN-Message from the global Variables ([↗](#) see page 21) CAN_BUFF_ID ([↗](#) see page 22), CAN_BUFF_LEN ([↗](#) see page 22), CAN_BUFF_DATA ([↗](#) see page 22)[8], CAN_BUFF_FORMAT ([↗](#) see page 22) into Transmit-Queue

1.1.14 CANRxISRCallback()

```
int CANRxISRCallback(CANI_qentrytyp* p);
```

File

can.h ([↗](#) see page 30)

Returns

0: put Message into CAN Receive Queue

1: do not put Message into CAN Receive Queue

Description

The Function is called from CAN Receive Interrupt function for each received CAN-Message to allow a faster response time for a received message. The definition of this function is part of the User Application. If this function returns 0 the CAN-Message is put into the standard receive queue and can be read with CAN_Read ([↗](#) see page 4)().

1.1.15 EEprom_CloseWrite

```
void EEprom_CloseRead(void);
```

File

eeeprom.h (see page 31)

Returns

none

Description

Finishes read access.

1.1.16 EEprom_CloseWrite

```
void EEprom_CloseWrite(void);
```

File

eeeprom.h (see page 31)

Returns

none

Description

Finishes write access, writes checksum to protect data.

1.1.17 EEprom_isvalid

```
byte EEprom_isvalid(void);
```

File

eeeprom.h (see page 31)

Returns

0: Checksum is incorrect. 1: Checksum is correct.

Description

Checks the consistency of the EEprom content.

1.1.18 EEprom_OpenRead

```
void EEprom_OpenRead(void);
```

File

eeeprom.h (see page 31)

Returns

none

Description

Open the EEprom for read access. To use the Eeprom I2C_Init (see page 12) must be called previously. The read-Pointer is set to the start of the EEprom.

1.1.19 EEPROM_OpenWrite

```
void EEPROM_OpenWrite(void);
```

File

EEPROM.h (see page 31)

Returns

none

Description

Open the EEPROM for write access. To use the EEPROM_I2C_Init (see page 12) must be called previously. The Write-Pointer is set to the start of the EEPROM.

1.1.20 EEPROM_Read()

```
void EEPROM_Read(uchar* dat, word len);
```

File

EEPROM.h (see page 31)

Parameter

Parameter	Description
uchar* dat	Destination Pointer
word len	Number of Bytes

Returns

none

Description

Reads 'len' data from EEPROM. Data are copied into memory that starts at address'dat'.

1.1.21 EEPROM_Write()

```
void EEPROM_Write(uchar* dat, word len);
```

File

EEPROM.h (see page 31)

Parameter

Parameter	Description
uchar* dat	Source Pointer
word len	Number of Bytes

Returns

none

Description

Writes 'len' data into EEPROM. Data are copied from memory that starts at address'dat'.

1.1.22 EnableExtendedID()

```
void EnableExtendedID(int n);
```

File

can.h (see page 30)

Parameter

Parameter	Description
number	number of reception slots that can receive extended identifiers. 0: only standard identifiers ... 7: only extended identifiers

Returns

none

Description

Enables reception of CAN-Messages with extended Identifier. Per default reception of extended Identifier is disabled (0).

1.1.23 Get_DIN()

```
uchar Get_DIN(uchar chan);
```

File

hardware.h (see page 31)

Parameter

Parameter	Description
uchar chan	number of digital Channel, 0..7

Returns

Status of digital Input.

Description

Reads one digital Input.

1.1.24 Get_DINByte

```
uchar Get_DINByte(void);
```

File

hardware.h (see page 31)

Returns

Status of all 8 digital Inputs.

bit 0: Status of Input 0

...

bit 7: Status of Input 7

Description

Reads all 8 digital Inputs.

1.1.25 Get_DOUT()

```
uchar Get_DOUT(uchar chan);
```

File

hardware.h (🔗 see page 31)

Parameter

Parameter	Description
uchar chan	number of digital Channel, 0..7

Returns

Status of digital Output.

Description

Reads one digital Output.

1.1.26 Get_DOUTByte

```
uchar Get_DOUTByte(void);
```

File

hardware.h (🔗 see page 31)

Returns

Status of all 8 digital Outputs.

bit 0: Status of Input 0

...

bit 7: Status of Input 7

Description

Reads all 8 digital Outputs.

