INICORE *Specification*

CANmodule-II

Date June 7, 2002

Version 2.01

Abstract:
This document describes INICORE's CANmodule, a full CAN 2.0B controller designed for embedded systems. Programmable message filters, a receive FIFO and priority based transmit buffers help to optimize system performance and are designed to cover a wide application range.

Table of Contents

Definition of Terms

Following conventions are used in this document:

- Message Byte 1..8 -> D_63..D_0
- All default values are ‘0’ unless otherwise noted
- Following nomoclature is used register mapping
  r: readback operation
  R: Read operation
  W: Write operation
- Signals ending with ‘_n’ are active low

1 Overview

CANmodule is a full functional CAN controller module that contains advanced message filtering and receive- and transmit buffers. It is designed for advanced system-on-chip (SOC) devices, where a fast 32-bit CPU powers the system. Full message filtering together with a prioritized transmit queue supports a wide range of application. An AMBA Advanced Peripheral Bus (APB) interface enables smooth intergration into ARM based SOC's.

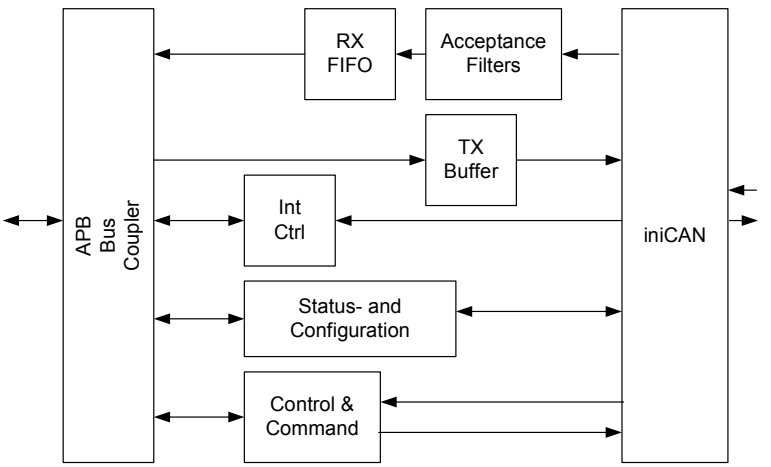
1.1 Features

The CANmodule is designed for a system-on-chip design.

- Full CAN 2.0B compliant
- Maximum baudrate of 1 Mbps
- 3 Programmable Acceptance Filters
  - Message filter covers: ID, IDE, RTR
- Transmit Path
  - 3 Tx message holding registers with programmable priority arbitration
  - Message abort command
- Receive FIFO
  - 4 message deep receive FIFO
  - FIFO status indicator
- APB Bus Coupler
  - AMBA 2.0 Advanced Peripheral Bus Interface
  - Full synchronous zero wait-states interface
  - 32-bit wide data path
  - Status and configuration interface
- Programmable Interrupt Controller
- Listen Only Mode
- Register Based Design
  - Technology independent
  - Acclerates time-to-market
  - Enables SCAN based test methodology
- 100% Synchronous Design

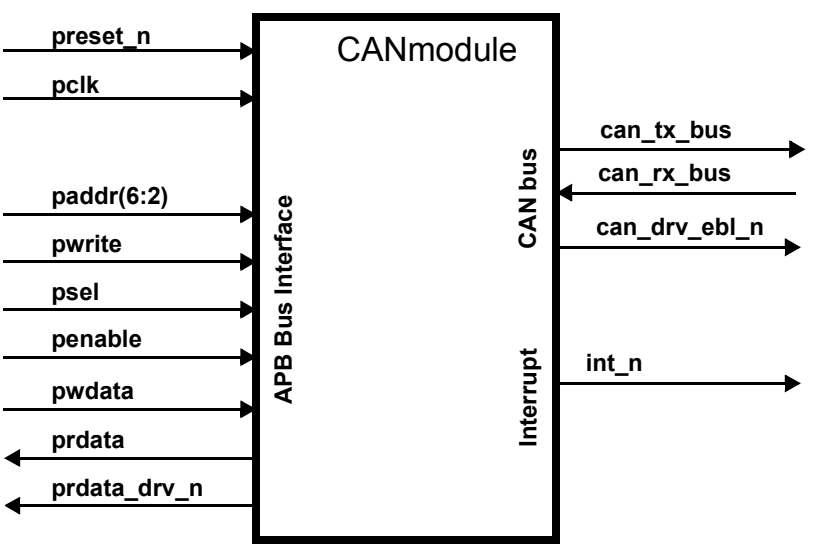
1.2 Block Diagram

The main building blocks are shown in the following figure:



1.3 Inputs - Outputs

This picture shows the main inputs and outputs:



2 IO Description

The following part lists the input and output ports of the CANmodule and gives a short overview of their functionality.

