CAN v5.0

LogiCORE IP Product Guide

Vivado Design Suite

PG096 August 2, 2016

Table of Contents

Introduction

The Xilinx® LogiCORE™ IP CAN core is ideally suited for automotive and industrial applications such as automotive gateways, body control units, automotive test equipment, instrument clusters, sensor controls, and industrial networks. Through user-configurable options, the Xilinx CAN core provides ultimate flexibility for multiple electronic control unit (ECU) applications. The core can be used in stand-alone mode or connected to Xilinx MicroBlaze™ or PowerPC™ processors.

Features

- Industrial (I, –40 to +100°C junction temperature) and automotive/defense Q-Grade (Q, –40 to +125°C junction temperature) device support.
- Supports both standard (11-bit identifier) and extended (29-bit identifier) frames
- Supports bit rates up to 1 Mb/s
- Transmit message FIFO with a user-configurable depth of up to 64 messages
- Transmit prioritization through one high-priority transmit buffer
- Automatic re-transmission on errors or arbitration loss
- Receive message FIFO with a user-configurable depth of up to 64 messages
- Acceptance filtering (through a user-configurable number) of up to four acceptance filters
- Sleep mode with automatic wake up
- Loopback mode for diagnostic applications
- Maskable error and status interrupts
- Readable error counters

LogiCORE™ IP Facts Table	
Core Specifics	
Supported Device Family[1]	UltraScale+™ Families, UltraScale™ Architecture, Zynq®-7000 All Programmable SoC, 7 Series
Supported User Interfaces	AXI4-Lite
Resources	See Table 2-3
Provided with Core	
Design Files	Encrypted RTL
Example Design	Verilog and VHDL
Test Bench	Verilog and VHDL
Constraints File	XDC
Simulation Model	Verilog and VHDL Structural
Supported S/W Driver[2]	Standalone
Tested Design Flows[3]	
Design Entry	Vivado® Design Suite Vivado
Simulation	For supported simulators, see the Xilinx Design Tools: Release Notes Guide.
Synthesis	Vivado Synthesis
Support	
Provided by Xilinx at the Xilinx Support web page	

Notes:

1. For a complete listing of supported devices, see the Vivado IP catalog.
2. Standalone driver details can be found in the SDK directory (<*install_directory*>/doc/usenglish/xilinx_drivers.htm). Linux OS and driver support information is available from the Xilinx Wiki page.
3. For the supported versions of the tools, see the Xilinx Design Tools: Release Notes Guide.

Overview

This product guide describes the architecture and features of the Xilinx® LogiCORE IP CAN core and the functionality of the various registers in the design. In addition, the CAN core user interface and its customization options are described. Defining the CAN protocol is outside the scope of this document, and knowledge of the specifications, *ISO 11898-1: Road Vehicles - Interchange of Digital Information - Controller Area Network (CAN) for High-Speed Communication* [Ref 1] and *Controller Area Network (CAN) version 2.0A and B Specification, Robert Bosch GmbH* [Ref 2] is assumed.

Figure 1-1 illustrates the high-level architecture of the CAN core. Descriptions of the submodules follow.

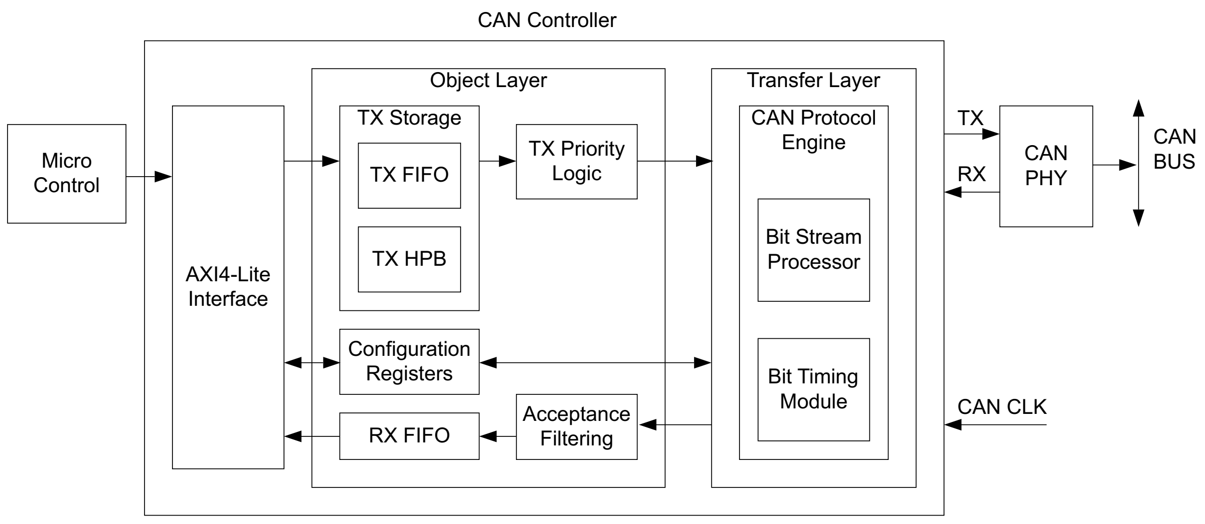


Figure 1-1: **CAN Core Block Diagram**

The CAN core requires an external 3.3V compatible physical-side interface (PHY) device.

Core Description

Configuration Registers

Table 2-5 defines the configuration registers. This module allows for read and write access to the registers through the external microcontroller interface.

Transmit and Receive Messages

Separate storage buffers exist for transmit (TX) and receive (RX) messages through a FIFO structure. The depth of each buffer is individually configurable up to a maximum of 64 messages.

TX High Priority Buffer

The Transmit High Priority Buffer (TX HPB) provides storage for one transmit message. Messages written on this buffer have maximum transmit priority. They are queued for transmission immediately after the current transmission is complete, preempting any message in the TX FIFO.

Acceptance Filters

Acceptance Filters sort incoming messages with the user-defined acceptance mask and ID registers to determine whether to store messages in the RX FIFO, or to acknowledge and discard them. The number of acceptance filters can be configured from 0 to 4. Messages passed through acceptance filters are stored in the RX FIFO.

CAN Protocol Engine

The CAN protocol engine consists primarily of the Bit Timing Logic (BTL) and the Bit Stream Processor (BSP) modules.

Figure 1-2 illustrates a block diagram of the CAN protocol engine.

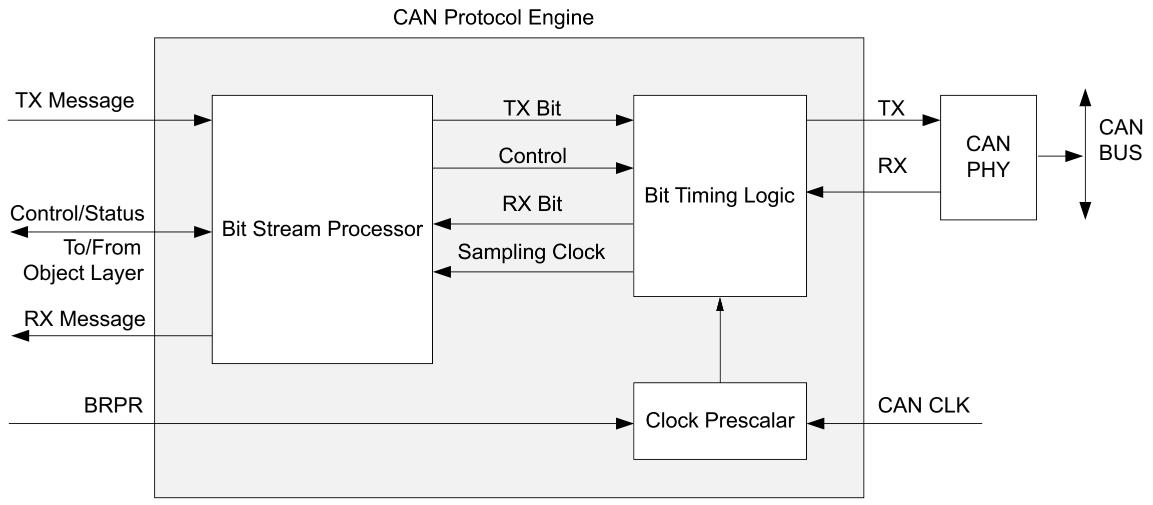


Figure 1-2: **CAN Protocol Engine**

Bit Timing Logic

The primary functions of the Bit Timing Logic (BTL) module include:

- Synchronizing the CAN core to CAN traffic on the bus
- Sampling the bus and extracting the data stream from the bus during reception
- Inserting the transmit bitstream onto the bus during transmission
- Generating a sampling clock for the BSP module state machine

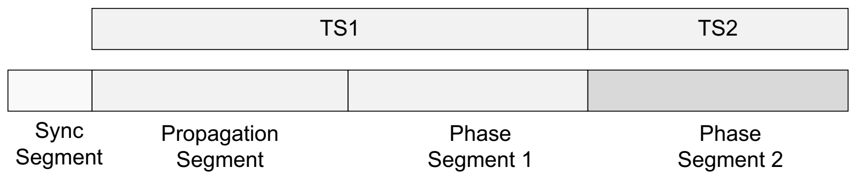


Figure 1-3: **CAN Bit Timing**

Figure 1-3 illustrates the CAN bit-time divided into four parts:

- Sync segment
- Propagation segment
- Phase segment 1
- Phase segment 2

The four bit-time parts are comprised of many smaller segments of equal length called *time quanta* (tq). The length of each time quantum is equal to the quantum clock time period (period = tq). The quantum clock is generated internally by dividing the incoming oscillator clock by the baud rate pre-scaler. The pre-scaler value is passed to the BTL module through the Baud Rate Prescaler (BRPR) register.

The propagation segment and phase segment 1 are joined together and called time segment1 (TS1), while phase segment 2 is called 'time segment2' (TS2). The number of time quanta in TS1 and TS2 vary with different networks and are specified in the Bit Timing Register (BTR), which is passed to the BTL module. The Sync segment is always one time quantum long.

The BTL state machine runs on the quantum clock. During the Start Of Frame (SOF) bit of every CAN frame, the state machine is instructed by the Bit Stream Processor module to perform a hard sync, forcing the recessive (r) to dominant edge (d) to lie in the sync segment. During the rest of the recessive-to-dominant edges in the CAN frame, the BTL is prompted to perform resynchronization.

During resynchronization, the BTL waits for a recessive-to-dominant edge. After that occurs, it calculates the time difference (number of tqs) between the edge and the nearest sync segment. To compensate for this time difference, and to force the sampling point to occur at the correct instant in the CAN bit time, the BTL modifies the length of phase segment 1 or phase segment 2.

The maximum amount by which the phase segments can be modified is dictated by the Synchronization Jump Width (SJW) parameter, which is also passed to the BTL through the BTR. The length of the bit time of subsequent CAN bits are unaffected by this process. This synchronization process corrects for propagation delays and oscillator mismatches between the transmitting and receiving nodes.

After the controller is synchronized to the bus, the state machine waits for a time period of TS1 and then samples the bus, generating a digital 0 or 1. This data is passed on to the BSP module for higher level tasks.

Bit Stream Processor

The Bit Stream Processor (BSP) module performs several Media Access Controller/Logical Link Control (MAC/LLC) functions during reception (RX) and transmission (TX) of CAN messages. The BSP receives a message for transmission from either the TX FIFO or the TX HPB and performs the following functions before passing the bitstream to BTL.

- Serializing the message
- Inserting stuff bits, Cyclic Redundancy Check (CRC) bits, and other protocol defined fields during transmission

During transmission the BSP simultaneously monitors RX data and performs bus arbitration tasks. It then transmits the complete frame when arbitration is won, and retrying when arbitration is lost.

During reception the BSP removes stuff bits, CRC bits, and other protocol fields from the received bitstream. The BSP state machine also analyzes bus traffic during transmission and reception for Form, CRC, ACK, Stuff and Bit violations. The state machine then performs error signaling and error confinement tasks. The CAN core does not voluntarily generate overload frames but does respond to overload flags detected on the bus.

This module determines the error state of the CAN core: Error Active, Error Passive or Bus-off. When TX or RX errors are observed on the bus, the BSP updates the transmit and receive error counters according to the rules defined in the CAN 2.0 A, CAN 2.0 B and ISO 11898-1 standards. Based on the values of these counters, the error state of the CAN core is updated by the BSP.

Applications

The CAN core can be connected to an AXI-based system along with other CAN nodes as shown.

Figure 1-4 shows the CAN core connected to a MicroBlaze™ processor. The MicroBlaze programs the two CAN nodes which can be used to transfer CAN messages between nodes through the CAN bus.