1.1.27 Getch

```
char Getch(void);
```

File

uart.h (🔗 see page 33)

Returns

-1: Error. Read karakter

Description

Reads a karakter from the UART. Waits until a character is received

1.1.28 GetFrequency()

```
word GetFrequency(byte chan);
```

File

pwm.h (see page 32)

Parameter

Parameter	Description
byte chan	channel 0..3

Returns

frequency

Description

Reads the current frequency signal. This is intended to measure low frequency signals (0..2500Hz)! Higher frequencies will decrease the overall performance of the module.

1.1.29 GetFrequencyInit

```
void GetFrequencyInit(void);
```

File

pwm.h (see page 32)

Returns

none

Description

Initializes Frequency Measurement.

1.1.30 Getch

```
char GetKey(void);
```

File

uart.h (see page 33)

Returns

-1: Error, 0: no character available Read charakter.

Description

Reads a charakter from the UART. Returns 0 if no character is available.

1.1.31 GetTimer32

```
unsigned long int GetTimer32(void);
```

File

pwm.h (see page 32)

Returns

Number of Milliseconds.

Description

Returns the number of Milliseconds since Power Up.

1.1.32 Hardware_Init

```
void Hardware_Init(void);
```

File

hardware.h (see page 31)

Returns

none

Description

Initializes the Hardware (Watchdog, directions,...)

1.1.33 I2C_2_Init

```
uchar I2C_2_Init(void);
```

File

I2C_2.H (see page 32)

Description

This is function I2C_2_Init.

1.1.34 I2C_2_MasterRead

```
uchar I2C_2_MasterRead(uchar ack);
```

File

I2C_2.H (see page 32)

Description

This is function I2C_2_MasterRead.

1.1.35 I2C_2_MasterWrite

```
uchar I2C_2_MasterWrite(uchar input_byte);
```

File

I2C_2.H (see page 32)

Description

This is function I2C_2_MasterWrite.

1.1.36 I2C_2_Start

```
void I2C_2_Start(void);
```

File

I2C_2.H ([↗](#) see page 32)

Description

This is function I2C_2_Start.

1.1.37 I2C_2_Stop

```
uchar I2C_2_Stop(void);
```

File

I2C_2.H ([↗](#) see page 32)

Description

This is function I2C_2_Stop.

1.1.38 I2C_Delay

```
void I2C_Delay(long count);
```

File

I2C.H ([↗](#) see page 32)

Description

This is function I2C_Delay.

1.1.39 I2C_Init

```
uchar I2C_Init(void);
```

File

I2C.H ([↗](#) see page 32)

Returns

none

Description

Initializes the I2C-Interface. This is used by the On-Board EEPROM.

1.1.40 I2C_MasterRead()

```
uchar I2C_MasterRead(uchar ack);
```

File

I2C.H (🔗 see page 32)

Parameter

Parameter	Description
uchar ack	1 generate no acknowledge 0: generate acknowledge

Returns

Data from I2C-Bus.

Description

Reads only Byte from the I2C-Bus.

1.1.41 I2C_MasterWrite()

```
uchar I2C_MasterWrite(uchar input_byte);
```

File

I2C.H (🔗 see page 32)

Parameter

Parameter	Description
uchar input_byte	Byte that is written to the I2C_Bus.

Returns

0: acknowledge-error

1: write successful

Description

Writes only Byte to the I2C-Bus.

1.1.42 I2C_Start

```
void I2C_Start(void);
```

File

I2C.H (🔗 see page 32)

Returns

none

Description

Generates Start-Sequence for I2C-Bus.

1.1.43 I2C_Stop

```
uchar I2C_Stop(void);
```

File

I2C.H (🔗 see page 32)

Returns

none

Description

Generates Stop-Sequence for I2C-Bus.

1.1.44 itoa()

```
char * itoa(int i);
```

File

uart.h (🔗 see page 33)

Parameter

Parameter	Description
int i	integer number.

Returns

Pointer to the String.

Description

Converts an integer into a String.

1.1.45 itoa_hex()

```
char * itoa_hex(unsigned int i);
```

File

uart.h (🔗 see page 33)

Parameter

Parameter	Description
unsigned int i	unsigned integer number.

Returns

Pointer to the String.

Description

Converts an integer into a String. Number base is 16.

1.1.46 LED_1000Hz

```
void LED_1000Hz(void);
```

File

hardware.h (see page 31)

Returns

none

Description

Call this function every 1ms for flashing.