2.1 General Inputs

These pins are used to clock and initialize the whole CANmodule circuit.

pin name	type	description
pclk	in	System clock
preset_n	in	Asynchronous system reset, active low

2.2 APB Bus Interface

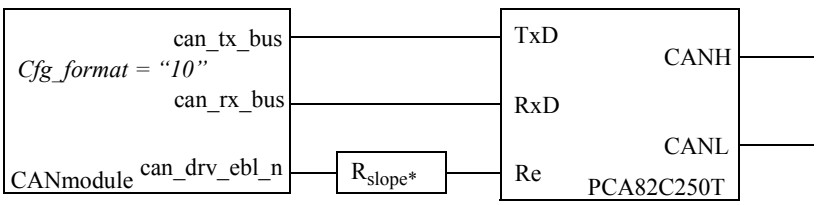
The on-chip bus interface is compliant to the AMBA 2.0 APB bus specification. The interface is full synchronous to the system clock. The interface supports true 32-bit access with zero wait-states.

If a bidirectional on-chip bus is used, prdata_drv_n can serve as drive enable.

pin name	type	description
psel	in	Module select signal
paddr(6:2)	in	Address bus
pwrite	in	Read/write signal ‘0’: read operation ‘1’: write operation
penable	in	Data enable
pwdata(31:0)	in	Write data bus
prdata(31:0)	out	Read data bus
prdata_drv_n	out	Data bus buffer enable signal ‘0’: CANmodule drives data bus ‘1’: Data bus is input
int_n	out	Interrupt request, active low

2.3 CAN Bus Interface

Three signals are provided to directly connect to a CANbus transceiver. The following picture shows how to connect an external Phillips CAN driver:



*) see specification of third party CAN transceiver for definition of R_{slope}.

pin name	type	description
can_rx_bus	in	Local receive signal (connect to can_rx_bus of external driver)
can_tx_bus	out	CANbus transmit signal, can be connected to external driver
can_drv_ebl_n	out	External driver control signal (unused without external driver)

3 Port Map

The table below shows the entire memory mapping of the CANmodule. All registers are 32-bit wide.

Following nomoclature is used to differentiate different bus access:

r: Data read back operation. Used to review to what value a configuration is set

R: Read operation.

W: Write operation

Port Map of internal registers:

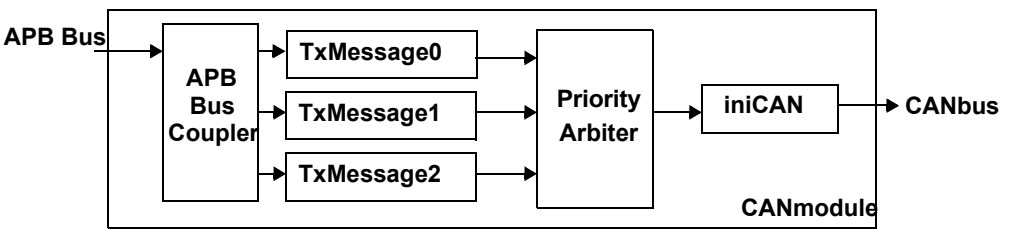
Address	Type	Register
0x00	r/W	TxMessage0 Buffer
0x04		
0x08		
0x0C		
0x10	r/W	TxMessage1 Buffer
0x14		
0x18		
0x1C		
0x20	r/W	TxMessage2 Buffer
0x24		
0x28		
0x2C		
0x30	R	RxMessage Buffer
0x34		
0x38		
0x3C		
0x40	r/W	Acceptance Register 0
0x44		
0x48	r/W	Acceptance Register 1
0x4C		
0x50	r/W	Acceptance Register 2
0x54		
0x58	r/W	Acceptance Config Register
0x60	R/W	Error Status Indicators
0x64	R/W	Interrupt Control
0x68		
0x6C		
0x70	R/W	CAN Operating Mode
0x74	r/W	CAN Configuration

3.1 Internal Register Description

This paragraph shows all internal registers and describes how the CANmodule can be used and programmed.