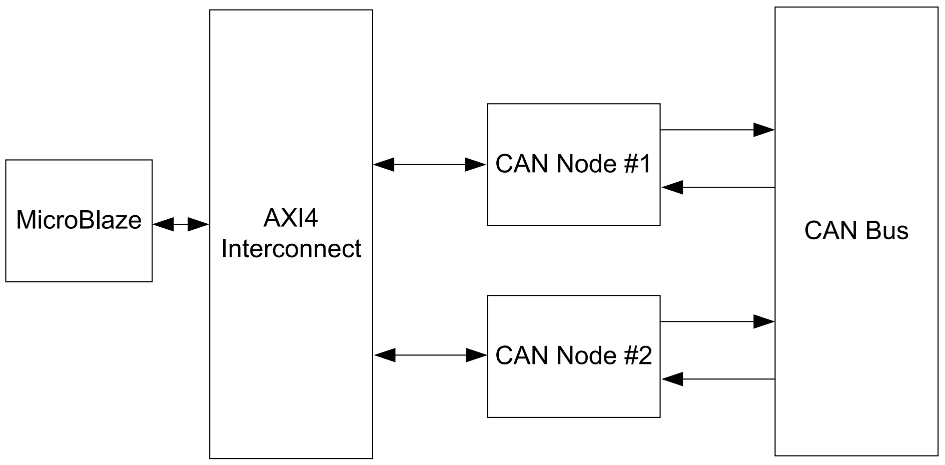


Figure 1-4: **CAN Core with MicroBlaze Processor**

Licensing and Ordering Information

License Checkers

If the IP requires a license key, the key must be verified. The Vivado® design tools have several license checkpoints for gating licensed IP through the flow. If the license check succeeds, the IP can continue generation. Otherwise, generation halts with error. License checkpoints are enforced by the following tools:

- Vivado design tools: Vivado Synthesis
- Vivado Implementation
- write_bitstream (Tcl command)

IMPORTANT: *IP license level is ignored at checkpoints. The test confirms a valid license exists. It does not check IP license level.*

License Type

The core is provided under the terms of the CAN LogiCORE™ IP License Agreement for Automotive or Non-Automotive applications. Click here for more information about obtaining a CAN license.

For more information, visit the CAN product web page.

Information about other Xilinx LogiCORE IP modules is available at the Xilinx Intellectual Property page. For information on pricing and availability of other Xilinx LogiCORE IP modules and tools, contact your local Xilinx sales representative.

Extra Design Consideration

The CAN core requires an input register on the RX line to avoid a potential error condition where multiple registers receive different values resulting in error frames. This error condition is rare; however, the work-around should be implemented in all cases. To work around this issue, insert a register on the RX line clocked by `can_clk` with an initial value of 1. This applies to all versions of the CAN core.

Chapter 2

Product Specification

CAN Core Modes

The CAN core supports these modes of operation:

- Configuration
- Normal
- Sleep
- Loopback

Table 2-1 defines the CAN core modes of operation and corresponding control and status bits. Inputs that affect the mode transitions are defined in Register Space.

Table 2-1: **CAN Core Modes of Operation**

s_axi_ aresetn	**SRST Bit (SRR)**	**CEN Bit (SRR)**	**LBACK Bit (MSR)**	**SLEEP Bit (MSR)**	**Status Register Bits (SR) (Read Only)**				**Operation Mode**
					CONFIG	LBACK	SLEEP	NORMAL	
0	X	X	X	X	1	0	0	0	Core is reset
1	1	X	X	X	1	0	0	0	Core is reset
1	0	0	X	X	1	0	0	0	Configuration mode
1	0	1	1	X	0	1	0	0	Loopback mode
1	0	1	0	1	0	0	1	0	Sleep mode
1	0	1	0	0	0	0	0	1	Normal mode

Configuration Mode

The CAN core enters Configuration mode, irrespective of the operation mode, when any of the following actions are performed:

- Writing a 0 to the CEN bit in the SRR register.
- Writing a 1 to the Software Reset (SRST) bit in the SRR register. The core enters Configuration mode immediately following the software reset.
- Driving a 0 on the `s_axi_aresetn` input. The core continues to be in reset as long as `s_axi_aresetn` is 0. The core enters Configuration mode after `s_axi_aresetn` is negated to 1.

Configuration Mode features:

- CAN core loses synchronization with the CAN bus and drives a constant recessive bit on the bus line.
- Error Counter Register (ECR) register is reset.
- Error Status Register (ESR) register is reset.
- BTR and BRPR registers can be modified.
- CONFIG bit in the Status Register is 1.
- CAN core does not receive any new messages.
- CAN core does not transmit any messages. Messages in the TX FIFO and the TX high priority buffer are held. These packets are sent when normal operation is resumed.
- Reads from the RX FIFO can be performed.
- Writes to the TX FIFO and TX HPB can be performed.
- Interrupt Status Register bits ARBLST, TXOK, RXOK, RXOFLW, ERROR, BSOFF, SLP and WKUP are cleared.
- Interrupt Status Register bits RXNEMP, RXUFLW can be set due to read operations to the RX FIFO.
- Interrupt Status Register bits TXBFLL and TXFLL, and the Status Register bits TXBFLL and TXFLL, can be set due to write operations to the TX HPB and TX FIFO, respectively.
- Interrupts are generated if the corresponding bits in the Interrupt Enable Register (IER) are 1.
- All Configuration Registers are accessible.

When in Configuration mode, the CAN core stays in this mode until the CEN bit in the SRR register is set to 1. After the CEN bit is set to 1 the CAN core waits for a sequence of 11 recessive bits before exiting Configuration mode.

The CAN core enters Normal, Loopback, or Sleep modes from Configuration mode, depending on the LBACK and SLEEP bits in the MSR Register.

Normal Mode

In Normal mode, the CAN core participates in bus communication by transmitting and receiving messages. From Normal mode, the CAN core can enter either Configuration or Sleep modes.

For Normal mode, the CAN core normal mode state transitions include the following:

- Enters Configuration mode when any configuration condition is satisfied
- Enters Sleep mode when the SLEEP bit in the Mode Select Register (MSR) is 1
- Enters Normal mode from Configuration mode only when LBACK and SLEEP bits in the MSR are 0 and CEN bit is 1
- Enters Normal mode from Sleep mode when a wake-up condition occurs

Sleep Mode

The CAN core enters Sleep mode from Configuration mode when the LBACK bit in MSR is 0, the SLEEP bit in MSR is 1, and the CEN bit in SRR is 1. The CAN core enters Sleep mode only when there are no pending transmission requests from either the TX FIFO or the TX High Priority Buffer.

The CAN core enters Sleep mode from Normal mode only when the SLEEP bit is 1, the CAN bus is idle, and there are no pending transmission requests from either the TX FIFO or TX High Priority Buffer.

When another node transmits a message, the CAN core receives the transmitted message and exits Sleep mode. When the controller is in Sleep mode, if there are new transmission requests from either the TX FIFO or the TX High Priority Buffer, these requests are serviced, and the CAN core exits Sleep mode. Interrupts are generated when the CAN core enters Sleep mode or wakes up from Sleep mode. From sleep mode, the CAN core can enter either the Configuration or Normal modes.

The CAN core can enter Configuration mode when any configuration condition is satisfied, and enters Normal mode under these (wake-up) conditions:

- Whenever the SLEEP bit is set to 0
- Whenever the SLEEP bit is 1, and bus activity is detected
- Whenever there is a new message in the TX FIFO or the TX High Priority Buffer

Loopback Mode

In Loopback mode, the CAN core transmits a recessive bitstream on to the CAN Bus. Any message transmitted is looped back to the RX line and is acknowledged. The CAN core receives any message that it transmits. It does not participate in normal bus communication and does not receive any messages transmitted by other CAN nodes.

This mode is used for diagnostic purposes — when in Loopback mode, the CAN core can only enter Configuration mode. The CAN core enters Configuration mode when any of the configuration conditions are satisfied. The CAN core enters Loopback mode from the Configuration mode if the LBACK bit in MSR is 1 and the CEN bit in SRR is 1.

Standards

The Xilinx® CAN core conforms to the ISO 11898 -1, CAN 2.0A, and CAN 2.0B standards.

Performance

Maximum Frequencies

The CAN core is characterized as per the benchmarking methodology described in the IP Characterization and F_{Max} Margin System Methodology section of the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 3]. Table 2-2 shows the results of the characterization runs.

Note: Maximum frequencies for Zynq®-7000 SoC All Programmable devices are expected to be similar to 7 series devices.

Table 2-2: **Maximum Frequencies**

Family	Speed Grade	F_{Max} (MHz)
Virtex-7	–1	180
Kintex-7		180
Artix-7		120
Virtex-7	–2	200
Kintex-7		200
Artix-7		140
Virtex-7	–3	220
Kintex-7		220
Artix-7		160

Resource Utilization

Resource requirements for the CAN core have been estimated for 7 series and Zynq-7000 devices (Table 2-3). These values were generated using the Vivado® Design Suite.

Note: Resource numbers for UltraScale™ architecture and Zynq-7000 devices are expected to be similar to 7 series device numbers.

Table 2-3: **Device Utilization – 7 Series and Zynq-7000 Devices**

TX Depth	RX Depth	No. of Acceptance Filters	Slice LUTs	Slice Registers
2	2	0	715	523
2	2	1	788	617
2	2	2	794	620
2	2	3	802	623
2	2	4	808	626
4	4	0	728	539
4	4	1	800	633
4	4	2	806	636
4	4	3	811	639
4	4	4	820	642
8	8	0	741	555
8	8	1	811	649
8	8	2	817	652
8	8	3	823	655
8	8	4	827	658
16	16	0	752	571
16	16	1	822	665
16	16	2	825	668
16	16	3	833	671
16	16	4	838	674
32	32	0	766	587
32	32	1	832	681
32	32	2	841	684
32	32	3	846	687
32	32	4	853	690
64	64	0	781	603
64	64	1	845	697
64	64	2	853	700

Table 2-3: **Device Utilization – 7 Series and Zynq-7000 Devices *(Cont'd)***

TX Depth	RX Depth	No. of Acceptance Filters	Slice LUTs	Slice Registers
64	64	3	857	703
64	64	4	866	706

Port Descriptions

The external interface of the CAN core is the AXI4-Lite Interface. Table 2-4 defines the CAN core interface signaling.

Table 2-4: **CAN Core I/O Signals**

Signal Name	Interface	Type	Default	Description
AXI4-Lite Interface Signals				
s_axi_*	S_AXI_LITE	-	-	See the *Vivado AXI Reference Guide* (UG1037) [Ref 4] for the description of the AXI4 Signals.
Clock, Interrupt and PHY Signals				
ip2bus_intrevent	Interrupt	O	0x0	Active-High interrupt line.(1)
can_clk	Clock	I	-	24 MHz oscillator clock input.
can_phy_tx	PHY	O	1	CAN bus transmit signal to PHY.
can_phy_rx	PHY	I	-	CAN bus receive signal from PHY.

Notes:

1. The interrupt line is level sensitive. Interrupts are indicated by the transition of the interrupt line logic from 0 to 1.

Register Space

The CAN core register space is shown in Table 2-5. The CAN registers are memory-mapped into read-only memory space.

IMPORTANT: *This memory space must be aligned to an AXI word (32-bit) boundary.*

Endianess

All registers are in little endian format as shown in Figure 2-1.

31 Byte 3 24	23 Byte 2 16	15 Byte 1 8	7 Byte 0 0
Address Offset 0x03	Address Offset 0x02	Address Offset 0x01	Address Offset 0x00

Figure 2-1: **32-bit Little Endian Example**

Table 2-5: **CAN Register Address Map**

Address Space Offset (Hex)	Name	Description
Control Registers		
0x000	SRR	Software Reset Register
0x004	MSR	Mode Select Register
Transfer Layer Configuration Registers		
0x008	BRPR	Baud Rate Prescaler Register
0x00C	BTR	Bit Timing Register
Error Indication Registers		
0x010	ECR	Error Counter Register
0x014	ESR	Error Status Register
Status Register		
0x018	SR	Status Register
Interrupt Registers		
0x01C	ISR	Interrupt Status Register
0x020	IER	Interrupt Enable Register
0x024	ICR	Interrupt Clear Register
0x028 to 0x02C	-	Reserved

Table 2-5: **CAN Register Address Map *(Cont'd)***

Address Space Offset (Hex)	Name	Description
Messages		
0x030	TXFIFO – ID	TXFIFO Message Storage Register
0x034	TXFIFO- DLC	TXFIFO Message Storage Register
0x038	TXFIFO-DW1	TXFIFO Message Storage Register
0x03C	TXFIFO-DW2	TXFIFO Message Storage Register
0x040	TXHPB-ID	TXHPB Message Storage Registers
0x044	TXHPB-DLC	TXHPB Message Storage Registers
0x048	TXHPB-DW1	TXHPB Message Storage Registers
0x04C	TXHPB-DW2	TXHPB Message Storage Registers
0x050	RXFIFO-ID	RXFIFO Message Storage Registers
0x054	RXFIFO-DLC	RXFIFO Message Storage Registers
0x058	RXFIFO-DW1	RXFIFO Message Storage Registers
0x05C	RXFIFO-DW2	RXFIFO Message Storage Registers
Acceptance Filtering		
0x060	AFR	Acceptance Filter Register
0x064	AFMR1	Acceptance Filter Mask Register – 1
0x068	AFIR1	Acceptance Filter ID Register – 1
0x06C	AFMR2	Acceptance Filter Mask Register – 2
0x070	AFIR2	Acceptance Filter ID Register – 2
0x074	AFMR3	Acceptance Filter Mask Register – 3
0x078	AFIR3	Acceptance Filter ID Register – 3
0x07C	AFMR4	Acceptance Filter Mask Register – 4
0x080	AFIR4	Acceptance Filter ID Register – 4
0x084 to 0x0FC	-	Reserved

Control Registers

Software Reset Register

Writing to the Software Reset Register (SRR) places the CAN core in Configuration mode. When in Configuration mode, the CAN core drives recessive on the bus line and does not transmit or receive messages. During power-up, CEN and SRST bits are 0 and the CONFIG bit in the Status Register (SR) is 1. The Transfer Layer Configuration Registers can be changed only when the CEN bit in the SRR Register is 0.