1.1.47 LED_SetSpeed()

```
void LED_SetSpeed(byte);
```

File

hardware.h (see page 31)

Parameter

Parameter	Description
speed	0..4

Returns

none

Description

Call this function to set the flashing frequency of the LED.

1.1.48 Putch()

```
void Putch(char ch);
```

File

uart.h (see page 33)

Parameter

Parameter	Description
char ch	charakter to write.

Returns

none

Description

Writes a karakter to the UART.

1.1.49 puts()

```
void puts(const char * Name2);
```

File

uart.h (see page 33)

Parameter

Parameter	Description
const char * Name2	Pointer to a 0-terminated String.

Returns

none

Description

Writes a String to the UART.

1.1.50 PWM_Init

```
void PWM_Init(void);
```

File

pwm.h (see page 32)

Returns

none

Description

Initializes PWM and Frequency output.

1.1.51 Set_DOUT()

```
void Set_DOUT(uchar chan, uchar val);
```

File

hardware.h (see page 31)

Parameter

Parameter	Description
uchar chan	Number of digital Channel, 0..7
uchar val	0: sets output inactive 1: sets output active

Returns

none

Description

Sets one digital Output.

1.1.52 Set_DOUTByte

```
void Set_DOUTByte(uchar val);
```

File

hardware.h (see page 31)

Returns

none

Description

Sets all 8 digital Outputs.

1.1.53 Set_DOUTByteMask

```
void Set_DOUTByteMask(uchar mask, uchar val);
```

File

hardware.h (see page 31)

Parameter

Parameter	Description
uchar mask	defines which Outputs to modify.
uchar val	bit 0: Status of Output 0 ... bit 7: Status of Output 7

Returns

none

Description

Sets all digital Outputs where Bits in Parameter 'mask' is 1.

1.1.54 SetFrequency()

```
void SetFrequency(uchar chan, unsigned int frequency);
```

File

pwm.h (see page 32)

Parameter

Parameter	Description
uchar chan	0, 2
unsigned int frequency	1..10000

Returns

none

Description

Enables Frequency-Output with given frequency and 50% duty cycle. The MicroMod does only support two Frequency outputs at the same time. If channel 0 (2) is used channel 1 (3) is disabled.

1.1.55 SetPWM()

```
void SetPWM(uchar chan, uchar ratio, unsigned int frequency);
```

File

pwm.h (🔗 see page 32)

Parameter

Parameter	Description
uchar chan	0..3
uchar ratio	duty cycle 0..255 0=0%, 255=100%
unsigned int frequency	32..100, 4000..10000

Returns

none

Description

Enables PWM-Output with given frequency and duty cycle. The frequency of Channel 0/1 and 2/3 must be equal.

1.1.56 Timer()

```
word Timer(uchar nr);
```

File

timer.h (🔗 see page 32)

Parameter

Parameter	Description
uchar nr	number of Timer 0..3

Returns

Remaining milliseconds.

Description

Reads back a Timer. The Timer is decremented every Millisecond. If the function returns 0 the Timer is stopped.

1.1.57 Timer_Init

```
void Timer_Init(void);
```

File

timer.h (🔗 see page 32)

Returns

none

Description

Initializes the Timer (🔗 see page 18).

1.1.58 Timer_Set()

```
void Timer_Set(uchar nr, word ms);
```

File

timer.h (see page 32)

Parameter

Parameter	Description
uchar nr	number of Timer (see page 18) 0..3
word ms	Time in Milliseconds

Returns

none

Description

Sets a downcounting Timer (see page 18) to a given value.

1.1.59 Timer_SystemReset

```
void Timer_SystemReset(void);
```

File

timer.h (see page 32)

Returns

none

Description

Resets a global Timer (see page 18) that starts counting Milliseconds at Power on.

1.1.60 Timer_SystemTime

```
unsigned long Timer_SystemTime(void);
```

File

timer.h (see page 32)

Returns

Number of Milliseconds since Power on.

Description

Reads back a global Timer (see page 18) that starts counting Milliseconds at Power on.

1.1.61 UART_Init()

```
void UART_Init(char baud);
```

File

uart.h (🔗 see page 33)

Parameter

Parameter	Description
baudrate	0: 76800 Baud 1: 38400 Baud 2: 19200 Baud 3: 9600 Baud

Returns

none

Description

Initializes the UARD with a given Baudrate.

1.1.62 uitoa()

```
char * uitoa(unsigned int i);
```

File

uart.h (🔗 see page 33)

Parameter

Parameter	Description
unsigned int i	unsigned integer number.

