

LogiCORE IP CAN v4.2

Product Guide

PG096 December 18, 2012

Table of Contents

SECTION I: SUMMARY

IP Facts

Chapter 1: Overview

Feature Summary	8
Operating System Requirements	11
Recommended Design Experience	12
Licensing and Ordering Information	12

Chapter 2: Product Specification

Operational CAN Controller Modes	13
Standards	16
Performance	16
Resource Utilization	16
Port Descriptions	26
Xilinx CAN Controller Configuration Register Descriptions	28

Chapter 3: Designing with the Core

Configuring the CAN Controller	54
Clocking	58
Resets	58
Interrupts	59
Xilinx CAN Controller Design Parameters	60

SECTION II: VIVADO DESIGN SUITE

Chapter 4: Customizing and Generating the Core

CAN Graphical User Interface	62
Parameter Values in the XCI File.....	64
Output Generation.....	64

Chapter 5: Constraining the Core

Required Constraints	69
Clock Frequencies	69

Chapter 6: Detailed Example Design

Generating the Core.....	71
Implementing the Example Design.....	73
Simulating the Example Design.....	74
Directory and File Contents.....	75
Example Design Configuration	76
Demonstration Test Bench	77
Implementation	80

SECTION III: ISE DESIGN SUITE

Chapter 7: Customizing and Generating the Core

GUI	82
Parameter Values in the XCO File.....	83
Output Generation.....	84

Chapter 8: Constraining the Core

Device and Package Selection.....	89
Location Constraints	89
Placement Constraints.....	89
Timing Constraints	90
I/O Constraints.....	91
I/O Standards	91

Chapter 9: Detailed Example Design

Generating the Core.	93
Implementing the Example Design.	95
Simulating the Example Design.	95
Directory and File Contents.	96
Implementation Scripts.	97
Example Design Configuration	99
Demonstration Test Bench	100

SECTION IV: APPENDICES

Appendix A: Verification, Compliance, and Interoperability

Compliance Testing	105
------------------------------	-----

Appendix B: Migrating

Parameter Changes in the XCO File	106
Parameter Values in the XCI File.	106

Appendix C: Debugging

Finding Help on Xilinx.com	107
Debug Tools	109
Simulation Debug.	111
Interface Debug	112

Appendix D: Additional Resources

Xilinx Resources	114
References	114
Technical Support	114
Revision History	115
Notice of Disclaimer.	115

SECTION I: SUMMARY

IP Facts

Overview

Product Specification

Designing with the Core

Introduction

The Xilinx LogiCORE™ IP Controller Area Network (CAN) product specification describes the architecture and features of the Xilinx CAN controller 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 described in the [References](#) section is assumed.