Use these steps to configure the CAN core at power up:

1. Configure the Transfer Layer Configuration Registers (BRPR and BTR) with the values calculated for the particular bit rate.

   See Baud Rate Prescaler Register and Bit Timing Register.

2. Do one of the following:

   - For Loopback mode, write 1 to the LBACK bit in the MSR.
   - For Sleep mode, write 1 to the SLEEP bit in the MSR.

   See Table 2-4 for information about operational modes.

3. Set the CEN bit in the SRR to 1.

   After the occurrence of 11 consecutive recessive bits, the CAN core clears the CONFIG bit in the Status Register to 0 and sets the appropriate Status bit in the Status Register.

Table 2-6 defines the bit positions in the Software Reset Register (SRR) and Table 2-7 defines the Software Reset Register bits.

Table 2-6: **Software Reset Register BIT Positions**

31:2	1	0
Reserved	CEN	SRST

Table 2-7: **Software Reset Register Bits**

Bits	Name	Access	Default Value	Description
31:2	Reserved	Read/Write	0	Reserved. Reserved for future expansion.
1	CEN	Read/Write	0	Can Enable. The Enable bit for the CAN core. 1 = The CAN core is in Loopback, Sleep or Normal mode depending on the LBACK and SLEEP bits in the MSR. 0 = The CAN core is in the Configuration mode.
0	SRST	Read/Write	0	Reset. The software reset bit for the CAN core. 1 = CAN core is reset. If a 1 is written to this bit, all the CAN core configuration registers (including the SRR) are reset. Reads to this bit always return a 0.

Mode Select Register

Writing to the Mode Select Register (MSR) enables the CAN core to enter Sleep, Loopback, or Normal modes. In Normal mode, the CAN core participates in normal bus communication. If the SLEEP bit is set to 1, the CAN core enters Sleep mode. If the LBACK bit is set to 1, the CAN core enters Loopback mode.

The LBACK and SLEEP bits should never be set to 1 at the same time. At any given point the CAN core can be either in Loopback mode or Sleep mode, but not both simultaneously. If both are set, the LBACK Mode takes priority.

Table 2-8 shows the bit positions in the MSR and Table 2-9 describes the MSR bits.

Table 2-8: **Mode Select Register Bits**

31:2	1	0
Reserved	LBACK	SLEEP

Table 2-9: **Mode Select Register Bits**

Bits	Name	Access	Default Value	Description
31:2	Reserved	Read/Write	0	Reserved. Reserved for future expansion.
1	LBACK	Read/Write	0	Loopback Mode Select. The Loopback Mode Select bit. 1 = CAN core is in Loopback mode. 0 = CAN core is in Normal, Configuration, or Sleep mode. This bit can be written to only when CEN bit in SRR is 0.
0	SLEEP	Read/Write	0	Sleep Mode Select. The Sleep Mode select bit. 1 = CAN core is in Sleep mode. 0 = CAN core is in Normal, Configuration or Loopback mode. This bit is cleared when the CAN core wakes up from the Sleep mode.

Transfer Layer Configuration Registers

There are two Transfer Layer Configuration Registers: Baud Rate Prescaler Register (BRPR) and Bit Timing Register (BTR). These registers can be written to only when CEN bit in the SRR is 0.

Baud Rate Prescaler Register

The CAN clock for the CAN core is divided by (prescaler+1) to generate the quantum clock needed for sampling and synchronization. Table 2-10 shows the bit positions in the BRPR, and Table 2-11 defines the BRPR bits.

Table 2-10: **Baud Rate Prescaler Register Positions**

31:8	7:0
Reserved	BPR [7:0]

Table 2-11: **Baud Rate Prescaler Register Bits**

Bits	Name	Access	Default Value	Description
31:8	Reserved	Read/Write	0	Reserved Reserved for future expansion.
7:0	BRP[7:0]	Read/Write	0	Baud Rate Prescaler These bits indicate the prescaler value. The actual value ranges from 1–256.

The BRPR can be programmed to any value in the range 0–255. The actual value is one more than the value written into the register.

The CAN quantum clock can be calculated using this equation:

$$tq = tosc \times (BRP + 1)$$

where tq and tosc are the time periods of the quantum and oscillator/system clocks respectively.

TIP: A given CAN bit rate can be achieved with several bit-time configurations, but values should be selected after careful consideration of oscillator tolerances and CAN propagation delays. For details about CAN bit-time register settings, see the *CAN 2.0A, CAN 2.0B, ISO 11898-1* specification [Ref 1].

Bit Timing Register

The Bit Timing Register (BTR) specifies the bits needed to configure bit time. Specifically, the Propagation Segment, Phase segment 1, Phase segment 2, and Synchronization Jump Width (as defined in *CAN 2.0A, CAN 2.0B and ISO 11891-1*) are written to the BTR. The actual value of each of these fields is one more than the value written to this register. Table 2-12 shows the bit positions in the BTR and Table 2-13 defines the BTR bits.

Table 2-12: **Bit Timing Register BIT Positions**

31:9	8:7	6:4	3:0
Reserved	SJW[1:0]	TS2[2:0]	TS1[3:0]

Table 2-13: **Bit Timing Register Bits**

Bits	Name	Access	Default Value	Description
31:9	Reserved	Read/Write	0	Reserved. Reserved for future expansion.
8:7	SJW[1:0]	Read/Write	0	Synchronization Jump Width. Indicates the Synchronization Jump Width as specified in the CAN 2.0A and CAN 2.0B standard. The actual value is one more than the value written to the register.
6:4	TS2[2:0]	Read/Write	0	Time Segment 2. Indicates Phase Segment 2 as specified in the CAN 2.0A and CAN 2.0B standard. The actual value is one more than the value written to the register.
3:0	TS1[3:0]	Read/Write	0	Time Segment 1. Indicates the Sum of Propagation Segment and Phase Segment 1 as specified in the CAN 2.0A and CAN 2.0B standard. The actual value is one more than the value written to the register.

These equations can be used to calculate the number of time quanta in bit-time segments:

$$tTSEG1 = tq \times (8 \times TSEG1[3] + 4 \times TSEG1[2] + 2 \times TSEG1[1] + TSEG1[0] + 1)$$

$$tTSEG2 = tq \times (4 \times TSEG2[2] + 2 \times TSEG2[1] + TSEG2[0] + 1)$$

$$tSJW = tq \times (2 \times SJW[1] + SJW[0] + 1)$$

–where tTSEG1, tTSEG2 and tSJW are the lengths of TS1, TS2 and SJW.

TIP: A given bit-rate can be achieved with several bit-time configurations, but values should be selected after careful consideration of oscillator tolerances and CAN propagation delays. For details on CAN bit-time register settings, see the *CAN 2.0A, CAN 2.0B, and ISO 11898-1* specification [Ref 1].

Error Indication Registers

The Error Counter Register (ECR) and the Error Status Register (ESR) comprise the Error Indication Registers.

Error Counter Register

The Error Counter Register (ECR) is a read-only register. Writes to the ECR have no effect. The value of the error counters in the register reflect the values of the transmit and receive error counters in the CAN Protocol Engine Module (see Figure 1-1).

These conditions reset the Transmit and Receive Error counters:

- When 1 is written to the SRST bit in the SRR
- When 0 is written to the CEN bit in the SRR
- When the CAN core enters Bus Off state
- During Bus Off recovery when the CAN core enters Error Active state after 128 occurrences of 11 consecutive recessive bits

When in Bus Off recovery, the Receive Error counter is advanced by one when a sequence of 11 consecutive recessive bits is seen.

Table 2-14 shows the bit positions in the ECR and Table 2-15 defines the ECR bits.

Table 2-14: **Error Count Register BIT Positions**

31:16	15:8	7:0
Reserved	REC[7:0]	TEC[7:0]

Table 2-15: **Error Count Register Bits**

Bits	Name	Access	Default Value	Description
31:16	Reserved	Read Only	0	Reserved. Reserved for future expansion.
15:8	REC[7:0]	Read Only	0	Receive Error Counter. Indicates the Value of the Receive Error Counter.
7:0	TEC[7:0]	Read Only	0	Transmit Error Counter. Indicates the Value of the Transmit Error Counter.

Error Status Register

The Error Status Register (ESR) indicates the type of error that has occurred on the bus. If more than one error occurs, all relevant error flag bits are set in this register. The ESR is a write-to-clear register. Writes to this register do not set any bits, but clear the bits that are set.

Table 2-16 shows the bit positions in the ESR and Table 2-17 describes the ESR bits. All the bits in the ESR are cleared when 0 is written to the CEN bit in the SRR.

Table 2-16: **Error Status Register BIT Positions**

31:5	4	3	2	1	0
Reserved	ACKER	BERR	STER	FMER	CRCER

Table 2-17: **Error Status Register Bits**

Bits	Name	Access	Default Value	Description
31:5	Reserved	Read/Write	0	Reserved. Reserved for future expansion.
4	ACKER	Write to Clear	0	ACK Error. Indicates an acknowledgement error. 1 = Indicates an acknowledgement error has occurred. 0 = Indicates an acknowledgement error has not occurred on the bus after the last write to this register. If this bit is set, writing a 1 clears it.
3	BERR	Write to Clear	0	Bit Error. Indicates the received bit is not the same as the transmitted bit during bus communication. 1 = Indicates a bit error has occurred. 0 = Indicates a bit error has not occurred on the bus after the last write to this register. If this bit is set, writing a 1 clears it.
2	STER	Write to Clear	0	Stuff Error. Indicates an error if there is a stuffing violation. 1 = Indicates a stuff error has occurred. 0 = Indicates a stuff error has not occurred on the bus after the last write to this register. If this bit is set, writing a 1 clears it.
1	FMER	Write to Clear	0	Form Error. Indicates an error in one of the fixed form fields in the message frame. 1 = Indicates a form error has occurred. 0 = Indicates a form error has not occurred on the bus after the last write to this register. If this bit is set, writing a 1 clears it.
0	CRCER	Write to Clear	0	CRC Error.(1) Indicates a CRC error has occurred. 1 = Indicates a CRC error has occurred. 0 = Indicates a CRC error has not occurred on the bus after the last write to this register. If this bit is set, writing a 1 clears it.

Notes:

1. In case of a CRC Error and a CRC delimiter corruption, only the FMER bit is set.

CAN Status Registers

The Status Register (SR) comprises the CAN Status Registers.

Status Register

The Status Register (SR) provides a status of all conditions of the core. Specifically, FIFO status, Error State, Bus State and Configuration mode are reported. Table 2-18 shows the SR bit positions in the SR and Table 2-19 provides SR bit descriptions.

Table 2-18: **Status Register BIT Positions**

31:12	11	10	9	8:7	6
Reserved	ACFBSY	TXFLL	TXBFLL	ESTAT[1..0]	ERRWRN
5	**4**	**3**	**2**	**1**	**0**
BBSY	BIDLE	NORMAL	SLEEP	LBACK	CONFIG

Table 2-19: **Status Register Bits**

Bits	Name	Access	Default Value	Description
31:12	Reserved	Read/Write	0	Reserved. Reserved for future expansion.
11	ACFBSY	Read Only	0	Acceptance Filter Busy. This bit indicates that the Acceptance Filter Mask Registers and the Acceptance Filter ID Registers cannot be written to. 1 = Acceptance Filter Mask Registers and Acceptance Filter ID Registers cannot be written to. 0 = Acceptance Filter Mask Registers and the Acceptance Filter ID Registers can be written to. This bit exists only when the number of acceptance filters is not 0 This bit is set when a 0 is written to any of the valid Use Acceptance Filter (UAF) bits in the Acceptance Filter Register.
10	TXFLL	Read Only	0	Transmit FIFO Full. Indicates that the TX FIFO is full. 1 = Indicates the TX FIFO is full. 0 = Indicates the TX FIFO is not full.
9	TXBFLL	Read Only	0	High Priority Transmit Buffer Full. Indicates the High Priority Transmit Buffer is full. 1 = Indicates the High Priority Transmit Buffer is full. 0 = Indicates the High Priority Transmit Buffer is not full.