Returns

Pointer to the String.

Description

Converts an unsigned integer into a String.

1.2 Types

1.2.1 CANI_qentrytyp

```
typedef struct {  
    long int ID;  
    word LEN_ERR;  
    uchar DATA[8];  
    uchar FORMAT;  
} CANI_qentrytyp;
```

File

can.h (🔗 see page 30)

Members

Members	Description
long int ID;	Identifier of CAN-Message, 0x000...0x7FF for Standard Identifier, 0x00000000-0x1FFFFFFF for Extended Identifier

word LEN_ERR;	Length of CAN-Message 0..8
uchar DATA[8];	Data-Array
uchar FORMAT;	Bit 0: 0=DataFrame / 1=RTR-Frame Bit 1: 0=11bit ID / 1=29 bit ID

Description

Eine im Queue gepufferte CAN-Message

1.2.2 uchar

```
typedef unsigned char uchar;
```

File

compiler.h (🔗 see page 31)

Description

This is type uchar.

1.2.3 ulong

```
typedef unsigned long ulong;
```

File

compiler.h (🔗 see page 31)

Description

This is type ulong.

1.2.4 word

```
typedef unsigned int word;
```

File

compiler.h (🔗 see page 31)

Description

neu

1.3 Variables

1.3.1 Variables

1.3.2 CAN_BUFF_DATA

```
uchar CAN_BUFF_DATA[8];
```

File

can.h (see page 30)

Description

This is variable CAN_BUFF_DATA.

1.3.3 CAN_BUFF_FORMAT

```
uchar CAN_BUFF_FORMAT;
```

File

can.h (see page 30)

Description

This is variable CAN_BUFF_FORMAT.

1.3.4 CAN_BUFF_ID

```
unsigned long CAN_BUFF_ID;
```

File

can.h (see page 30)

Description

This is variable CAN_BUFF_ID.

1.3.5 CAN_BUFF_LEN

```
uchar CAN_BUFF_LEN;
```

File

can.h (see page 30)

Description

This is variable CAN_BUFF_LEN.

1.4 Macros

1.4.1 __I2C2H__

```
#define __I2C2H__
```

File

I2C_2.H (see page 32)

Description

Schutz gegen mehrfaches #include

1.4.2 CAN_BAUD_100K

```
#define CAN_BAUD_100K 0x4D07u
```

File

can.h (see page 30)

Description

This is macro CAN_BAUD_100K.

1.4.3 CAN_BAUD_10K

```
#define CAN_BAUD_10K 0x7FFFu
```

File

can.h (see page 30)

Description

This is macro CAN_BAUD_10K.

1.4.4 CAN_BAUD_125K

```
#define CAN_BAUD_125K 0x3A07u
```

File

can.h (see page 30)

Description

This is macro CAN_BAUD_125K.

1.4.5 CAN_BAUD_1M

```
#define CAN_BAUD_1M 0x1C00u
```

File

can.h (see page 30)

Description

This is macro CAN_BAUD_1M.

1.4.6 CAN_BAUD_20K

```
#define CAN_BAUD_20K 0x4D27u
```

File

can.h (see page 30)

Description

This is macro CAN_BAUD_20K.

1.4.7 CAN_BAUD_250K

```
#define CAN_BAUD_250K 0x1C03u
```

File

can.h (see page 30)

Description

This is macro CAN_BAUD_250K.

1.4.8 CAN_BAUD_2M

```
#define CAN_BAUD_2M 0x1400u
```

File

can.h (see page 30)

Description

This is macro CAN_BAUD_2M.

1.4.9 CAN_BAUD_33K3

```
#define CAN_BAUD_33K3 0x581Du
```

File

can.h (see page 30)

Description

This is macro CAN_BAUD_33K3.

1.4.10 CAN_BAUD_47K6

```
#define CAN_BAUD_47K6 0x1429u
```

File

can.h (see page 30)

Description

This is macro CAN_BAUD_47K6.

1.4.11 CAN_BAUD_500K

```
#define CAN_BAUD_500K 0x1C01u
```

File

can.h (see page 30)

Description

This is macro CAN_BAUD_500K.

1.4.12 CAN_BAUD_50K

```
#define CAN_BAUD_50K 0x4D0fu
```

File

can.h (see page 30)

Description

This is macro CAN_BAUD_50K.

