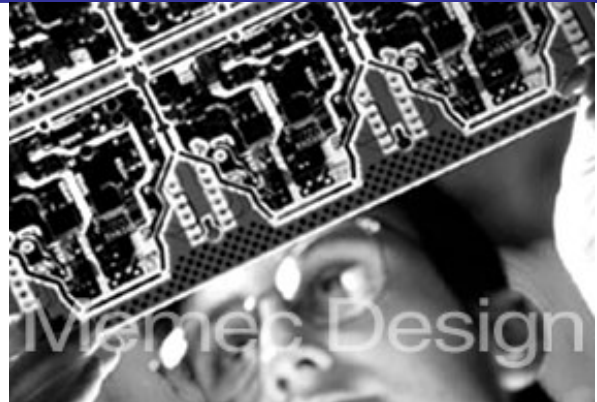




MemecCore™ Product Line
3721 Valley Centre Drive
San Diego, CA 92130 USA
Americas: +1 800-752-3040
Europe: +41 (0) 32 374 32 00
Asia: +(852) 2410 2720
E-mail: actel.info@memecdesign.com
URL: www.memecdesign.com/actel



Product Summary

Intended Use

- Automotive Industry
- Engine Control Unit
- Sensors

Key Features

- Supports SX-A devices, Axcelerator and ProASIC^{PLUS}
- Conforms to International Standard ISO/IEC 12989-1 Specification
- CAN 2.0B, 1Mbit/s
- Fully Synchronous Design
- Easy Parallel Interfaces
- Access to Internal Status
- Error reporting
- Customizable for Special Requirements

Targeted Devices

- SX-A Family
- Axcelerator Family
- ProASIC^{PLUS} Family

General Description

The MC-ACT-CAN core contains the complete data link layer, including the framer, transmit and receive control, error handling, error reporting and synchronization. Its structured core design and flexible interface enables access to each internal status and frame reference.

The core is designed to provide a bus bit rate up to 1Mbit/s with a minimum core clock frequency of 8 MHz. MC-ACT-CAN is a "core" logic module specifically designed for Actel FPGAs.

Core Deliverables

- Netlist Version
 - Netlist compatible with the Actel Designer place and route tool
- RTL Version
 - VHDL Source Code
 - Test Bench
- All
 - User Guide

Synthesis and Simulation Support

- Synthesis: Synplicity
- Simulation: ModelSim
- Other tools supported upon request