Table 2-19: **Status Register Bits *(Cont'd)***

Bits	Name	Access	Default Value	Description
8:7	ESTAT[1..0]	Read Only	0	Error Status. Indicates the error status of the CAN core. 00 = Indicates Configuration Mode (CONFIG = 1). Error State is undefined. 01 = Indicates Error Active State. 11 = Indicates Error Passive State. 10 = Indicates Bus Off State.
6	ERRWRN	Read Only	0	Error Warning. Indicates that either the Transmit Error counter or the Receive Error counter has exceeded a value of 96. 1 = One or more error counters have a value ≥ 96. 0 = Neither of the error counters has a value ≥ 96.
5	BBSY	Read Only	0	Bus Busy. Indicates the CAN bus status. 1 = Indicates that the CAN core is either receiving a message or transmitting a message. 0 = Indicates that the CAN core is either in Configuration mode or the bus is idle.
4	BIDLE	Read Only	0	Bus Idle. Indicates the CAN bus status. 1 = Indicates no bus communication is taking place. 0 = Indicates the CAN core is either in Configuration mode or the bus is busy.
3	NORMAL	Read Only	0	Normal Mode. Indicates the CAN core is in Normal Mode. 1 = Indicates the CAN core is in Normal Mode. 0 = Indicates the CAN core is not in Normal mode.
2	SLEEP	Read Only	0	Sleep Mode. Indicates the CAN core is in Sleep mode. 1 = Indicates the CAN core is in Sleep mode. 0 = Indicates the CAN core is not in Sleep mode.
1	LBACK	Read Only	0	Loopback Mode. Indicates the CAN core is in Loopback mode. 1 = Indicates the CAN core is in Loopback mode. 0 = Indicates the CAN core is not in Loopback mode.
0	CONFIG	Read Only	1	Configuration Mode Indicator. Indicates the CAN core is in Configuration mode. 1 = Indicates the CAN core is in Configuration mode. 0 = Indicates the CAN core is not in Configuration mode.

Interrupt Registers

The CAN core contains a single interrupt line, but several interrupt conditions. Interrupts are controlled by the interrupt status, enable, and clear registers.

Interrupt Status Register

The Interrupt Status Register (ISR) contains bits that are set when a particular interrupt condition occurs. If the corresponding mask bit in the Interrupt Enable Register is set, an interrupt is generated.

Interrupt bits in the ISR can be cleared by writing to the Interrupt Clear Register. For all bits in the ISR, a set condition takes priority over the clear condition and the bit continues to remain 1. Table 2-20 shows the bit positions in the ISR and Table Table 2-21 describes the ISR bits.

Table 2-20: **Interrupt Status Register BIT Positions**

31:12	11	10	9	8	7	6
Reserved	WKUP	SLP	BSOFF	ERROR	RXNEMP	RXOFLW
5	**4**	**3**	**2**	**1**	**0**	
RXUFLW	RXOK	TXBFLL	TXFLL	TXOK	ARBLST	

Table 2-21: **Interrupt Status Register Bits**

Bits	Name	Access	Default Value	Description
31:12	Reserved	Read/Write	0	Reserved. Reserved for future expansion.
11	WKUP	Read Only	0	Wake up Interrupt. A 1 indicates that the CAN core entered Normal mode from Sleep Mode. This bit can be cleared by writing to the ICR. This bit is also cleared when a 0 is written to the CEN bit in the SRR.
10	SLP	Read Only	0	Sleep Interrupt. A 1 indicates that the CAN core entered Sleep mode. This bit can be cleared by writing to the ICR. This bit is also cleared when a 0 is written to the CEN bit in the SRR.
9	BSOFF	Read Only	0	Bus Off Interrupt. A 1 indicates that the CAN core entered the Bus Off state. This bit can be cleared by writing to the ICR. This bit is also cleared when a 0 is written to the CEN bit in the SRR.

Table 2-21: **Interrupt Status Register Bits *(Cont'd)***

Bits	Name	Access	Default Value	Description
8	ERROR	Read Only	0	Error Interrupt. A 1 indicates that an error occurred during message transmission or reception. This bit can be cleared by writing to the ICR. This bit is also cleared when a 0 is written to the CEN bit in the SRR.
7	RXNEMP	Read Only	0	Receive FIFO Not Empty Interrupt. A 1 indicates that the Receive FIFO is not empty. This bit can be cleared only by writing to the ICR.
6	RXOFLW	Read Only	0	RX FIFO Overflow Interrupt. A 1 indicates that a message has been lost. This condition occurs when a new message is being received and the Receive FIFO is Full. This bit can be cleared by writing to the ICR. This bit is also cleared when a 0 is written to the CEN bit in the SRR.
5	RXUFLW	Read Only	0	RX FIFO Underflow Interrupt. A 1 indicates that a read operation was attempted on an empty RX FIFO. This bit can be cleared only by writing to the ICR.
4	RXOK	Read Only	0	New Message Received Interrupt. A 1 indicates that a message was received successfully and stored into the RX FIFO. This bit can be cleared by writing to the ICR. This bit is also cleared when a 0 is written to the CEN bit in the SRR.
3	TXBFLL	Read Only	0	High Priority Transmit Buffer Full Interrupt. A 1 indicates that the High Priority Transmit Buffer is full. The status of the bit is unaffected if write transactions occur on the High Priority Transmit Buffer when it is already full. This bit can be cleared only by writing to the ICR.
2	TXFLL	Read Only	0	Transmit FIFO Full Interrupt. A 1 indicates that the TX FIFO is full. The status of the bit is unaffected if write transactions occur on the Transmit FIFO when it is already full. This bit can be cleared only by writing to the Interrupt Clear Register.
1	TXOK(1)	Read Only	0	Transmission Successful Interrupt. A 1 indicates that a message was transmitted successfully. This bit can be cleared by writing to the ICR. This bit is also cleared when a 0 is written to the CEN bit in the SRR.

Table 2-21: **Interrupt Status Register Bits *(Cont'd)***

Bits	Name	Access	Default Value	Description
0	ARBLST	Read Only	0	Arbitration Lost Interrupt. A 1 indicates that arbitration was lost during message transmission. This bit can be cleared by writing to the ICR. This bit is also cleared when a 0 is written to the CEN bit in the SRR.

Notes:

1. In Loopback mode, both TXOK and RXOK bits are set. The RXOK bit is set before the TXOK bit.

Interrupt Enable Register

The Interrupt Enable Register (IER) is used to enable interrupt generation. Table 2-22 shows the bit positions in the IER and Table 2-23 describes the IER bits.

Table 2-22: **Interrupt Enable Register Bit Positions**

31:12	11	10	9	8	7	6
Reserved	EWKUP	ESLP	EBSOFF	EERROR	ERXNEMP	ERXOFLW
5	**4**	**3**	**2**	**1**	**0**	
ERXUFLW	ERXOK	ETXBFLL	ETXFLL	ETXOK	EARBLST	

Table 2-23: **Interrupt Enable Register Bits**

Bits	Name	Access	Default Value	Description
31:12	Reserved	Read/Write	0	Reserved. Reserved for future expansion.
11	EWKUP	Read/Write	0	Enable Wake up Interrupt. Writes to this bit enable or disable interrupts when the WKUP bit in the ISR is set. 1 = Enable interrupt generation if WKUP bit in ISR is set. 0 = Disable interrupt generation if WKUP bit in ISR is set.
10	ESLP	Read/Write	0	Enable Sleep Interrupt. Writes to this bit enable or disable interrupts when the SLP bit in the ISR is set. 1 = Enable interrupt generation if SLP bit in ISR is set. 0 = Disable interrupt generation if SLP bit in ISR is set.
9	EBSOFF	Read/Write	0	Enable Bus OFF Interrupt. Writes to this bit enable or disable interrupts when the BSOFF bit in the ISR is set. 1 = Enable interrupt generation if BSOFF bit in ISR is set. 0 = Disable interrupt generation if BSOFF bit in ISR is set.

Table 2-23: **Interrupt Enable Register Bits *(Cont'd)***

Bits	Name	Access	Default Value	Description
8	EERROR	Read/Write	0	Enable Error Interrupt. Writes to this bit enable or disable interrupts when the ERROR bit in the ISR is set. 1 = Enable interrupt generation if ERROR bit in ISR is set. 0 = Disable interrupt generation if ERROR bit in ISR is set.
7	ERXNEMP	Read/Write	0	Enable Receive FIFO Not Empty Interrupt. Writes to this bit enable or disable interrupts when the RXNEMP bit in the ISR is set. 1 = Enable interrupt generation if RXNEMP bit in ISR is set. 0 = Disable interrupt generation if RXNEMP bit in ISR is set.
6	ERXOFLW	Read/Write	0	Enable RX FIFO Overflow Interrupt. Writes to this bit enable or disable interrupts when the RXOFLW bit in the ISR is set. 1 = Enable interrupt generation if RXOFLW bit in ISR is set. 0 = Disable interrupt generation if RXOFLW bit in ISR is set.
5	ERXUFLW	Read/Write	0	Enable RX FIFO Underflow Interrupt. Writes to this bit enable or disable interrupts when the RXUFLW bit in the ISR is set. 1 = Enable interrupt generation if RXUFLW bit in ISR is set. 0 = Disable interrupt generation if RXUFLW bit in ISR is set.
4	ERXOK	Read/Write	0	Enable New Message Received Interrupt. Writes to this bit enable or disable interrupts when the RXOK bit in the ISR is set. 1 = Enable interrupt generation if RXOK bit in ISR is set. 0 = Disable interrupt generation if RXOK bit in ISR is set.
3	ETXBFLL	Read/Write	0	Enable High Priority Transmit Buffer Full Interrupt. Writes to this bit enable or disable interrupts when the TXBFLL bit in the ISR is set. 1 = Enable interrupt generation if TXBFLL bit in ISR is set. 0 = Disable interrupt generation if TXBFLL bit in ISR is set.
2	ETXFLL	Read/Write	0	Enable Transmit FIFO Full Interrupt. Writes to this bit enable or disable interrupts when TXFLL bit in the ISR is set. 1 = Enable interrupt generation if TXFLL bit in ISR is set. 0 = Disable interrupt generation if TXFLL bit in ISR is set.

Table 2-23: **Interrupt Enable Register Bits *(Cont'd)***

Bits	Name	Access	Default Value	Description
1	ETXOK	Read/Write	0	Enable Transmission Successful Interrupt. Writes to this bit enable or disable interrupts when the TXOK bit in the ISR is set. 1 = Enable interrupt generation if TXOK bit in ISR is set. 0 = Disable interrupt generation if TXOK bit in ISR is set.
0	EARBLST	Read/Write	0	Enable Arbitration Lost Interrupt. Writes to this bit enable or disable interrupts when the ARBLST bit in the ISR is set. 1 = Enable interrupt generation if ARBLST bit in ISR is set. 0 = Disable interrupt generation if ARBLST bit in ISR is set.

Interrupt Clear Register

The Interrupt Clear Register (ICR) is used to clear interrupt status bits. Table 2-24 shows the bit positions in the ICR and Table 2-25 describes the ICR bits.

Table 2-24: **Interrupt Clear Register Bit Positions**

31:12	11	10	9	8	7	6
Reserved	CWKUP	CSLP	CBSOFF	CERROR	CRXNEMP	CRXOFLW
5	**4**	**3**	**2**	**1**	**0**	
CRXUFLW	CRXOK	CTXBFLL	CTXFLL	CTXOK	CARBLST	

Table 2-25: **Interrupt Clear Register Bit Descriptions**

Bits	Name	Access	Default Value	Description
31:12	Reserved	Read/Write	0	Reserved. Reserved for future expansion.
11	CWKUP	Write Only	0	Clear Wake up Interrupt. Writing a 1 to this bit clears the WKUP bit in the ISR.
10	CSLP	Write Only	0	Clear Sleep Interrupt. Writing a 1 to this bit clears the SLP bit in the ISR.
9	CBSOFF	Write Only	0	Clear Bus Off Interrupt. Writing a 1 to this bit clears the BSOFF bit in the ISR.
8	CERROR	Write Only	0	Clear Error Interrupt. Writing a 1 to this bit clears the ERROR bit in the ISR.
7	CRXNEMP	Write Only	0	Clear Receive FIFO Not Empty Interrupt. Writing a 1 to this bit clears the RXNEMP bit in the ISR.
6	CRXOFLW	Write Only	0	Clear RX FIFO Overflow Interrupt. Writing a 1 to this bit clears the RXOFLW bit in the ISR.

Table 2-25: **Interrupt Clear Register Bit Descriptions *(Cont'd)***

Bits	Name	Access	Default Value	Description
5	CRXUFLW	Write Only	0	Clear RX FIFO Underflow Interrupt. Writing a 1 to this bit clears the RXUFLW bit in the ISR.
4	CRXOK	Write Only	0	Clear New Message Received Interrupt. Writing a 1 to this bit clears the RXOK bit in the ISR.
3	CTXBFLL	Write Only	0	Clear High Priority Transmit Buffer Full Interrupt. Writing a 1 to this bit clears the TXBFLL bit in the ISR.
2	CTXFLL	Write Only	0	Clear Transmit FIFO Full Interrupt. Writing a 1 to this bit clears the TXFLL bit in the ISR.
1	CTXOK	Write Only	0	Clear Transmission Successful Interrupt. Writing a 1 to this bit clears the TXOK bit in the ISR.
0	CARBLST	Write Only	0	Clear Arbitration Lost Interrupt. Writing a 1 to this bit clears the ARBLST bit in the ISR.

Message Storage

The CAN core has a Receive FIFO (RX FIFO) for storing received messages. The RX FIFO depth is configurable and can store up to 64 messages. Messages that pass any of the acceptance filters are stored in the RX FIFO. When no acceptance filter has been selected, all received messages are stored in the RX FIFO.

The CAN core has a configurable Transmit FIFO (TX FIFO) that can store up to 64 messages. The CAN core also has a High Priority Transmit Buffer (TX HPB), with storage for one message. When a higher priority message needs to be sent, write the message to the High Priority Transmit Buffer. The message in the Transmit Buffer has priority over messages in the TX FIFO.