1.4.13 CAN_BAUD_83K3

```
#define CAN_BAUD_83K3 0x1417u
```

File

can.h (see page 30)

Description

This is macro CAN_BAUD_83K3.

1.4.14 CAN_BAUD_95K2

```
#define CAN_BAUD_95K2 0x1414u
```

File

can.h (🔗 see page 30)

Description

This is macro CAN_BAUD_95K2.

1.4.15 CAN_ERR_BUSOFF

```
#define CAN_ERR_BUSOFF 0x10
```

File

can.h (🔗 see page 30)

Description

Busfehler: CAN_Controller ging 'Bus-Off'

1.4.16 CAN_ERR_OK

```
#define CAN_ERR_OK 0x00
```

File

can.h (🔗 see page 30)

Description

Fehlerzustaende

1.4.17 CAN_ERR_OVERRUN

```
#define CAN_ERR_OVERRUN 0x02
```

File

can.h (🔗 see page 30)

Description

CAN-Controller wurde zu spaet gelesen

1.4.18 CAN_ERR_QOVERRUN

```
#define CAN_ERR_QOVERRUN 0x40
```

File

can.h (🔗 see page 30)

Description

RcvQueue wurde zu spaet gelesen

1.4.19 CAN_ERR_QRCVEMPTY

```
#define CAN_ERR_QRCVEMPTY 0x20
```

File

can.h (see page 30)

Description

RcvQueue ist leergelesen

1.4.20 CAN_ERR_QXMTFULL

```
#define CAN_ERR_QXMTFULL 0x80
```

File

can.h (see page 30)

Description

Sendequeue ist voll

1.4.21 CAN_ERR_XMTFULL

```
#define CAN_ERR_XMTFULL 0x01
```

File

can.h (see page 30)

Description

Sendepuffer im Controller ist voll

1.4.22 CAN_EXT

```
#define CAN_EXT 0x02
```

File

can.h (see page 30)

Description

for Extended CAN-ID

1.4.23 CAN_RTR

```
#define CAN_RTR 0x01
```

File

can.h (🔗 see page 30)

Description

for RTR

1.4.24 CAN_SETBUSOFF

```
#define CAN_SETBUSOFF CSR_HALT=1;
```

File

can.h (🔗 see page 30)

Description

This is macro CAN_SETBUSOFF.

1.4.25 CAN_SETBUSON

```
#define CAN_SETBUSON CSR_HALT=0;
```

File

can.h (🔗 see page 30)

Description

This is macro CAN_SETBUSON.

1.4.26 NULL

```
#define NULL ((void*)0)
```

File

compiler.h (🔗 see page 31)

Description

This is macro NULL.

1.4.27 SER_BAUD_19200

```
#define SER_BAUD_19200 2
```

File

uart.h (🔗 see page 33)

Description

This is macro SER_BAUD_19200.

1.4.28 SER_BAUD_38400

```
#define SER_BAUD_38400 1
```

File

uart.h (see page 33)

Description

This is macro SER_BAUD_38400.

1.4.29 SER_BAUD_76800

```
#define SER_BAUD_76800 0
```

File

uart.h (see page 33)

Description

This is macro SER_BAUD_76800.

1.4.30 SER_BAUD_9600

```
#define SER_BAUD_9600 3
```

File

uart.h (see page 33)

Description

This is macro SER_BAUD_9600.

1.5 Files

1.5.1 ADI.H

This is file ADI.H.

Functions

Function	Description
AD_1000Hz (see page 1)	This function should be called every 1ms to read in all analogue channels and perform a lowpass function, if enabled.
AD_Init (see page 1)	Initializes AD-converter.
AD_Read (see page 2)	Reads an analog channel.
AD_SetTimeConstant (see page 2)	Sets a time for lowpass function of the selected channel.

1.5.2 can.h

This is file can.h.