Verification

- Test Bench

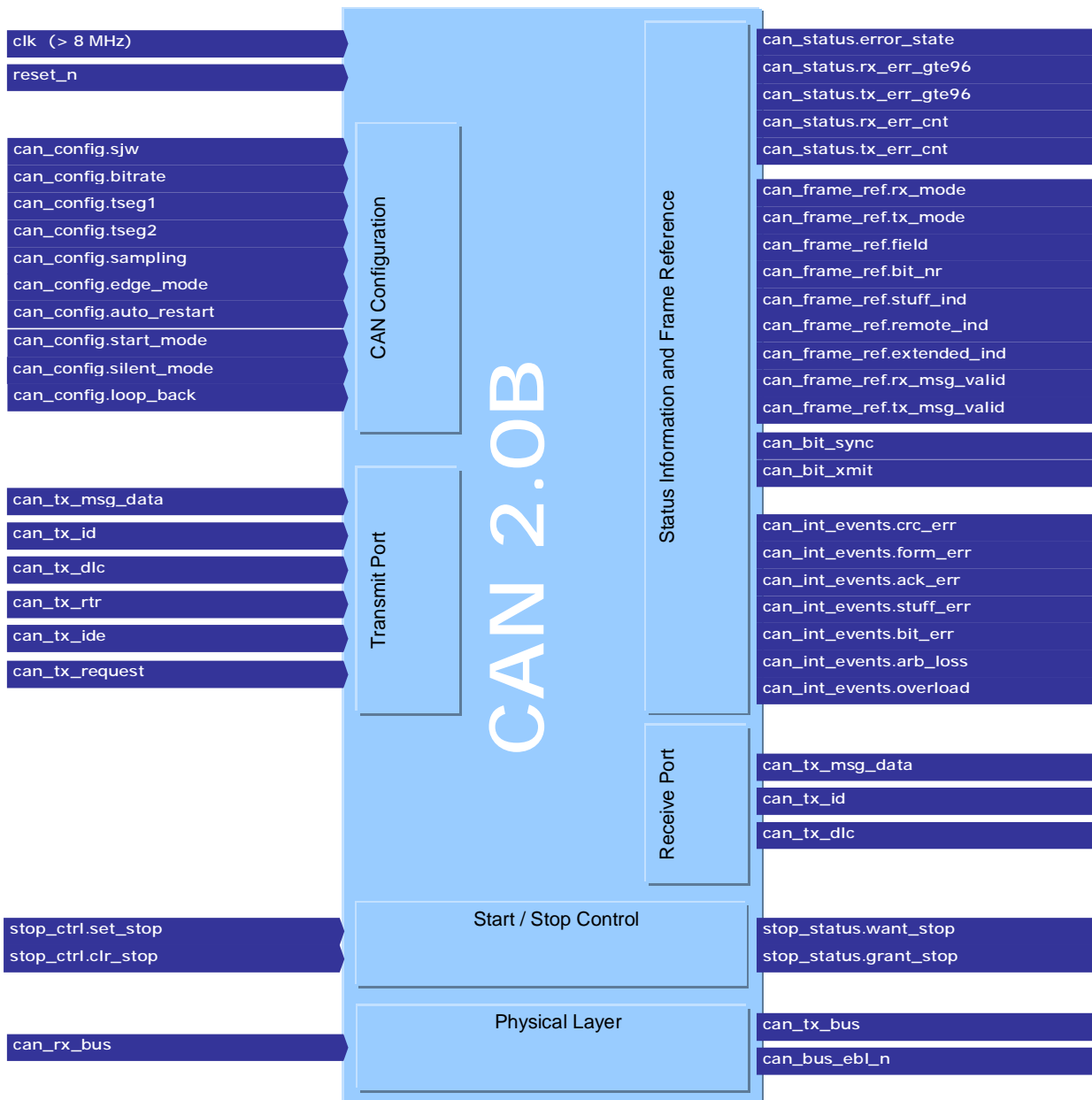


Figure 1: Logic Symbol

Functional Description

CAN Configuration

Bit rate, sub-bit segments (timing segment 1 & 2) and the synchronization jump width (sjw) for resynchronization can be configured to meet the required timing to the connected CAN bus.

The edge for resynchronization on the incoming message is defined in the edge-mode. Either the R-D edge or both (R-D and D-R) are used for resynchronization.

The MC-ACT-CAN provides two different sampling modes: Direct Sampling or three point sampling with majority decision.

Message Interface

The MC-ACT-CAN has a parallel interface for transmitting and receiving messages.

For sending a message, all signals for the transmitting message (id, dlc, rtr, ide and data) have to be applied and must be stable during the transmission. In case of arbitration loss or transmitting error, the MC-ACT-CAN supports message automatic retransmission. This feature can be disabled.

For receiving messages, all signals for the receiving message (id, dlc, rtr, ide and data) are valid when the message is successfully received.

Error Reporting

Error reporting is done by error counters and interrupt events.

The MC-ACT-CAN has two error counters. A transmit error counter for counting transmit errors and a receive error counter for counting receive errors. The counters represent the error value according to the CAN2.0B specification. An error count greater than 96 (dec) indicates a heavily disturbed bus. Therefore the core has two additional signals which indicate a counter value greater than 96. (One for the rx counter and one for the tx counter.)

The MC-ACT-CAN has three different error states:

- o Error Active (normal case)
- o Error Passive
- o Bus Off

In "Bus Off" state the framer doesn't have any influence on the bus. The behavior of the MC-ACT-CAN after entering "Bus Off" state can be configured: The CAN restarts only by a user command or it restarts automatically when the protocol allows it.