3.1.1 TX Message Registers

Three transmit message holding buffers are provided. An internal priority arbiter selects the message according to the chosen arbitration scheme. Upon transmission of a message or arbitration loss, the priority arbiter re-evaluates all pending messages.



Message Arbitration

The priority arbiter supports two different arbitration modes:

- Round Robin: Buffers are served in a defined order: 0-1-2-0-1... A particular buffer is only selected if its TxReq flag is set. This scheme guarantees that all buffers receive the same propability to send a message
- Fixed Priority: Buffer 0 has the highest priority, buffer 2 has the lowest priority. This way it is possible to designate buffer 0 as the buffer for error messages and it is guaranteed that they are sent first

See the description of the configuration register on how to select these modes.

Register Description:

Address	R/W	Name	Description
0x00	r/W	TxMessage0	TxMessage0 Buffer: Identifier [31:3]: ID[28:0] [2:0]: N/A
0x04	r/W		TxMessage0 Buffer: Data high [1] [31:0]: Data[63:32]
0x08	r/W		TxMessage0 Buffer: Data low [1] [31:0]: Data[31:0]

Address	R/W	Name	Description
0x0C	r/W	TxMessage0 *continued*	TxMessage0 Buffer: Control Flags [31]: WPN, Write Protect Not [2] '0': Bit [13:8] remain unchanged '1': Bit [13:8] are modified, default. This bit is always zero for readback [21]: RTR, Remote Bit [20]: IDE, Extended Identifier Bit [19:16]: DLC, Data Length Code. Invalid values are transmitted as they are, but only the number of data bytes is limited to eight [1]: TxAbort, Transmit Abort Request '0': idle, default [0]: TxReq, Transmit Request '0': idle, default
0x0C	R/W		TxMessage0 Buffer: Command Flags [31]: WPN, Write Protect Not [2] '0': default '1': prohibited [1]: TxAbort, Transmit Abort Request '0': idle '1': Requests removal of a pending message. The message is removed the next time an arbitration loss happened. The flag is cleared when the message was removed or when the message won arbitration. The TxReq flag is released at the same time [0]: TxReq, Transmit Request Write: '0': idle '1': Message Transmit Request[3] Read: '0': TxReq completed '1': TxReq pending
0x10-0x1C	TxMessage1 Buffer, see TxMessage0 Buffer for description		
0x20-0x2C	TxMessage2 Buffer, see TxMessage0 Buffer for description		

1. Byte 1 is Data[63:56], Byte 2 is Data[55:48], etc.
2. Using the WPN flag enables simple retransmission of the same message by only having to set the TRX flag without taking care of the special flags
3. The Tx message buffer must not be changed while TxReq is '1'!

Procedure for sending a message

- Write message into one of the transmit message holding registers (TxMessage0/1/2)
- Request transmission by setting the respective TxReq flag. This flag remains set as long as the message holding registers contains this message. The content of the message buffer must not be changed while the TxReq flag is set!
- The TxReq flag remains set as long as the message transmit request is pending
- The internal message priority arbiter selects the message according to the chosen arbitration scheme
- The successful transfer of a message is indicated by the respective tx_xfer interrupt and by releasing the TxReq flag. Depending on the tx_level configuration settings an additional interrupt source tx_msg is available to indicate that the message holding registers are empty or below a certain level

Procedure for removing a message from a transmit holding register

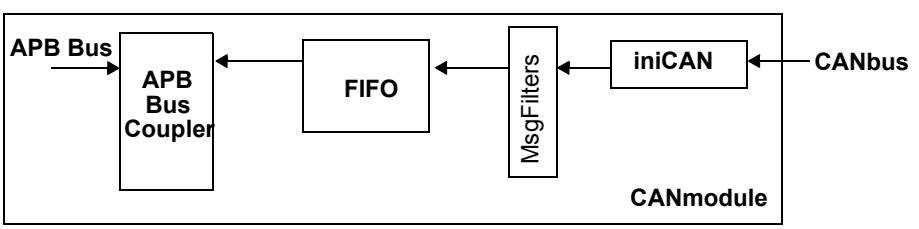
A message can be removed from one of the three transmit holding registers (TxMessage0/1/2) by setting the TxAbort flag.