Features

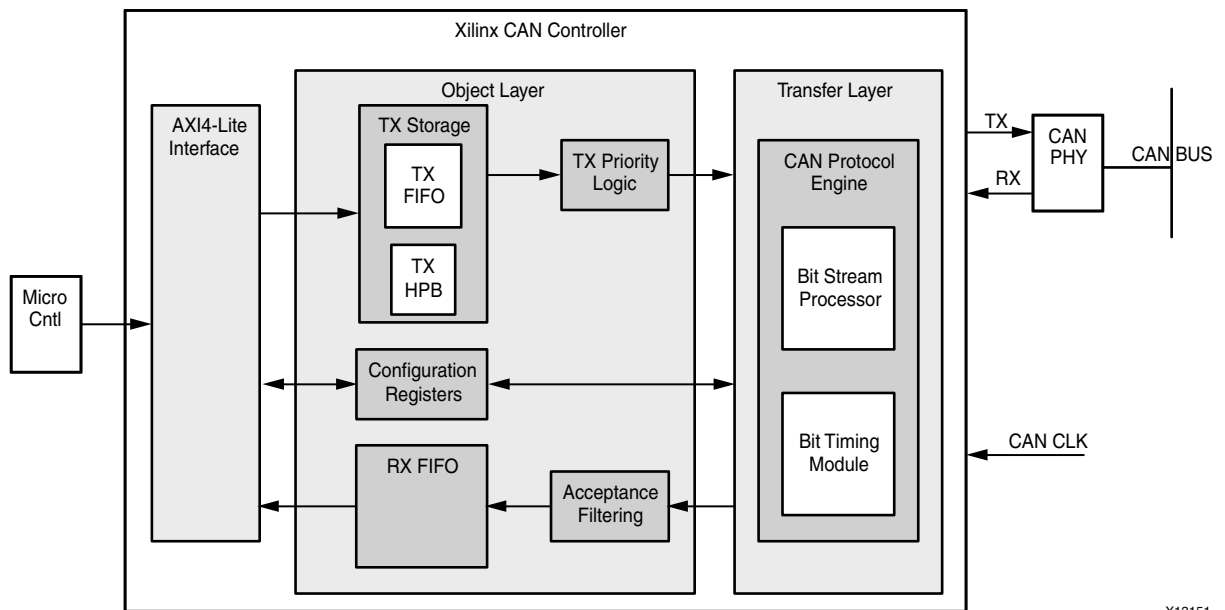
- Industrial (I, -40 to +100°C junction temperature) and Automotive/Defense Q-Grade (Q, -40 to +125°C junction temperature) device support.
- Supports both standard (11-bit identifier) and extended (29-bit identifier) frames
- Supports bit rates up to 1 Mb/s
- Transmit message First In First Out (FIFO) 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 4 acceptance filters
- Sleep Mode with automatic wake up
- Loop Back Mode for diagnostic applications
- Maskable Error and Status Interrupts
- Readable Error Counters

LogiCORE IP Facts Table					
Core Specifics					
Supported Device Family ⁽¹⁾	Zynq™-7000 ⁽²⁾ Virtex®-7 Kintex™-7 Artix™-7 XC Virtex-6 XA/XC Spartan®-6				
Supported User Interfaces	AXI4-Lite				
Resources ⁽³⁾	LUTs	FFs	Slices	Block RAMs	I/O⁽⁴⁾
	811 – 1006	553 – 743	272 – 415	2	3
Provided with Core					
Design Files	ISE®: VHDL Vivado™: Encrypted RTL				
Example Design	Verilog and VHDL				
Test Bench	VHDL, Verilog				
Constraints File	ISE: Xilinx Constraints File Vivado: Xilinx Design Constraints (XDC)				
Simulation Model	Verilog				
Supported S/W Driver	Standalone ⁽⁵⁾				
Tested Design Flows ⁽⁶⁾					
Design Entry	ISE Design Suite v14.4 Vivado Design Suite ⁽⁷⁾ v2012.4				
Simulation	Mentor Graphics ModelSim Cadence Incisive Enterprise Simulator (IES)				
Synthesis	XST Vivado Synthesis				
Support					
Provided by Xilinx @ www.xilinx.com/support					

1. For the complete list of supported devices, see the release notes for this core [release notes](#).
2. Supported in ISE Design Suite implementations only.
3. Resources numbers for Zynq-7000 devices are expected to be similar to 7 series device numbers.
4. For External I/O only.
5. Standalone driver details can be found in the EDK or SDK directory (<install_directory>/doc/usenglish/xilinx_drivers.htm).
6. For the supported versions of the tools, see the [Xilinx Design Tools: Release Notes Guide](#).
7. Supports only 7 series devices.

Overview

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



X13151

Figure 1-1: CAN Core Block Diagram

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

Feature Summary

This section contains the following subsections.

- [Configuration Registers](#)
- [Transmit and Receive Messages](#)
- [TX High Priority Buffer](#)
- [Acceptance Filters](#)
- [CAN Protocol Engine](#)
- [Bit Timing Logic](#)
- [Bit Stream Processor](#)

Configuration Registers

[Table 2-8](#) defines the configuration registers. This module allows for read and write access to the registers through the external micro-controller interface.