Interrupt events are generated by the core when one of the following situations occurs:

- o crc value doesn't match
- o format is not correct
- o message was not acknowledged
- o bit stuffing error
- o rx pin doesn't equal tx pin while transmitting
- o when arbitration is lost against other node
- o when overload occurs

Start/Stop Controlling

The MC-ACT-CAN can be set in stop or start mode by user command.

The behavior of the CAN after reset if it remains in stop mode or automatically synchronizes to the CAN bus can be set on a configuration input.

Frame Reference

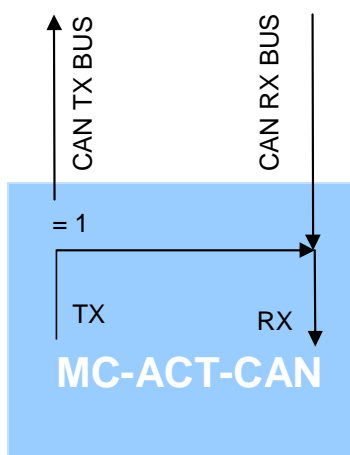
The MC-ACT-CAN allows the access to the internal frame status.

Following information is available:

- mode of the framer (transmit, receive or idle)
- message field and bit number of the actual sending or receiving message
- a stuff-index flag indicates if a stuff bit is inserted

Silent Mode

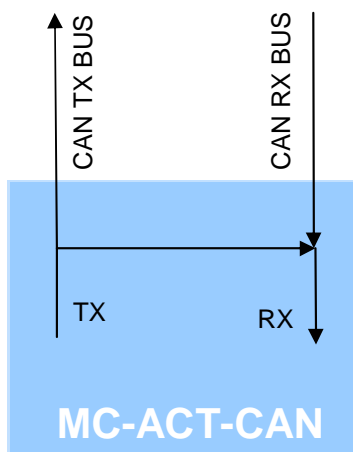
In silent mode the MC-ACT-CAN receives valid data and remote frames but sends only recessive bits on the CAN bus. In this mode the bus traffic can be analyzed without affecting it by sending dominant bits. The tx bus of the core is internally routed to the rx bus to receive acknowledges generated by this core.



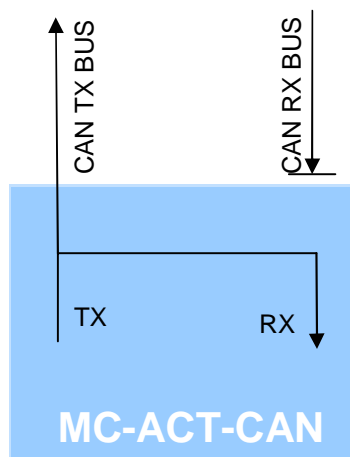
Loop Back Mode

In loop back mode the MC-ACT-CAN receives also messages which are sent by the core itself. Two different loop back mode can be configured.

Mode a: The core receives also messages sent by itself.



Mode b: The CAN receives only messages sent by itself and ignores messages and acknowledges from CAN rx bus.

**Silent mode combined with loop back mode**

Loop back and silent mode can be used together. In this configuration the MC-ACT-CAN can be tested without affecting a running CAN system.

Core Modifications

Customization is available through Memec Design.

Device Requirements

Family	Device	Utilization			Performance
		COMB	SEQ	Total	
SX-A	SX72A-STD	878 (22%)	254 (13%)	1132 (19%)	20 MHz
ProASIC ^{PLUS}	APA300-STD	n/a	n/a	1671 (14%)	18 MHz
Axcelerator	AX500-STD	826 (16%)	258 (10%)	1084 (14%)	28 MHz

Table 1: Device Utilization and Performance

Verification and Compliance

Complete functional and timing simulation has been performed on the CAN2.0B protocol handler using ModelSim 5.5d. The verification is based on the Bosch Reference CAN model. This core has also been used successfully in customer designs.

Signal Descriptions

The following signal descriptions define the IO signals.