Functions

Function	Description
CAN_Init (↗ see page 2)	Initializes the CAN-Bus with a given Baudrate
CAN_Offline (↗ see page 3)	Sets CAN-Controller offline. No Messages are received or transmitted anymore.
CAN_Online (↗ see page 3)	Sets CAN-Controller online. CAN-Controller can transmit and receive Messages.
CAN_Read (↗ see page 4)	Reads a received CAN-Message from the internal Receive-Queue.
CAN_RegisterMsg (↗ see page 4)	Sets the Acceptance filter for the given Identifier
CAN_RxQueueReset (↗ see page 4)	Clears the CAN Receive Queue.
CAN_Status (↗ see page 4)	Returns Status of CAN-Controller
CAN_TxQueueReset (↗ see page 5)	Clears the CAN Transmit Queue.
CAN_Write (↗ see page 5)	Writes a CAN-Message from the global Variables (↗ see page 21) CAN_BUFF_ID (↗ see page 22), CAN_BUFF_LEN (↗ see page 22), CAN_BUFF_DATA (↗ see page 22)[8], CAN_BUFF_FORMAT (↗ see page 22) into Transmit-Queue
CANRxISRCallBack (↗ see page 5)	The Function is called from CAN Receive Interrupt function for each received CAN-Message to allow a faster response time for a received message. The definition of this function is part of the User Application. If this function returns 0 the CAN-Message is put into the standard receive queue and can be read with CAN_Read (↗ see page 4).
EnableExtendedID (↗ see page 8)	Enables reception of CAN-Messages with extended Identifier. Per default reception of extended Identifier is disabled (0).

Macros

Macro	Description
CAN_BAUD_100K (↗ see page 23)	This is macro CAN_BAUD_100K.
CAN_BAUD_10K (↗ see page 23)	This is macro CAN_BAUD_10K.
CAN_BAUD_125K (↗ see page 23)	This is macro CAN_BAUD_125K.
CAN_BAUD_1M (↗ see page 23)	This is macro CAN_BAUD_1M.
CAN_BAUD_20K (↗ see page 24)	This is macro CAN_BAUD_20K.
CAN_BAUD_250K (↗ see page 24)	This is macro CAN_BAUD_250K.
CAN_BAUD_2M (↗ see page 24)	This is macro CAN_BAUD_2M.
CAN_BAUD_33K3 (↗ see page 24)	This is macro CAN_BAUD_33K3.
CAN_BAUD_47K6 (↗ see page 25)	This is macro CAN_BAUD_47K6.
CAN_BAUD_500K (↗ see page 25)	This is macro CAN_BAUD_500K.
CAN_BAUD_50K (↗ see page 25)	This is macro CAN_BAUD_50K.
CAN_BAUD_83K3 (↗ see page 25)	This is macro CAN_BAUD_83K3.
CAN_BAUD_95K2 (↗ see page 25)	This is macro CAN_BAUD_95K2.
CAN_ERR_BUSOFF (↗ see page 26)	Busfehler: CAN_Controller ging 'Bus-Off
CAN_ERR_OK (↗ see page 26)	Fehlerzustände
CAN_ERR_OVERRUN (↗ see page 26)	CAN-Controller wurde zu spaet gelesen
CAN_ERR_QOVERRUN (↗ see page 26)	RcvQueue wurde zu spaet gelesen
CAN_ERR_QRCVEMPTY (↗ see page 27)	RcvQueue ist leergelesen
CAN_ERR_QXMTFULL (↗ see page 27)	Sendequueue ist voll
CAN_ERR_XMTFULL (↗ see page 27)	Sendepuffer im Controller ist voll
CAN_EXT (↗ see page 27)	for Extended CAN-ID
CAN_RTR (↗ see page 27)	for RTR
CAN_SETBUSOFF (↗ see page 28)	This is macro CAN_SETBUSOFF.
CAN_SETBUSON (↗ see page 28)	This is macro CAN_SETBUSON.

Types

Type	Description
CANl_qentrytyp (↗ see page 20)	Eine im Queue gepufferte CAN-Message

Variables

Variable	Description
CAN_BUFF_DATA (↗ see page 22)	This is variable CAN_BUFF_DATA.

CAN_BUFF_FORMAT (↗ see page 22)	This is variable CAN_BUFF_FORMAT.
CAN_BUFF_ID (↗ see page 22)	This is variable CAN_BUFF_ID.
CAN_BUFF_LEN (↗ see page 22)	This is variable CAN_BUFF_LEN.

1.5.3 compiler.h

This is file compiler.h.

Macros

Macro	Description
NULL (↗ see page 28)	This is macro NULL.

Types

Type	Description
uchar (↗ see page 21)	This is type uchar.
ulong (↗ see page 21)	This is type ulong.
word (↗ see page 21)	neu

1.5.4 eeprom.h

This is file eeprom.h.