Use following procedure to remove the contents of a particular TxMessage buffer:

- Set TxAbort to request the message removal
- This flag remains set as long as the message abort request is pending. It is cleared when either the message won arbitration (tx_xmit interrupt active) or the message was removed (tx_xmit interrupt inactive)

3.1.2 Rx Message Registers

A four message deep FIFO stores the incoming messages. Status flags indicate how many messages are stored. Additional flags are provided to determine from which acceptance filter the actual message is coming from.



Procedure for reading received messages

The following sequence outlines the recommended Rx message handling:

- Wait for rx_msg interrupt
- MessageReadLoop:
  - read message
  - acknowledge ‘message read’ by writing a ‘1’ to MsgAv register
  - read MsgAv; reading a ‘1’ means a new message is available
  - IF MsgAv=1 THEN jump to MessageReadLoop
- Acknowledge rx_msg interrupt by writing a ‘1’ to this register location

Register Description:

Address	R/W	Name	Description
0x30	R	RxMessage	RxMessage Buffer: Identifier [1] [31:3]: ID[28:0] [2:0]: zeros
0x34	R		RxMessage Buffer: Data high [2] [31:0]: Data[63:32]
0x38	R		RxMessage Buffer: Data low [2] [31:0]: Data[31:0]

Address	R/W	Name	Description
0x3C	R	RxMessage *continued*	RxMessage Buffer: Control Flags [21]: RTR, Remote Bit [20]: IDE, Extended Identifier Bit [19:16]: DLC, Data Length Code. Invalid values are shown as received [1]: TxAbort, Transmit Abort Request '0': idle, default [0]: MsgAv, Message Available '0': idle, default
0x3C	R/W		RxMessage Buffer: Config/Command Flags [3:1]: RxFifo[2:0]. This fill level indicator shows how many messages are available in the RxBuffer "000": buffer empty "001": 1 message waiting "010": 2 message waiting “011": 3 message waiting "100": 4 message waiting, FIFO full others: N/A [0]: MsgAv, Message Available Read: '0': No new message available '1': New message available Write: '0': idle '1': Acknowledges receipt of new message [3]

1. For standard sized messages (IDE=0), ID bits [17:0] are set to '1'
2. Byte 1 is Data[63:56], Byte 2 is Data[55:48], etc.
3. Acknowledging a message cleares the MsgAv flag. If a new message is directly available from the RxFIFO then this flag is not cleared and the new message can directly be read

3.1.3 Acceptance Filter and Acceptance Code Mask

Three programmable Acceptance Mask Register (AMR) and Acceptance Code Register (ACR) pairs are available to filter incoming messages. Each pair can be individually enabled through an Acceptance Filter Enable (AFE) register.

Following fields are covered:

- ID
- IDE
- RTC

The acceptance mask register (AMR) defines whether the incoming bit is checked against the acceptance code register (ACR).

AMR: ‘0’: The incoming bit is checked against the respective ACR. The message is discarded when the the incoming bit doesn’t match respective ACR flag

‘1’: The incoming bit is don’t care

Register Description:

Address	Name	R/W	Comment
0x40	AMR0	r/W	Acceptance Mask Register 0 [31:3]: Identifier [2]: IDE [1]: RTR [0]: N/A
0x44	ACR0	r/W	Acceptance Code Register 0 [31:3]: Identifier [2]: IDE [1]: RTR [0]: N/A
0x48	AMR1	r/W	Acceptance Mask Register 1 [31:3]: Identifier [2]: IDE [1]: RTR [0]: N/A
0x4C	ACR1	r/W	Acceptance Code Register 1 [31:3]: Identifier [2]: IDE [1]: RTR [0]: N/A
0x50	AMR2	r/W	Acceptance Mask Register 2 [31:3]: Identifier [2]: IDE [1]: RTR [0]: N/A
0x54	ACR2	r/W	Acceptance Code Register 2 [31:3]: Identifier [2]: IDE [1]: RTR [0]: N/A
0x58	AFE	r/W	Acceptance Filter Enable Register [2]: AFE2 ‘0’: Acceptance filter is disabled ‘1’: Acceptance filter is enabled [1]: AFE1 ‘0’: Acceptance filter is disabled ‘1’: Acceptance filter is enabled [0]: AFE0 ‘0’: Acceptance filter is disabled ‘1’: Acceptance filter is enabled