Message Transmission and Reception

These following rules apply for message transmission and reception:

- A message in the TX High Priority Buffer (TX HPB) has priority over messages in the TX FIFO.
- In case of arbitration loss or errors during the transmission of a message, the CAN core tries to retransmit the message. No subsequent message, even a newer, higher priority message, is transmitted until the original message is transmitted without errors or arbitration loss.
- The messages in the TX FIFO, TX HPB and RX FIFO are retained even if the CAN core enters Bus off state or Configuration mode.

Message Structure

Each message is 16 bytes. Byte ordering for CAN message structure is shown in Table 2-26 through Table 2-29.

Table 2-26: **Message Identifier [IDR]**

31:21	20	19	18:1	0
ID [28:18]	SRR/RTR	IDE	ID[17:0]	RTR

Table 2-27: **Data Length Code [DLCR]**

31:28	27:0
DLC [3:0]	Reserved

Table 2-28: **Data Word 1 [DW1R]**

31:24	23:16	15:8	7:0
DB0[7:0]	DB1[7:0]	DB2[7:0]	DB3[7:0]

Table 2-29: **Data Word 2 [DW2R]**

31:24	23:16	15:8	7:0
DB4[7:0]	DB5[:0]	DB6[7:0]	DB7[7:0]

Reads from RX FIFO

All 16 bytes must be read from the RX FIFO to receive the complete message. The first word read (four bytes) returns the identifier of the received message (IDR). The second read returns the Data Length Code (DLC) field of the received message (DLCR). The third read returns Data Word 1 (DW1R), and the fourth read returns Data Word 2 (DW2R).

All four words have to be read for each message, even if the message contains less than eight data bytes. Write transactions to the RX FIFO are ignored. Reads from an empty RX FIFO return invalid data.

Writes to TX FIFO and High Priority TX Buffer

When writing to the TX FIFO or the TX HPB, all 16 bytes must be written. The first word written (four bytes) is the Identifier (IDR). The second word written is the DLC field (DLCR). The third word written is Data Word 1 (DW1R) and the fourth word written is Data Word 2 (DW2R).

When transmitting on the CAN bus, the CAN core transmits the data bytes in the following order (DB0, DB1, DB2, DB3, DB4, DB5, DB6, DB7). The MSB of a data byte is transmitted first.

IMPORTANT: *All four words must be written for each message, including messages containing fewer than eight data bytes. Read transactions from the TX FIFO or the TX High Priority Buffer return 0.*

- 0s must be written to unused Data Fields in the DW1R and DW2R registers
- 0s must be written to Bits[31:4] in the DLCR
- 0s must be written to Identifier of Received Message (IDR) [31:13] for standard frames

The Identifier (IDR) word contains the identifier field of the CAN message. Two formats exist for the Identifier field of the CAN message frame:

- **Standard Frames**: Standard frames have an 11-bit identifier field called the Standard Identifier. Only the ID[28:18], Software Reset Register/Remote Transmission Request (SRR/RTR), and IDE bits are valid. ID[28:18] is the 11-bit identifier. The SRR/RTR bit differentiates between data and remote frames. IDE is 0 for standard frames. The other bit fields are not used.
- **Extended Frames**: Extended frames have an 18-bit identifier extension in addition to the Standard Identifier. All bit fields are valid. The RTR bit is used to differentiate between data and remote frames (The SRR/RTR bit and IDE bit are both 1 for all Extended Frames).

Table 2-30 provides bit descriptions for the Identifier Word. Table 2-31 describes the DLC Word bits. Table 2-32 describes the Data Word 1 and Data Word 2 bits.

Table 2-30: **Identifier Word Bits**

Bits	Name	Access	Default Value	Description
31:21	ID[28:18]	Reads from RX FIFO Writes to TX FIFO and TX HPB	0	Standard Message ID. The Identifier portion for a Standard Frame is 11 bits. These bits indicate the Standard Frame ID. This field is valid for both Standard and Extended Frames.
20	SRR/RTR	Reads from RX FIFO Writes to TX FIFO and TX HPB	0	Substitute Remote Transmission Request. This bit differentiates between data frames and remote frames. Valid only for Standard Frames. For Extended frames this bit is 1. 1 = Indicates that the message frame is a Remote Frame. 0 = Indicates that the message frame is a Data Frame.
19	IDE	Reads from RX FIFO Writes to TX FIFO and TX HPB	0	Identifier Extension. This bit differentiates between frames using the Standard Identifier and those using the Extended Identifier. Valid for both Standard and Extended Frames. 1 = Indicates the use of an Extended Message Identifier. 0= Indicates the use of a Standard Message Identifier.
18:1	ID[18:0]	Reads from RX FIFO Writes to TX FIFO and TX HPB	0	Extended Message ID. This field indicates the Extended Identifier. Valid only for Extended Frames. For Standard Frames, reads from this field return 0s. For Standard Frames, writes to this field should be 0s.
0	RTR	Reads from RX FIFO Writes to TX FIFO and TX HPB	0	Remote Transmission Request. This bit differentiates between data frames and remote frames. Valid only for Extended Frames. 1 = Indicates the message object is a Remote Frame. 0 = Indicates the message object is a Data Frame. For Standard Frames, reads from this bit returns 0. For Standard Frames, writes to this bit should be 0.

Table 2-31: **DLC Word Bits**

Bits	Name	Access	Default Value	Description
31:28	DLC	Read/Write	0	Data Length Code. This is the data length portion of the control field of the CAN frame. This indicates the number valid data bytes in Data Word 1 and Data Word 2 registers.
27:0	Reserved	Read/Write		Reads from this field return 0s. Writes to this field should be 0s.

Table 2-32: **Data Word 1 and Data Word 2 Bits**

Register	Field	Access	Default Value	Description
DW1R [31:24]	DB0[7:0]	Read/Write	0	Data Byte 0. Reads from this field return invalid data if the message has no data.
DW1R [23:16]	DB1[7:0]	Read/Write	0	Data Byte 1. Reads from this field return invalid data if the message has only 1 byte of data or fewer.
DW1R [15:8]	DB2[7:0]	Read/Write	0	Data Byte 2. Reads from this field return invalid data if the message has 2 bytes of data or fewer.
DW1R [7:0]	DB3[7:0]	Read/Write	0	Data Byte 3. Reads from this field return invalid data if the message has 3 bytes of data or fewer.
DW2R [31:24]	DB4[7:0]	Read/Write	0	Data Byte 4. Reads from this field return invalid data if the message has 4 bytes of data or fewer.
DW2R [23:16]	DB5[7:0]	Read/Write	0	Data Byte 5. Reads from this field return invalid data if the message has 5 bytes of data or fewer.
DW2R [15:8]	DB6[7:0]	Read/Write	0	Data Byte 6. Reads from this field return invalid data if the message has 6 bytes of data or fewer.
DW2R [7:0]	DB7[7:0]	Read/Write	0	Data Byte 7. Reads from this field return invalid data if the message has 7 bytes of data or fewer.

Acceptance Filters

The number of acceptance filters is configurable from 0 to 4. The Number of Acceptance Filters parameter specifies the number of acceptance filters chosen. Each acceptance filter has an Acceptance Filter Mask Register and an Acceptance Filter ID Register.

Acceptance filtering is performed in this sequence:

1. The incoming Identifier is masked with the bits in the Acceptance Filter Mask Register.
2. The Acceptance Filter ID Register is also masked with the bits in the Acceptance Filter Mask Register.
3. Both resulting values are compared.
4. If both these values are equal, then the message is stored in the RX FIFO.
5. Acceptance filtering is processed by each of the defined filters. If the incoming identifier passes through any acceptance filter, then the message is stored in the RX FIFO.

These rules apply to the acceptance filtering process:

- If no acceptance filters are selected (for example, if all the valid UAF bits in the AFR register are 0s or if the parameter Number of Acceptance Filters = 0), all received messages are stored in the RX FIFO.
- If the number of acceptance filters is ≥ 1, all the Acceptance Filter Mask Register and the Acceptance Filter ID Register locations can be written to and read from. However, the use of these filter pairs for acceptance filtering is governed by the existence of the UAF bits in the AFR register.

Acceptance Filter Register

The Acceptance Filter Register (AFR) defines which acceptance filters to use. Each Acceptance Filter ID Register (AFIR) and Acceptance Filter Mask Register (AFMR) pair is associated with a UAF bit.

When the UAF bit is 1, the corresponding acceptance filter pair is used for acceptance filtering. When the UAF bit is 0, the corresponding acceptance filter pair is not used for acceptance filtering. The AFR exists only if the Number of Acceptance Filters parameter is not set to 0.

To modify an acceptance filter pair in Normal mode, the corresponding UAF bit in this register must be set to 0. After the acceptance filter is modified, the corresponding UAF bit must be set to 1.

These conditions govern the number of UAF bits that can exist in the AFR.

- If the number of acceptance filters is 1: UAF1 bit exists
- If the number of acceptance filters is 2: UAF1 and UAF2 bits exist
- If the number of acceptance filters is 3: UAF1, UAF2, and UAF3 bits exist
- If the number of acceptance filters is 4: UAF1, UAF2, UAF3, and UAF4 bits exist
- UAF bits that do not exist are not written to
- Reads from UAF bits that do not exist return 0s
- If all existing UAF bits are set to 0, then all received messages are stored in the RX FIFO
- If the UAF bits are changed from a 1 to 0 during reception of a CAN message, the message might not be stored in the RX FIFO.

Table 2-33 shows the bit positions in the AFR and Table 2-34 describes the AFR bits.

Table 2-33: **Acceptance Filter Register Bit Positions**

31:4	3	2	1	0
Reserved	UAF4	UAF3	UAF2	UAF1

Table 2-34: **Acceptance Filter Register Bits**

Bits	Name	Access	Default Value	Description
31:4	Reserved	Read/Write	0	Reserved Reserved for future expansion.
3	UAF4	Read/Write	0	Use Acceptance Filter Number 4. Enables the use of acceptance filter pair 4. 1 = Indicates Acceptance Filter Mask Register 4 and Acceptance Filter ID Register 4 are used for acceptance filtering. 0 = Indicates Acceptance Filter Mask Register 4 and Acceptance Filter ID Register 4 are not used for acceptance filtering.
2	UAF3	Read/Write	0	Use Acceptance Filter Number 3. Enables the use of acceptance filter pair 3. 1 = Indicates Acceptance Filter Mask Register 3 and Acceptance Filter ID Register 3 are used for acceptance filtering. 0 = Indicates Acceptance Filter Mask Register 3 and Acceptance Filter ID Register 3 are not used for acceptance filtering.
1	UAF2	Read/Write	0	Use Acceptance Filter Number 2. Enables the use of acceptance filter pair 2. 1 = Indicates Acceptance Filter Mask Register 2 and Acceptance Filter ID Register 2 are used for acceptance filtering. 0 = Indicates Acceptance Filter Mask Register 2 and Acceptance Filter ID Register 2 are not used for acceptance filtering.
0	UAF1	Read/Write	0	Use Acceptance Filter Number 1. Enables the use of acceptance filter pair 1. 1 = Indicates Acceptance Filter Mask Register 1 and Acceptance Filter ID Register 1 are used for acceptance filtering. 0 = Indicates Acceptance Filter Mask Register 1 and Acceptance Filter ID Register 1 are not used for acceptance filtering.

Acceptance Filter Mask Register

The Acceptance Filter Mask Registers (AFMR) contain mask bits used for acceptance filtering. The incoming message identifier portion of a message frame is compared with the message identifier stored in the acceptance filter ID register. The mask bits define which identifier bits stored in the acceptance filter ID register are compared to the incoming message identifier.

There are at most four AFMRs. These registers are stored in a block RAM. Asserting a software reset or system reset does not clear register contents. If the number of acceptance filters is ≥ 1, then all the four AFMRs are defined. These registers can be read from and written to. However, filtering operations are only performed on the number of filters defined by the Number of Acceptance Filters parameter. These registers are written to only when the corresponding UAF bits in the AFR are 0 and ACFBSY bit in the SR is 0.

These conditions govern AFMRs:

- If the number of acceptance filters is 1: AFMR 1 is used for acceptance filtering.
- If the number of acceptance filters is 2: AFMR 1 and AFMR 2 are used for acceptance filtering.
- If the number of acceptance filters is 3: AFMR 1, AFMR 2, and AFMR 3 are used for acceptance filtering.
- If the number of acceptance filters is 4: AFMR 1, AFMR 2, AFMR 3, and AFMR 4 are used for acceptance filtering.
- Extended Frames. All bit fields (AMID [28:18], AMSRR, AMIDE, AMID [17:0], and AMRTR) need to be defined.
- Standard Frames. Only AMID [28:18], AMSRR and AMIDE need to be defined. AMID [17:0] and AMRTR should be written as 0.

Table 2-35 shows the bit positions in the AFMR and Table 2-36 describes the AFMR bits.