Signal	Direction	Description
clk	Input	System Clock, rising edge used only, at least 8MHz for 1Mbit/s CAN data rate
reset_n	Input	Asynchronous system reset, active low, goes to all flip flops
can_config.bitrate[7:0]	Input	Defines the time quantum (TQ); one TQ is (bitrate+1)/clk e.g. for 1 Mbit/s and 8MHz clk: bitrate = 0
can_config.tseg1[3:0]	Input	(tseg1 + 1) = number of TQ in the first bit time segment: tseg1 = 0 and tseg1 = 1 are not allowed! e.g. for 1 Mbit/s and 8 MHz clk: tseg1 = 3
can_config.tseg2[2:0]	Input	(tseg2 + 1) = number of TQ in the second bit time segment: tseg2 = 0 are not allowed, tseg2 = 1 only allowed for direct sampling mode! e.g. for 1 Mbit/s and 8MHz clk: tseg2 = 2
can_config.sjw[1:0]	Input	(sjw + 1) = sync jump width (TQ) for resynchronization
can_config.sampling	Input	defines the sampling mode of the incoming message: '0': direct sampling (1 point) '1': 3 point sampling with majority decision
can_config.edge_mode	Input	defines, which edges on the incoming messages are used for resynchronization: '0': use only R-D edges '1': use R-D and D-R edges
can_config.auto_restart	Input	defines, if the MC-ACT-CAN should restart after bus off: '0': restart by user command (event on "stop_ctrl.clr_stop") '1': restart automatically when the protocol allows it (128 X 11 R bits)
can_config.start_mode	Input	defines, if the MC-ACT-CAN synchronizes to the bus or remains in stop mode after reset: '0': it remains in stop mode after reset '1': it synchronizes to the bus after reset
can_config.silent_mode	Input	In silent mode the MC-ACT-CAN only listens to the bus, it sends recessive bits on to CAN bus. '0': normal mode '1': silent mode
can_config.loob_back[1:0]	Input	In loop back mode the MC-ACT-CAN receives its own sent messages. "00": normal mode "01": receives incoming messages and its own sent messages "11": receives only messages sent by the core itself "10": n.a.
can_stop_ctrl.clr_stop	Input	Event sets the MC-ACT-CAN in the 'run' mode.

Signal	Direction	Description
can_stop_ctrl.set_stop	Input	Event sets the MC-ACT-CAN in the 'stop' mode, as soon as the protocol allows it (bus idle). So no protocol errors are generated when the CAN is stopped.
can_stop_status.want_stop	Output	'1' means, that the MC-ACT-CAN will stop as soon as possible (when in bus idle)
can_stop_status.grant_stop	Output	'1' means, that the MC-ACT-CAN is in user stop mode
can_status.rx_error_cnt[7:0]	Output	The receive error counter represents the error value according to the CAN2.0B specification. When in bus off, the counter will count up from 1(dec) to 128(dec) for counting the 128 x 11 recessive bits, before it will be allowed again to go error active.
can_status.tx_error_cnt[8:0]	Output	The transmit error counter represents the transmit error value according to the CAN2.0B specification
can_status.rx_err_gte96	Output	When the receive error counter is greater or equal 96(dec) this signal is activated (= '1') to signal a highly disturbed bus.
can_status.tx_err_gte96	Output	When the transmit error counter is greater or equal 96(dec) this signal is activated (= '1') to signal a highly disturbed bus.
can_status.error_state[1:0]	Output	Informs about the error state: "00": error active (normal case) "01": error passive "1x": bus off
can_int_events	Output	Error events are generated in following situations: .crc_err: crc value doesn't match .from_err: format (delimiters etc.) is not correct .ack_err: message was not acknowledged .stuff_err: bit stuffing .bit_err: rx pin doesn't equal tx pin while transmitting .arb_loss: when arbitration is lost against other node .overload: when overload occurs
can_frame_ref	Output	The whole internal framer status is available on the frame reference record. It contains the following signals: .field[4:0]: actual message field (coding see below) .bit_nr[6:0]: actual bit number in the message field .rx_mode: active '1' when in receive mode .tx_mode: active '1' when in transmit mode .stuff_ind: active '1' when a stuff bit is inserted .remote_ind: the RTR bit, valid at the end of a frame .extenden_ind: the IDE bit, valid at the end of a frame .rx_msg_valid: event for a successfully received message .tx_msg_valid: event for a successfully transmitted message
can_bit_sync	Output	The internal CAN framer (can_frame_ref) is triggered by this event. It can be used to synchronize external CAN logic to the framer.
can_bit_xmit	Output	The transmitter pin can_tx_bus is triggered by this event.
can_tx_msg_data[63:0]	Input	The data to be transmitted. The first data byte is represented in bits [63:56], the second in [55:48] etc. Shorter messages, the unused tx_msg_data may remain undefined. Bit 63 is the first one to be transmitted
can_tx_id[28:0]	Input	The identifier to be transmitted. When a standard message is sent, the 11bit identifiers must be placed in bit[28:18] while an extended frame will use 29 bits. Bit 28 is the first one to be transmitted.
can_tx_dlc[3:0]	Input	The data length code. Values between 0 and 8 are valid and determine how many data bytes will be transmitted. Values above 8 are illegal but will be transmitted as they are, but with 8 data bytes maximum.
can_tx_ide	Input	The extended identifier bit: '0': send a standard frame '1': send an extended frame
can_tx_rtr	Input	The remote indicate bit: '0': send a data frame