If all three message filters are disabled then no messages will be received! To receive all messages then one message filter must be enabled and programmed with all its fields set as “don’t care”.

3.1.4 Error Status Indicators

Status indicators are provided to report the CAN controller error state, receive error count and transmit error count. Special flags to report error counter values equal to or in excess of 96 errors are available to indicate heavily disturbed bus situations.

Address	Name	R/W	Comment
0x60	Error Status	R	CAN Error Status [19]: rxgte96 The receiver error counter is greater or equal 96(dec) [18]:txgte96 The transmitter error counter is greater or equal 96(dec) [17:16]: error_stat[1:0] The error state of the CAN node: “00”: error active (normal operation) “01”: error passive “1x”: bus off [15:8]: rx_err_cnt[7:0] The receiver error counter according to the Bosch CAN 2.0 specification. When in bus off, this counter is used to count the idle states. [7:0]: tx_err_cnt[7:0] The transmitter error counter according to the Bosch CAN 2.0 specification. When it is greater than 255(dec), it is fixed at 255.

3.1.5 Interrupt Controller

An interrupt enable register is provided to enable a particular interrupt source. This is done by setting this flag to ‘1’. Interrupt source are available for regular traffic, error, and diagnostic information.

Address	Name	R/W	Comment
0x64	Cfg_MsgTh	r/W	Cfg Message Threshold [3:2]: rx_level[1:0]:Sets the rx_msg interrupt threshold: 0: at least 1 message in receive Fifo 1: at least 2 messages in receive Fifo 2: at least 3 messages in receive Fifo 3: at least 4 messages in receive Fifo [1:0]: tx_level[1:0]:Sets the tx_msg interrupt threshold: 0: all tx buffers are empty 1: minimum 2 empty buffers 2: minimum 1 empty buffer 3: not applicable

Address	Name	R/W	Comment
0x68	IntEbl	r/W	Interrupt Enable Register An interrupt source is enabled by setting its respective flag to '1'. [15]: rx_msg [14]: tx_msg [13]: tx_xmit2 [12]: tx_xmit1 [11]: tx_xmit0 [10]: bus_off [9]: crc_err [8]: form_err [7]: ack_err [6]: stuff_errr [5]: bit_err [4]: tx_ovr [3]: ovr_load [2]: arb_loss [1]: N/A [0]: int_ebl, global interrupt enable flag '0': All interrupts are disabled '1': Enabled interrupt sources are available
0x6C	IntStatus	R/W	Interrupt Status Register A pending interrupt is indicated that its respective flag is set to '1'. To acknowledge an interrupt, set its flag to '1'. [15]: rx_msg [14]: tx_msg [1] [13]: tx_xmit2 [1] [12]: tx_xmit1 [1] [11]: tx_xmit0 [1] [10]: bus_off [9]: crc_err [8]: form_err [7]: ack_err [6]: stuff_errr [5]: bit_err [4]: tx_ovr [3]: ovr_load [2]: arb_loss [1]: N/A [0]: N/A

1. Acknowledging the tx_msg interrupt resets all tx_xmit interrupt sources as well. Acknowledging one of the tx_xmit interrupt sources resets the tx_msg interrupt too

Explanation of interrupt sources:

rx_msg: Depending on rx_level, at least one message is available

tx_msg: Depending on tx_level, at least one message buffer is empty

tx_xmit2/1/0 Indicates that the respective message was successfully sent

bus_off: The CAN has reached the bus off state

crc_err, form_err, ack_err, stuff_err, bit_err: Any of the mentioned error occurred while receiving or transmitting a message

rx_ovr: Receiver overrun: This Flag indicates that a new message arrived while the receive buffer is full. This Flag is set if either the incoming message overwrites an existing one or if it is discarded. (see also overwrite_new_message)

ovr_load: An overload condition has occurred

arb_loss: The arbitration was lost while sending a message

3.1.6 CANmodule Operating Mode

The CANmodule can be used in different operating modes. By disabling transmitting data, it is possible to us the CAN in listen only mode enabling features such as automatic bit rate detection. The two modules can be used in an on-chip loop-back mode.