Table 2-35: **Acceptance Filter Mask Registers Bit Positions**

31 =:21	20	19	18:1	0
AMID[28:18]	AMSRR	AMIDE	AMID[17:0]	AMRTR

Table 2-36: **Acceptance Filter Mask Bit Descriptions**

Bits	Name	Access	Default Value	Description
31:21	AMID[28:18]	Read/ Write	0	Standard Message ID Mask. These bits are used for masking the Identifier in a Standard Frame. 1= Indicates the corresponding bit in Acceptance Mask ID Register is used when comparing the incoming message identifier. 0 = Indicates the corresponding bit in Acceptance Mask ID Register is not used when comparing the incoming message identifier.
20	AMSRR	Read/ Write	0	Substitute Remote Transmission Request Mask. This bit is used for masking the RTR bit in a Standard Frame. 1= Indicates the corresponding bit in Acceptance Mask ID Register is used when comparing the incoming message identifier. 0 = Indicates the corresponding bit in Acceptance Mask ID Register is not used when comparing the incoming message identifier.
19	AMIDE	Read/ Write	0	Identifier Extension Mask. Used for masking the IDE bit in CAN frames. 1= Indicates the corresponding bit in Acceptance Mask ID Register is used when comparing the incoming message identifier. 0 = Indicates the corresponding bit in Acceptance Mask ID Register is not used when comparing the incoming message identifier. If AMIDE = 1 and the AIIDE bit in the corresponding Acceptance ID register is 0, this mask is applicable to only Standard frames. If AMIDE = 1 and the AIIDE bit in the corresponding Acceptance ID register is 1, this mask is applicable to only extended frames. If AMIDE = 0 this mask is applicable to both Standard and Extended frames.
18:1	AMID[17:0]	Read/ Write	0	Extended Message ID Mask. These bits are used for masking the Identifier in an Extended Frame. 1= Indicates the corresponding bit in Acceptance Mask ID Register is used when comparing the incoming message identifier. 0 = Indicates the corresponding bit in Acceptance Mask ID Register is not used when comparing the incoming message identifier.
0	AMRTR	Read/ Write	0	Remote Transmission Request Mask. This bit is used for masking the RTR bit in an Extended Frame. 1= Indicates the corresponding bit in Acceptance Mask ID Register is used when comparing the incoming message identifier. 0 = Indicates the corresponding bit in Acceptance Mask ID Register is not used when comparing the incoming message identifier.

Acceptance Filter ID Register

The Acceptance Filter ID registers (AFIR) contain Identifier bits, which are used for acceptance filtering. There are at most four Acceptance Filter ID Registers. These registers are stored in a block RAM. Asserting a software reset or system reset does not clear the contents of these registers. If the number of acceptance filters is ≥ 1, then all four AFIRs are defined. These registers can be read from and written to. These registers should be written to only when the corresponding UAF bits in the AFR are 0 and ACFBSY bit in the SR is 0.

These conditions govern the use of the AFIRs:

- If the number of acceptance filters is 1: AFIR 1 is used for acceptance filtering.
- If the number of acceptance filters is 2: AFIR 1 and AFIR 2 are used for acceptance filtering.
- If the number of acceptance filters is 3: AFIR 1, AFIR 2, and AFIR 3 are used for acceptance filtering.
- If the number of acceptance filters is 4: AFIR 1, AFIR 2, AFIR 3, and AFIR 4 are used for acceptance filtering.
- Extended Frames. All the bit fields (AIID [28:18], AISRR, AIIDE, AIID [17:0,] and AIRTR) must be defined.
- Standard Frames. Only AIID [28:18], AISRR and AIIDE need to be defined. AIID [17:0] and AIRTR should be written with 0.

Table 2-37 shows AFIR bit positions, and Table 2-38 describes the AFIR bits.

Table 2-37: **Acceptance Filter ID Registers Bit Positions**

31:21	20	19	18:1	0
AIID[28:18]	AISRR	AIIDE	AIID[17:0]	AIRTR

Table 2-38: **Acceptance Filter ID Registers Bits**

Bits	Name	Access	Default Value	Description
31:21	AIID [28:18]	Read/Write	0	Standard Message ID. Standard Identifier.
20	AISRR	Read/Write	0	Substitute Remote Transmission Request. Indicates the Remote Transmission Request bit for Standard frames.
19	AIIDE	Read/Write	0	Identifier Extension. Differentiates between Standard and Extended frames.
18:1	AIID[17:0]	Read/Write	0	Extended Message ID. Extended Identifier.
0	AIRTR	Read/Write	0	Remote Transmission Request. RTR bit for Extended frames.

Designing with the Core

This chapter includes guidelines and additional information to facilitate designing with the core.

Configuring the CAN Core

This section covers the various configuration steps that must be performed to program the CAN core for operation.

The key configuration steps are detailed in this section.

- Choosing the operation mode
- Programming the configuration registers to initialize the core
- Writing messages to the TX FIFO/TX HPB
- Reading messages from the RX FIFO

Programming the Configuration Registers

These steps configure the core when the core is powered on or after system or software reset.

1. Choose the operation mode.
   - For Loopback mode, write a 1 to the LBACK bit in the MSR and 0 to the SLEEP bit in the MSR.
   - For Sleep mode, write a 1 to the SLEEP bit in the MSR and 0 to the LBACK bit in the MSR.
   - For Normal Mode, write 0s to the LBACK and SLEEP bits in the MSR.
2. Configure the Transfer Layer Configuration Registers.
   - Program the Baud Rate Prescaler Register and the Bit Timing Register to correspond to the network timing parameters and the network characteristics of the system.
3. Configure the Acceptance Filter Registers.

The number of Acceptance Filter Mask and Acceptance Filter ID Register pairs is chosen at build time. To configure these registers, do the following:

- Write a 0 to the UAF bit in the AFR register corresponding to the Acceptance Filter Mask and ID Register pair to be configured.
- Wait until the ACFBSY bit in the SR is 0.
- Write the appropriate mask information to the Acceptance Filter Mask Register.
- Write the appropriate ID information to the to the Acceptance Filter ID Register.
- Write a 1 to the UAF bit corresponding to the Acceptance Filter Mask and ID Register pair.
- Repeat the preceding steps for each Acceptance Filter Mask and ID Register pair.

4. Write to the Interrupt Enable Register to choose the bits in the Interrupt Status Register that can generate an interrupt.
5. Enable the CAN core by writing a 1 to the CEN bit in the SRR register.

Transmitting a Message

A message to be transmitted can be written to either the TX FIFO or the TX HPB. A message in the TX HPB gets priority over the messages in the TX FIFO. The TXOK bit in the ISR is set after the CAN core successfully transmits a message.

Writing a Message to the TX FIFO

All messages written to the TX FIFO should follow the format defined in Message Storage in Chapter 2.

To perform a write:

1. Poll the TXFLL bit in the SR. The message can be written into the TX FIFO when the TXFLL bit is 0.
2. Write the ID of the message to the TX FIFO ID memory location (`0x030`).
3. Write the DLC of the message to the TX FIFO DLC memory location (`0x034`).
4. Write Data Word 1 of the message to the TX FIFO DW1 memory location (`0x038`).
5. Write Data Word 2 of the message to the TX FIFO DW2 memory location (`0x03C`).

Messages can be continuously written to the TX FIFO until the TX FIFO is full. When the TX FIFO is full the TXFLL bit in the ISR and the TXFLL bit in the SR are set. If polling, the TXFLL bit in the Status Register should be polled after each write. If using interrupt mode, writes can continue until the TXFLL bit in the ISR generates an interrupt.

Writing a Message to the TX HPB

All messages written to the TX FIFO should follow the format described in Message Storage in Chapter 2.

To write a message to the TX HPB:

1. Poll the TXBFLL bit in the SR.

   The message can be written into the TX HPB when the TXBFLL bit is 0.

2. Write the ID of the message to the TX HPB ID memory location (`0x040`).
3. Write the DLC of the message to the TX HPB DLC memory location (`0x044`).
4. Write Data Word 1 of the message to the TX HPB DW1 memory location (`0x048`).
5. Write Data Word 2 of the message to the TX HPB DW2 memory location (`0x04C`).

After each write to the TX HPB, the TXBFLL bit in the Status Register and the TXBFLL bit in the Interrupt Status Register are set.

Receiving a Message

Whenever a new message is received and written into the RX FIFO, the RXNEMP bit and the RXOK bits in the ISR are set. In case of a read operation on an empty RX FIFO, the RXUFLW bit in the ISR is set.

Reading a Message from the RX FIFO

Perform these steps to read a message from the RX FIFO.

1. Poll the RXOK or RXNEMP bits in the ISR. In interrupt mode, the reads can occur after the RXOK or RXNEMP bits in the ISR generate an interrupt.
   - Read from the RX FIFO memory locations. All the locations must be read regardless of the number of data bytes in the message.
   - Read from the RX FIFO ID location (`0x050`).
   - Read from the RX FIFO DLC location (`0x054`).
   - Read from the RX FIFO DW1 location (`0x058`).
   - Read from the RX FIFO DW2 location (`0x05C`).
2. After performing the read, if there are one or more messages in the RX FIFO, the RXNEMP bit in the ISR is set. This bit can either be polled or can generate an interrupt.
3. Repeat until the FIFO is empty.

Clocking

The CAN core has two clocks: `can_clk` and `s_axi_aclk`. There is no fixed-frequency dependency between the two clocks. These conditions apply for clock frequencies:

- `can_clk` can be 8 to 24 MHz
- `s_axi_aclk` can be 8 to 100 MHz
- `can_clk` and `s_axi_aclk` can be asynchronous or can be clocked from the same source

IMPORTANT: *Either of these clocks can be sourced from external oscillator sources or generated within the FPGA. The oscillator used for `can_clk` must be compliant with the oscillator tolerance range given in the ISO 11898 -1, CAN 2.0A and CAN 2.0B standards.*

s_axi_aclk

The operating frequency for `s_axi_aclk` can be specified; using a DCM to generate `s_axi_aclk` is optional.

can_clk

The range of `can_clk` clock is 8 – 24 MHz.

The designer determines whether a DCM or an external oscillator is used to generate the `can_clk`. If an external oscillator is used, it should meet the tolerance requirements specified in the *ISO 11898-1*, *CAN 2.0A* and *CAN 2.0B* standards.

Resets

Two different reset mechanisms are provided for the CAN core. The `s_axi_aresetn` input mentioned in Table 2-4, page 16 acts as the system reset. Apart from the system reset, a software reset is provided through the SRST bit in the SRR register. The software and system reset both reset the complete CAN core (both the Object Layer and the Transfer Layer as shown in Figure 1-1, page 5).

Software Reset

The software reset can be enabled by writing a 1 to the SRST bit in the SRR Register. When a software reset is asserted, all the configuration registers including the SRST bit in the SRR Register are reset to their default values. Read/Write transactions can be performed starting at the next valid transaction window.

System Reset

The system reset can be enabled by driving a 0 on the `s_axi_aresetn` input. All the configuration registers are reset to their default values. Read/Write transactions cannot be performed when the `s_axi_aresetn` input is 0.

Exceptions

The contents of the acceptance filter mask registers and acceptance filter ID registers are not cleared when the software reset or system reset is asserted.

Reset Synchronization

A reset synchronizer resets each clock domain in the core. Because of this, some latency exists between the assertion of reset and the actual reset of the core.

Interrupts

The CAN core uses a hard-vector interrupt mechanism. It has a single interrupt input (`IP2Bus_IntrEvent`) to indicate an interrupt. Interrupts are indicated by asserting the `IP2Bus_IntrEvent` line (transition of the `IP2Bus_IntrEvent` line from a logic 0 to a logic 1).

Events such as errors on the bus line, message transmission and reception, FIFO overflows and underflow conditions can generate interrupts. During power on, the Interrupt line is driven Low.

The Interrupt Status Register (ISR) indicates the interrupt status bits. These bits are set and cleared regardless of the status of the corresponding bit in the Interrupt Enable Register (IER). The IER handles the interrupt-enable functionality. The clearing of a status bit in the ISR is handled by writing a 1 to the corresponding bit in the Interrupt Clear Register (ICR).

Two conditions cause the `IP2Bus_IntrEvent` line to be asserted:

- If a bit in the ISR is 1 and the corresponding bit in the IER is 1.
- Changing an IER bit from a 0 to 1 when the corresponding bit in the ISR is already 1.

Two conditions cause the `IP2Bus_IntrEvent` line to be deasserted:

- Clearing a 1 bit in the ISR (by writing a 1 to the corresponding bit in the ICR provided the corresponding bit in the IER is 1).
- Changing an IER bit from 1 to 0 when the corresponding bit in the ISR is 1.

When both deassertion and assertion conditions occur simultaneously, the `IP2Bus_IntrEvent` line is deasserted first, and is reasserted if the assert condition remains TRUE.

Design Flow Steps

This chapter describes customizing and generating the core, constraining the core, and the simulation, synthesis and implementation steps that are specific to this IP core. More detailed information about the standard Vivado® design flows and the Vivado IP integrator can be found in the following Vivado Design Suite user guides:

- *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) [Ref 5]
- *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 3]
- *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 6]
- *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 7]

Customizing and Generating the Core

This chapter includes information about using Xilinx® tools to customize and generate the core in the Vivado Design Suite.

If you are customizing and generating the core in the IP integrator, see the *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) [Ref 5] for detailed information. IP integrator might auto-compute certain configuration values when validating or generating the design. To view the parameter value you can run the `validate_bd_design` command in the Tcl Console.