Signal	Direction	Description
		'1': send a remote frame
can_tx_request	Input	Active high signals to the core that a message is ready to be sent. All data and ty_request must be stable until tx_msg_valid is active. Use also the ty_msg_valid signal to clear tx_request!
can_rx_msg_data[63:0]	Output	The received data. The first data byte is represented in bits [63:56], the second in [55:48] etc. In shorter messages, the unused bits contain invalid data.
can_rx_id[28:0]	Output	The received identifier. When a standard message is received, the 11bit identifier is be placed in bits[28:18], bits[17:0] are '1'
can_rx_dlc[3:0]	Output	The data length code. Values between 0 and 8 are valid and determine how many data bytes have been received. Wrong values above 8 means that 8 data bytes are available!
can_tx_bus	Output	Transmitter pin of CAN bus D = low ('0') R = high ('1')
can_rx_bus	Input	Receiver pin of CAN bus D = low ('0') R = high ('1')
can_bus_ebl_n	Output	CAN bus driver enable not '0': active '1': passive This signal is passive when the CAN controller is stopped or when reset_n = '0'.

Table 2: Core I/O Signals

Recommended Design Experience

For the source version, users should be familiar with HDL entry and Actel design flows. Users should be familiar with Actel Libero v2.2 Integrated Design Environment (IDE) and preferably with Synplify and ModelSim.

Ordering Information

Part Number	Description
MC-ACT-CAN-NET	Core Netlist
MC-ACT-CAN-VHD	Core VHDL

Table 3: Core Part Numbers

The CORE is provided under license from Memec Design for use in Actel programmable logic devices. Please contact Memec Design for pricing and more information.

Information furnished by Memec Design is believed to be accurate and reliable. Memec Design reserves the right to change specifications detailed in this data sheet at any time without notice, in order to improve reliability, function or design, and assumes no responsibility for any errors within this document. Memec Design does not make any commitment to update this information.

Memec Design assumes no obligation to correct any errors contained herein or to advise any user of this text of any correction, if such be made, nor does the Company assume responsibility for the functioning of undescribed features or parameters. Memec Design will not assume any liability for the accuracy or correctness of any support or assistance provided to a user.

Memec Design does not represent that products described herein are free from patent infringement or from any other third-party right. No license is granted by implication or otherwise under any patent or patent rights of Memec Design.

MemecCore products are not intended for use in life support appliances, devices, or systems. Use of a MemecCore product in such application without the written consent of the appropriate Memec Design officer is prohibited.

All trademarks, registered trademarks, or service marks are property of their respective owners.

Datasheet Revision History

Version	Date	Description
Datasheet 1.0	January 9, 2003	Initial Release
Datasheet 1.1	January 23, 2003	Modification done in section core deliverables, Added logo to footer
Datasheet 1.2	February 25, 2003	Modification done in section device requirements, new URL and address inserted