Functions

Function	Description
EEprom_CloseRead (↗ see page 5)	Finishes read access.
EEprom_CloseWrite (↗ see page 6)	Finishes write access, writes checksum to protect data.
EEprom_isvalid (↗ see page 6)	Checks the consistency of the EEPROM content.
EEprom_OpenRead (↗ see page 6)	Open the EEPROM for read access. To use the Eeprom I2C_Init (↗ see page 12) must be called previously. The read-Pointer is set to the start of the EEPROM.
EEprom_OpenWrite (↗ see page 7)	Open the EEPROM for write access. To use the Eeprom I2C_Init (↗ see page 12) must be called previously. The Write-Pointer is set to the start of the EEPROM.
EEprom_Read (↗ see page 7)	Reads 'len' data from EEPROM. Data are copied into memory that starts at address'dat'.
EEprom_Write (↗ see page 7)	Writes 'len' data into EEPROM. Data are copied from memory that starts at address'dat'.

1.5.5 hardware.h

This is file hardware.h.

Functions

Function	Description
Get_DIN (↗ see page 8)	Reads one digital Input.
Get_DINByte (↗ see page 8)	Reads all 8 digital Inputs.
Get_DOUT (↗ see page 9)	Reads one digital Output.
Get_DOUTByte (↗ see page 9)	Reads all 8 digital Outputs.
Hardware_Init (↗ see page 11)	Initializes the Hardware (Watchdog, directions,...)
LED_1000Hz (↗ see page 15)	Call this function every 1ms for flashing.
LED_SetSpeed (↗ see page 15)	Call this function to set the flashing frequency of the LED.
Set_DOUT (↗ see page 16)	Sets one digital Output.
Set_DOUTByte (↗ see page 17)	Sets all 8 digital Outputs.
Set_DOUTByteMask (↗ see page 17)	Sets all digital Outputs where Bits in Parameter 'mask' is 1.

1.5.6 I2C.H

This is file I2C.H.

Functions

Function	Description
I2C_Delay (↗ see page 12)	This is function I2C_Delay.
I2C_Init (↗ see page 12)	Initializes the I2C-Interface. This is used by the On-Board EEPROM.
I2C_MasterRead (↗ see page 13)	Reads only Byte from the I2C-Bus.
I2C_MasterWrite (↗ see page 13)	Writes only Byte to the I2C-Bus.
I2C_Start (↗ see page 13)	Generates Start-Sequence for I2C-Bus.
I2C_Stop (↗ see page 14)	Generates Stop-Sequence for I2C-Bus.

1.5.7 I2C_2.H

This is file I2C_2.H.

Functions

Function	Description
I2C_2_Init (↗ see page 11)	This is function I2C_2_Init.
I2C_2_MasterRead (↗ see page 11)	This is function I2C_2_MasterRead.
I2C_2_MasterWrite (↗ see page 11)	This is function I2C_2_MasterWrite.
I2C_2_Start (↗ see page 12)	This is function I2C_2_Start.
I2C_2_Stop (↗ see page 12)	This is function I2C_2_Stop.

Macros

Macro	Description
__I2C2H__ (↗ see page 23)	Schutz gegen mehrfaches #include

1.5.8 pwm.h

This is file pwm.h.

Functions

Function	Description
GetFrequency (↗ see page 10)	Reads the current frequency signal. This is intended to measure low frequency signals (0..2500Hz)! Higher frequencies will decrease the overall performance of the module.
GetFrequencyInit (↗ see page 10)	Initializes Frequency Measurement.
GetTimer32 (↗ see page 10)	Returns the number of Milliseconds since Power Up.
PWM_Init (↗ see page 16)	Initializes PWM and Frequency output.
SetFrequency (↗ see page 17)	Enables Frequency-Output with given frequency and 50% duty cycle. The MicroMod does only support two Frequency outputs at the same time. If channel 0 (2) is used channel 1 (3) is disabled.
SetPWM (↗ see page 18)	Enables PWM-Output with given frequency and duty cycle. The frequency of Channel 0/1 and 2/3 must be equal.

1.5.9 timer.h

This is file timer.h.

Functions

Function	Description
Timer (↗ see page 18)	Reads back a Timer. The Timer is decremented every Millisecond. If the function returns 0 the Timer is stopped.
Timer_Init (↗ see page 18)	Initializes the Timer (↗ see page 18).
Timer_Set (↗ see page 19)	Sets a downcounting Timer (↗ see page 18) to a given value.
Timer_SystemReset (↗ see page 19)	Resets a global Timer (↗ see page 18) that starts counting Milliseconds at Power on.
Timer_SystemTime (↗ see page 19)	Reads back a global Timer (↗ see page 18) that starts counting Milliseconds at Power on.