Vivado Integrated Design Environment

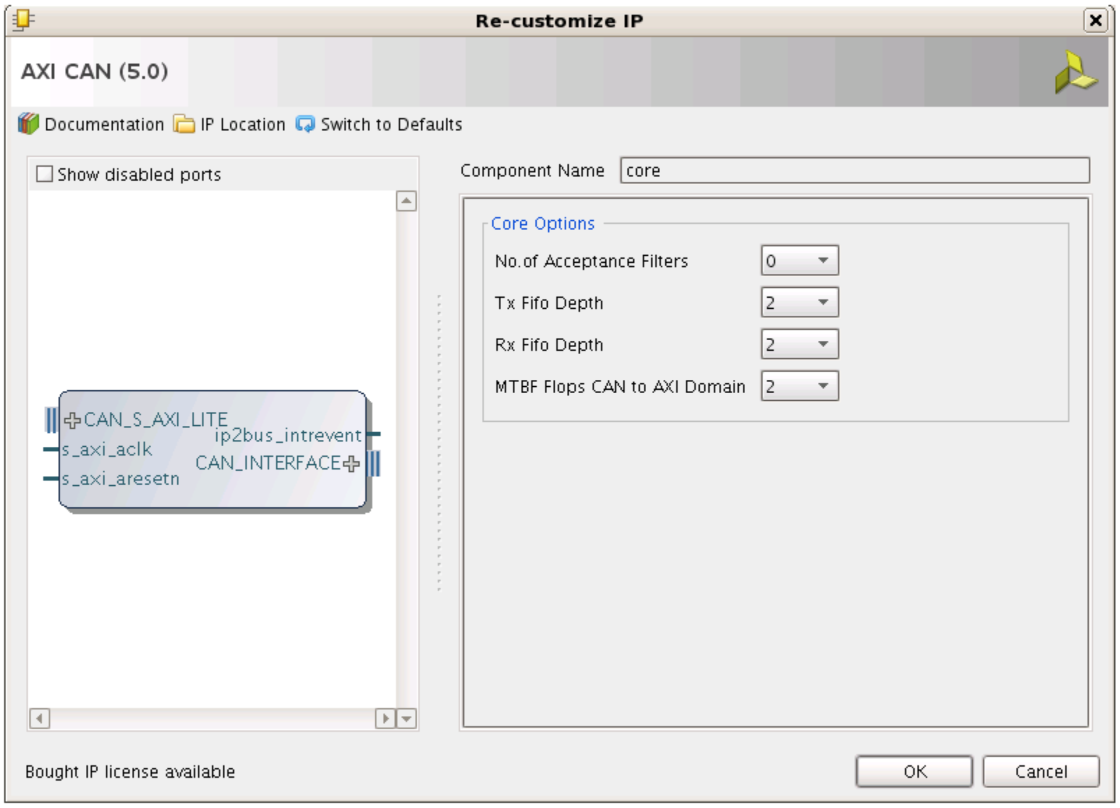
You can customize the IP for use in your design by specifying values for the various parameters associated with the core using the following steps:

1. Select the IP from the IP catalog.
2. Double-click the selected IP or select the Customize IP command from the toolbar or right-click menu.

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 3] and the *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 6].

Note: Figure in this chapter is an illustration of the CAN in the Vivado Integrated Design Environment (IDE). This layout might vary from the current version.

Figure 4-1 shows the main CAN customization screen, which is used to set the component name and core options, described in the following sections.



Re-customize IP

AXI CAN (5.0)

Component Name: core

Core Options

No.of Acceptance Filters: 0

Tx Fifo Depth: 2

Rx Fifo Depth: 2

MTBF Flops CAN to AXI Domain: 2

Bought IP license available

Figure 4-1: **Vivado Main Screen**

Component Name

The Component Name is the base name of the output files generated for this core.

IMPORTANT: *The name must begin with a letter and be composed of the following characters: a to z, A to Z, 0 to 9 and "_."*

Core Options

Number of Acceptance Filters

This specifies the number of acceptance filter pairs used by the CAN core. Each acceptance filter pair consists of a Mask Register and an ID register. These registers can be configured so that a specific identifier or a range of identifiers can be received. Valid range is from 0 to 4.

TX FIFO Depth

The TX FIFO depth is measured in terms of the number of CAN messages. For example, a TX FIFO with a depth of 2 can hold at most 2 CAN messages.

Valid values are 2, 4, 8, 16, 32, or 64 to configure the depth of the TX FIFO.

Note: Depths 8, 16, 32 and 64 are only valid when the **MTBF Flip-flops CAN to AXI Domain** selection is set to 4.

RX FIFO Depth

The RX FIFO depth is measured in terms of the number of CAN messages. For example, an RX FIFO with a depth of 2 can hold at most 2 CAN messages.

Valid values are 2, 4, 8, 16, 32, or 64 to configure the depth of the RX FIFO.

Note: Depths 8, 16, 32 and 64 are only valid when the **MTBF Flip-flops CAN to AXI Domain** selection is set to 4.

MTBF Flip-flops CAN to AXI Domain

Number of MTBF flip-flops used for synchronizing signals from CAN to AXI domain. Selecting a value of four improves the Mean Time between Failures (MBTF) for the core.

- MTBF flip-flops for synchronizing signals from the AXI to the CAN domain is fixed at two in the design.
- Core is compliance-tested with a value of two for CAN to AXI and AXI to CAN synchronizer flip-flops.

Output Generation

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 3].

Constraining the Core

This section contains information about constraining the core in the Vivado Design Suite.

Required Constraints

CAN and AXI clocks are treated as asynchronous to each other and the core writes out appropriate clock domain crossing constraints.

Table 4-1 shows the files delivered in the `<project_name>/<project_name>.srcs/source_1/ip/<component_name>/` directory for core constraints.

Table 4-1: **Core Constraint Files**

Name	Description
<component_name>.xdc	Core constraints

The core also delivers constraints for out-of-context (OOC) mode.

Table 4-2 shows the files delivered in the `<project_name>/<project_name>.srcs/source_1/ip/<component_name>/` directory for core constraints.

Table 4-2: **OOC Constraint Files**

Name	Description
<component_name>_ooc.xdc	Out-of-context (OOC) mode constraints

Device, Package, and Speed Grade Selections

This section is not applicable for this IP core.

Clock Frequencies

- `can_clk` frequency can be 8 to 24 MHz.
- `s_axi_aclk` frequency can be 8 to 100 MHz.
- `can_clk` and `s_axi_aclk` can be asynchronous or can be clocked from the same source.

Clock Management

This section is not applicable for this IP core.

Clock Placement

This section is not applicable for this IP core.

Banking

This section is not applicable for this IP core.

Transceiver Placement

This section is not applicable for this IP core.

I/O Standard and Placement

This section is not applicable for this IP core.

Simulation

This section contains information about simulating IP in the Vivado Design Suite. For comprehensive information about Vivado simulation components, as well as information about using supported third-party tools, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 7].

Synthesis and Implementation

This section contains information about synthesis and implementation in the Vivado Design Suite. For details about synthesis and implementation, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 3].

Example Design

Overview

This chapter contains information about the example design provided in the Vivado® Design Suite environment.

The top module instantiates all components of the core and example design that are needed to implement the design in hardware, as shown in Figure 5-1. This includes clock generator, generator, and checker modules.

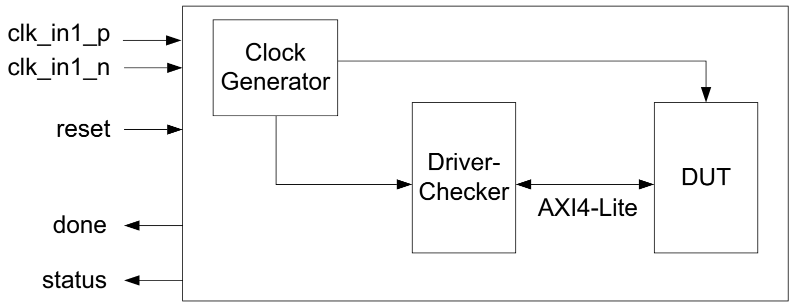


Figure 5-1: **Example Design**

This example design demonstrates the CAN transfers by register programming the core through driver module and checking the loopback transfer through checker modules.

- **Clock Generator** – Clocking wizard is used to generate two clocks, one for register interface (AXI4-Lite) and the other for CAN clock.
- **Driver-Checker** – AXI traffic generator in system test mode is used to configure the DUT and to check the status and message received in the DUT. For more information on the AXI Traffic Generator, see the *AXI Traffic Generator Product Guide* (PG125) [Ref 8].

Example Design Directory Structure

In the current project directory, a new project with the name `<componentname>_example` is created and the files are generated in the `<componentname>_example.srcs/sources_1/imports/example_design/` and `<componentname>_example.srcs/constrs_1/imports/example_design/` directories. This directory and its subdirectories contain all the source files that are required to create the CAN example design.

Table 5-1 shows the files delivered in the `<componentname>_example/<componentname>_example.srcs/` directory. This `<component_name>/example_design` directory contains the generated example design top files.

Table 5-1: **Example Design Directory**

Name	Description
<component_name>_exdes.xdc	Top-level constraints file for the example design.
<componentname>_exdes.v/vhd	Top-level HDL file for the example design.

Simulating the Example Design

For more information on Simulation, the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 7].

Simulation Results

The simulation script compiles the CAN example design and supporting simulation files. It then runs the simulation and checks to ensure that it completed successfully.

If the test passes, then the following message is displayed:

```
Test Completed Successfully
```

If the test fails, then the following message is displayed:

```
ERROR: Test Failed
```

If the test hangs, then the following message is displayed:

```
ERROR: Test did not complete (timed-out)
```

Test Bench

This chapter contains information about the test bench provided in the Vivado® Design Suite.

Figure 6-1 shows the test bench for the CAN example design. The top-level test bench generates a 200 MHz and drives an initial reset to the example design.

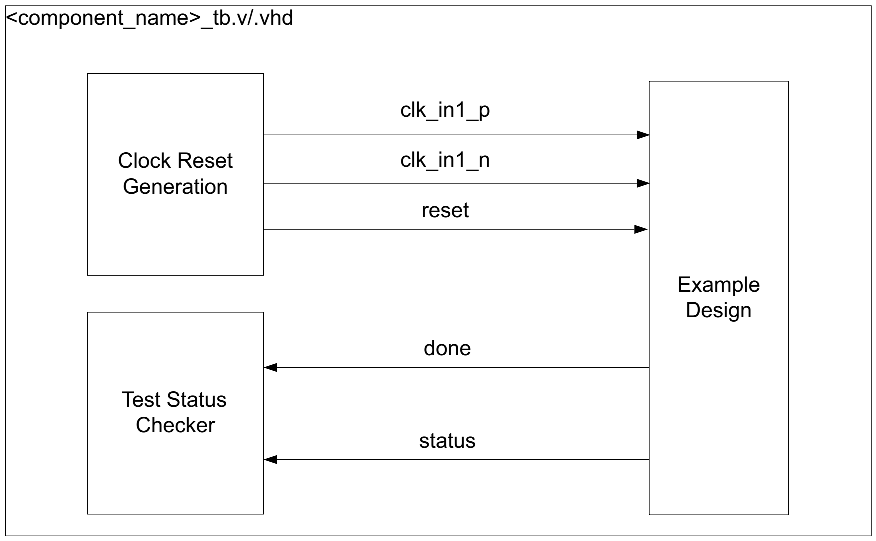


Figure 6-1: **Demonstration Test Bench**

Example Sequence

The demonstration test bench performs these tasks:

- The Mode Select register is written to select the loopback operation mode.
- The Baud Rate Prescalar register and Bit Timing registers are written.
- The Interrupt Enable register is written to enable interrupts for TXBFLL and RXOK bits. This register is read and the value read is compared with the value written.
- The Software Reset register is written to enable the CEN bit.
- Five messages are written in sequence:
    a. The first message is written to the TXHPB and is a standard data frame.
    b. The second message is written to the TX FIFO and is a standard data frame.
    c. The third message is written to the TX FIFO and is a standard remote frame.
    d. The fourth message is written to the TX FIFO and is an extended data frame.
    e. The fifth message is written to the TX FIFO and is an extended remote frame.
- After each message is written, the test bench waits for the assertion of TXOK.
- The RX FIFO is read if the RXNEMP bit is set. The message received is compared with the message previously transmitted.
- The ICR is written to clear TXOK and RXNEMP in the ISR.

Verification, Compliance, and Interoperability

Compliance Testing

Xilinx® VHDL OPB CAN core Version 1.00.a passed ISO CAN Conformance Tests. The current version of the core with the **MTBF Flip-flops CAN to AXI domain** parameter value set to two is functionally equivalent to conformance tested "Xilinx VHDL OPB CAN Core Version 1.00.a".

Migrating and Upgrading

This appendix contains information about migrating a design from ISE® to the Vivado® Design Suite, and for upgrading to a more recent version of the core. For customers upgrading in the Vivado Design Suite, important details (where applicable) about any port changes and other impact to user logic are included.

Migrating to the Vivado Design Suite

For information on migrating from the Xilinx® ISE Design Suite tools to the Vivado® Design Suite, see the *ISE to Vivado Design Suite Migration Guide* (UG911) [Ref 9].