Address	Name	R/W	Comment
0x70	Command	R/W	CAN Command Register [1]: Listen only mode: '0': Active '1': CAN listen only: The output is held at 'R' level. The CANmodule is only listening [0]: Run/Stop mode: '0': Sets the CAN controller into stop mode. Read '0' when stopped '1': Sets the CAN controller into run mode. Read '1' when running

3.1.7 CAN Configuration Register

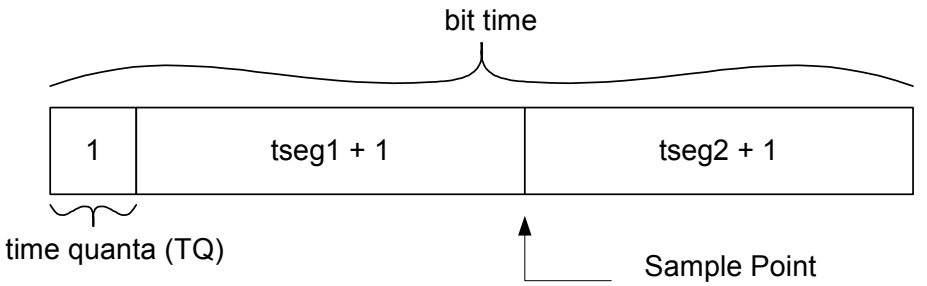
See chapter 4 for additional information on setting time segment 1, time segment 2, and the bitrate.

Address	Name	R/W	Comment
0x74	Config	r/W	CAN Configuration [27:15]: Cfg_bitrate[14:0]: Prescaler for generating the time quantum: "0000000000000":Maximum speed (1 TQ = 1 clock cycle) "0000000000001": 1 TQ = 2 clock cycles ... "1111111111111": 1 TQ = 9096 clock cycles [13]: Cfg_arbiter: '0': Round Robin: TxMessage0-1-2-0-etc. '1': Fixed Priority: TxMessage0 is *highest*, TxMessage 2 is *lowest* [12]: overwrite_last_msg: '0': Under the same conditions, a new message will be discarded and no rx_msg flag will be set '1': When the FIFO is full and a new message arrives it overwrites the message in RxMsg3 buffer [11:9]: Cfg_tseg1: Length - 1 of the first time segment (bit timing). It includes the propagation time segment. Cfg_tseg1=0 and cfg_tseg1=1 are not allowed [8:5]: Cfg_tseg2: Length -1 of the second time segment. Cfg_tseg2=0 is not allowed; cfg_tseg2=1 is only allowed in direct sampling mode [4]: auto_restart: '0': After bus off, the CAN must be started 'by hand' '1': After bus off, the CAN is restarting automatically after 128 groups of 11 recessive bits [3:2]: Cfg_sjw: Synchronization jump width - 1 $sjw \leq tseg1$ and $sjw \leq tseg2$ [1]: sampling_mode: '0': One sampling point is used in the receiver path '1': 3 sampling points with majority decision are used [0]: edge_mode: '0': Edge from 'R' to 'D' is used for synchronization '1': Both edges are used

4 Configuring the CAN Controller

4.1 Setting proper bitrate, tseg1 and tseg2

Following relations exist for bit time, time quanta, time segments 1/2 and the data sampling point.



$$\text{bittime} = (1 + (\text{tseg1} + 1) + (\text{tseg2} + 1)) \times \text{timequanta}$$

$$\text{timequanta} = \frac{\text{bitrate} + 1}{f_{clk}}$$

e.g., for 1 Mbps with f_{clk} = 8 Mhz, set bitrate = 0, tseg1 = 3 and tseg2 = 2

Please observe following conditions for setting tseg1 and tseg2:

tseg1=0 and tseg1=1 are not allowed

tseg2=0 is not allowed; tseg2=1 is only allowed in direct sampling mode.