1.5.10 uart.h

This is file uart.h.

Functions

Function	Description
Getch (↗ see page 9)	Reads a charakter from the UART. Waits until a character is received
GetKey (↗ see page 10)	Reads a charakter from the UART. Returns 0 if no character is available.
itoa (↗ see page 14)	Converts an integer into a String.
itoa_hex (↗ see page 14)	Converts an integer into a String. Number base is 16.
Putch (↗ see page 15)	Writes a charakter to the UART.
puts (↗ see page 16)	Writes a String to the UART.
UART_Init (↗ see page 19)	Initializes the UARD with a given Baudrate.
uitoa (↗ see page 20)	Converts an unsigned integer into a String.

Macros

Macro	Description
SER_BAUD_19200 (↗ see page 28)	This is macro SER_BAUD_19200.
SER_BAUD_38400 (↗ see page 29)	This is macro SER_BAUD_38400.
SER_BAUD_76800 (↗ see page 29)	This is macro SER_BAUD_76800.
SER_BAUD_9600 (↗ see page 29)	This is macro SER_BAUD_9600.

Index

—
__I2C2H__ 23

A

AD_1000Hz 1
AD_Init 1
AD_Read() 2
AD_SetTimeConstant() 2
ADI.H 29

C

can.h 30
CAN_BAUD_100K 23
CAN_BAUD_10K 23
CAN_BAUD_125K 23
CAN_BAUD_1M 23
CAN_BAUD_20K 24
CAN_BAUD_250K 24
CAN_BAUD_2M 24
CAN_BAUD_33K3 24
CAN_BAUD_47K6 25
CAN_BAUD_500K 25
CAN_BAUD_50K 25
CAN_BAUD_83K3 25
CAN_BAUD_95K2 25
CAN_BUFF_DATA 22
CAN_BUFF_FORMAT 22
CAN_BUFF_ID 22
CAN_BUFF_LEN 22
CAN_ERR_BUSOFF 26
CAN_ERR_OK 26
CAN_ERR_OVERRUN 26
CAN_ERR_QOVERRUN 26
CAN_ERR_QRCVEMPTY 27
CAN_ERR_QXMTFULL 27
CAN_ERR_XMTFULL 27
CAN_EXT 27
CAN_Init() 2
CAN_Offline 3

CAN_Online 3
CAN_Read 4
CAN_RegisterMsg() 4
CAN_RTR 27
CAN_RxQueueReset 4
CAN_SETBUSOFF 28
CAN_SETBUSON 28
CAN_Status 4
CAN_TxQueueReset 5
CAN_Write 5
CANI_qentrytyp 20
CANRxISRCallBack() 5
compiler.h 31

E

eeeprom.h 31
EEprom_CloseWrite 5, 6
EEprom_isvalid 6
EEprom_OpenRead 6
EEprom_OpenWrite 7
EEprom_Read() 7
EEprom_Write() 7
EnableExtendedID() 8

G

Get_DIN() 8
Get_DINByte 8
Get_DOUT() 9
Get_DOUTByte 9
Getch 9, 10
GetFrequency() 10
GetFrequencyInit 10
GetTimer32 10

H

hardware.h 31
Hardware_Init 11

I

I2C.H 32
I2C_2.H 32

I2C_2_Init 11
I2C_2_MasterRead 11
I2C_2_MasterWrite 11
I2C_2_Start 12
I2C_2_Stop 12
I2C_Delay 12
I2C_Init 12
I2C_MasterRead() 13
I2C_MasterWrite() 13
I2C_Start 13
I2C_Stop 14
itoa() 14
itoa_hex() 14

L

LED_1000Hz 15
LED_SetSpeed() 15

N

NULL 28

P

Putch() 15
puts() 16
pwm.h 32
PWM_Init 16

S

SER_BAUD_19200 28
SER_BAUD_38400 29
SER_BAUD_76800 29
SER_BAUD_9600 29
Set_DOUT() 16
Set_DOUTByte 17
Set_DOUTByteMask 17
SetFrequency() 17
SetPWM() 18

T

Timer() 18
timer.h 32

Timer_Init 18
Timer_Set() 19
Timer_SystemReset 19
Timer_SystemTime 19

U

uart.h 33
UART_Init() 19
uchar 21
uitoa() 20
ulong 21

V

Variables 21

W

word 21