Upgrading in the Vivado Design Suite

This section provides information about any changes to the user logic or port designations that take place when you upgrade to a more current version of this core in the Vivado Design Suite.

Parameter Changes

The parameter Cs_Mtbf_Stages was added; allowed values are 2 and 4; the default value of 2 matches with the previous version of the core.

Port Changes

No port changes.

Debugging

This appendix includes details about resources available on the Xilinx® Support website and debugging tools.

TIP: *If the IP generation halts with an error, there might be a license issue. See License Checkers in Chapter 1 for more details.*

Finding Help on Xilinx.com

To help in the design and debug process when using the CAN core, the Xilinx Support web page contains key resources such as product documentation, release notes, answer records, information about known issues, and links for obtaining further product support.

Documentation

This product guide is the main document associated with the CAN core. This guide, along with documentation related to all products that aid in the design process, can be found on the Xilinx Support web page or by using the Xilinx Documentation Navigator.

Download the Xilinx Documentation Navigator from the Downloads page. For more information about this tool and the features available, open the online help after installation.

Answer Records

Answer Records include information about commonly encountered problems, helpful information on how to resolve these problems, and any known issues with a Xilinx product. Answer Records are created and maintained daily ensuring that users have access to the most accurate information available.

Answer Records for this core can be located by using the Search Support box on the main Xilinx support web page. To maximize your search results, use proper keywords such as:

- Product name
- Tool message(s)
- Summary of the issue encountered

A filter search is available after results are returned to further target the results.

Master Answer Record for the CAN

AR: 54449

Technical Support

Xilinx provides technical support in the Xilinx Support web page for this LogiCORE™ IP product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support if you do any of the following:

- Implement the solution in devices that are not defined in the documentation.
- Customize the solution beyond that allowed in the product documentation.
- Change any section of the design labeled DO NOT MODIFY.

To contact Xilinx Technical Support, navigate to the Xilinx Support web page.

Debug Tools

There are many tools available to address CAN design issues. It is important to know which tools are useful for debugging various situations.

Vivado Design Suite Debug Feature

The Vivado® Design Suite debug feature inserts logic analyzer and virtual I/O cores directly into your design. The debug feature also allows you to set trigger conditions to capture application and integrated block port signals in hardware. Captured signals can then be analyzed. This feature in the Vivado IDE is used for logic debugging and validation of a design running in Xilinx devices.

The Vivado logic analyzer is used with the logic debug IP cores, including:

- ILA 2.0 (and later versions)
- VIO 2.0 (and later versions)

See the *Vivado Design Suite User Guide: Programming and Debugging* (UG908) [Ref 10].

Simulation Debug

The simulation debug flow for Mentor Graphics Questa Simulator (QuestaSim) is illustrated in Figure C-1. A similar approach can be used with other simulators.

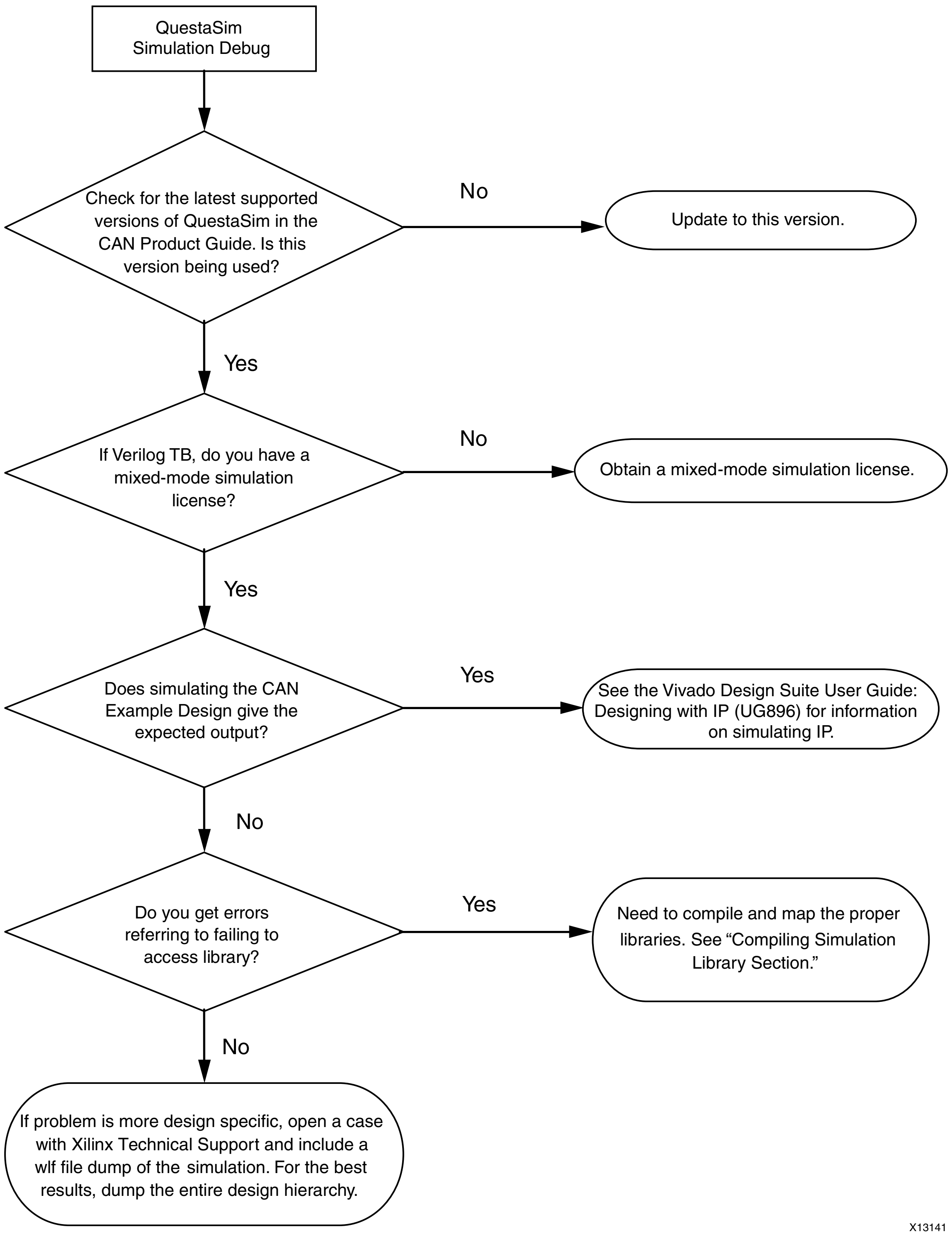

QuestaSim Simulation Debug
Check for the latest supported versions of QuestaSim in the CAN Product Guide. Is this version being used?
No
Update to this version.
Yes
If Verilog TB, do you have a mixed-mode simulation license?
No
Obtain a mixed-mode simulation license.
Yes
Does simulating the CAN Example Design give the expected output?
Yes
See the Vivado Design Suite User Guide: Designing with IP (UG896) for information on simulating IP.
No
Do you get errors referring to failing to access library?
Yes
Need to compile and map the proper libraries. See "Compiling Simulation Library Section."
No
If problem is more design specific, open a case with Xilinx Technical Support and include a wlf file dump of the simulation. For the best results, dump the entire design hierarchy.
X13141

Figure C-1: **QuestaSim Debug**

Hardware Debug

These are some common issues that might be encountered.

1. The desired baud rate is not seen on the TX/RX lines.

   Action: Ensure that the desired values are written to the BRPR and BTR registers. The actual value is one more than the value written into the registers.

2. The core is not achieving CONFIG state after it is enabled.

   Action: After the occurrence of 11 consecutive recessive bits, the CAN core clears the CONFIG bit and sets the appropriate bit in the Status register. Ensure that 11 consecutive recessive bits are seen by the core.

3. The core is enabled and the desired BRPR/BTR values are written but the lines are not toggling.

   Action: Ensure that the `can_clk` port is connected to the desired clock source.

Interface Debug

AXI4-Lite Interfaces

Read from a register that does not have all 0s as a default to verify that the interface is functional. See Figure C-2 for a read timing diagram.

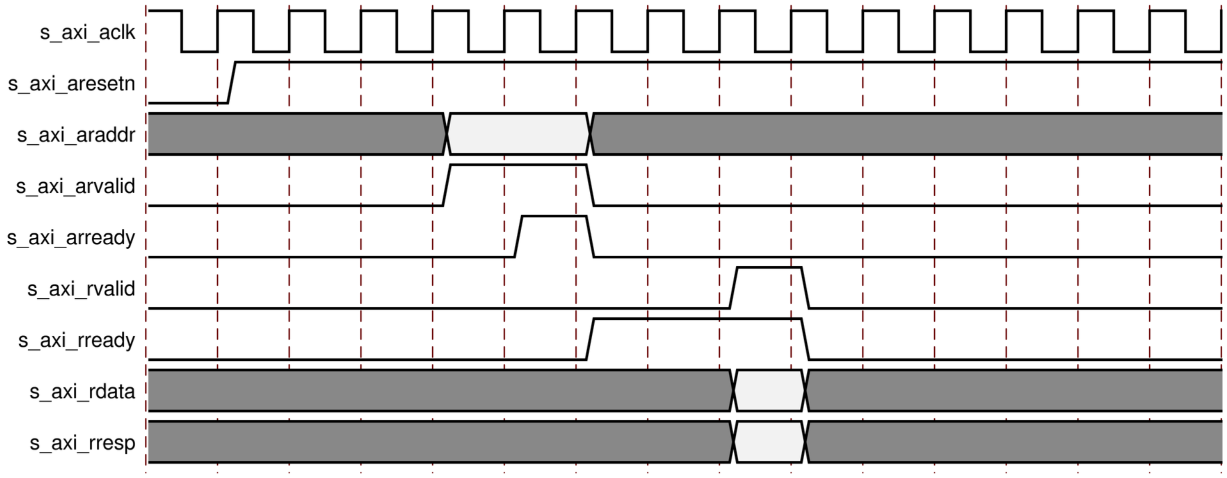


Figure C-2: **AXI4-Lite Interface Read Timing Diagram**

Output `s_axi_arready` asserts when the read address is valid, and output `s_axi_rvalid` asserts when the read data/response is valid. If the interface is unresponsive, ensure that the following conditions are met:

- The `s_axi_aclk and aclk` inputs are connected and toggling.
- The interface is not being held in reset, and `s_axi_areset` is an active-Low reset.
- The interface is enabled, and `s_axi_aclken` is active-High (if used).
- The main core clocks are toggling and that the enables are also asserted.
- If the simulation has been run, verify in simulation and/or a Vivado lab tools capture that the waveform is correct for accessing the AXI4-Lite interface.

Additional Resources and Legal Notices

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see Xilinx Support.

References

These documents provide supplemental material useful with this product guide:

1. ISO 11898-1: Road Vehicles - Interchange of Digital Information - Controller Area Network (CAN) for High-Speed Communication.
2. *Controller Area Network (CAN) version 2.0A and B Specification, Robert Bosch GmbH.*
3. *Vivado® Design Suite User Guide: Designing with IP* (UG896)
4. *Vivado AXI Reference Guide* (UG1037)
5. *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994)
6. *Vivado Design Suite User Guide: Getting Started* (UG910)
7. *Vivado Design Suite User Guide: Logic Simulation* (UG900)
8. *AXI Traffic Generator Product Guide* (PG125)
9. *ISE® to Vivado Design Suite Migration Guide* (UG911)
10. *Vivado Design Suite User Guide: Programming and Debugging* (UG908)

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
08/02/2016	5.0	Added Xilinx automotive applications disclaimer.
11/18/2015	5.0	Added support for UltraScale+ families.
04/02/2014	5.0	• Added Applications section. • Updated `s_axi_aresetn` description in Configuration Mode section. • Updated Resource Utilization section. • Updated System Reset section. • Updated Example Design and Test Bench chapters.
12/18/2013	5.0	Added UltraScale support.
10/02/2013	5.0	• Revision number advanced to 5.0 to align with core version number 5.0. • Configurable MTBF Flip-flops to improve MTBF.

Date	Version	Revision
03/20/2013	1.1	Revised for Vivado-only release: • Update to PG template v3.4. • Remove references to ISE and ChipScope Pro. Additional content changes: • Revised IP Facts introduction paragraph. • Added Introduction paragraph to Chapter 1. • Revised Register Space table. • Reversed bit order in register descriptions. • Revised output directory and file content description. • Replaced screen image with Vivado IP catalog version. • Revised Performance and Resource Utilization tables. • Replaced AXI port descriptions with reference to UG761. • Removed ISE-specific design parameter tables from Chapter 3 and Appendix B. • Added Hardware Debug section to Appendix C. • Removed redundant IDE section in Chapter 6.
12/18/2012	1.0	Initial release of document in product guide format. This product guide replaces DS798 and UG765. The following items indicate new information that those two documents did not have. • Updated licensing and ordering information in Chapter 2. • Added resource numbers and maximum frequencies in Chapter 3, Product Specification. • Updated all screen captures. • Added false path constraints and clock frequencies. • Added XCO file parameter values to Chapter 7: Customizing and Generating the Core. • Added compliance testing information to Appendix A. • Added Appendix C, Debugging.

Please Read: Important Legal Notices