Transmit and Receive Messages

Separate storage buffers exist for transmit (TX FIFO) and receive (RX FIFO) 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 Transfer 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.

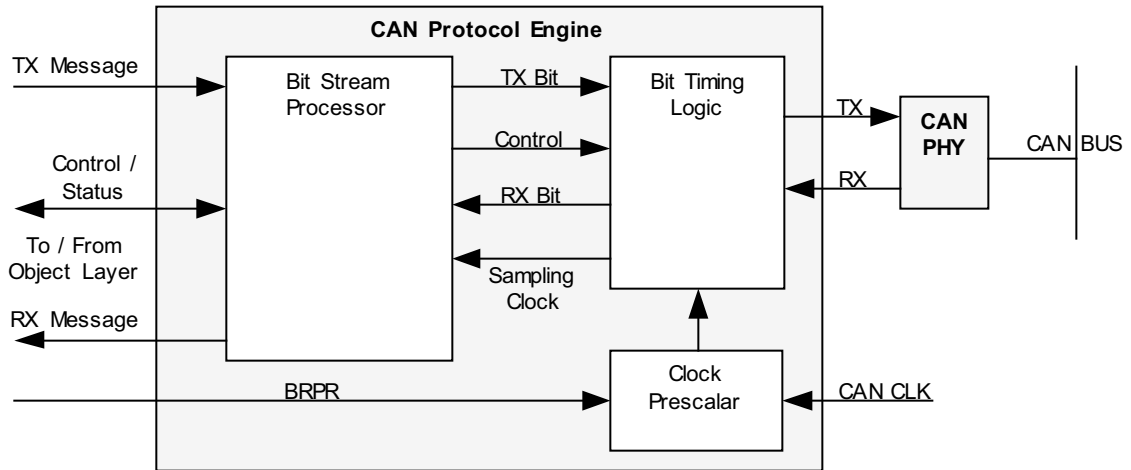


Figure 1-2: CAN Protocol Engine

Bit Timing Logic

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

- Synchronizing the CAN controller 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

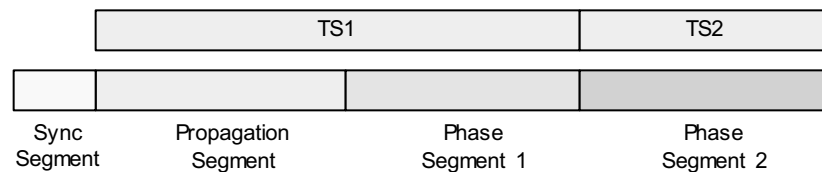


Figure 1-3: CAN Bit Timing

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 tq's) 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 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 analyses 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 controller 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 controller: 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 controller is updated by the BSP.

Operating System Requirements

For a list of operating system requirements, For the supported versions of the tools, see the [Xilinx Design Tools: Release Notes Guide](#).

Recommended Design Experience

Although the CAN core is a fully-verified targeted design platform, the challenge associated with implementing a complete CAN design varies, depending on the application requirements.



RECOMMENDED: For best results, previous experience with building high-performance FPGA designs using Xilinx implementation software and a user constraints file (UCF) is recommended.

Contact your local Xilinx representative for assistance in evaluating your specific requirements.

Licensing and Ordering Information

This Xilinx LogiCORE™ IP module is provided under the terms of the CAN LogiCORE IP License Agreement for [Automotive](#) or [Non-Automotive](#) applications. The module is shipped as part of the Vivado™ Design Suite/ISE Design Suite. For full access to all core functionalities in simulation and in hardware, you must purchase a license for the core. [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 and Xilinx Platform Studio (XPS) CAN cores require 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 and XPS CAN cores.

Product Specification

Operational CAN Controller Modes

The CAN controller supports these modes of operation:

- Configuration
- Normal
- Sleep
- Loop Back

Table 2-1 defines the CAN Controller modes of operation and corresponding control and status bits. Inputs that affect the mode transitions are defined in [Xilinx CAN Controller Configuration Register Descriptions](#).

Table 2-1: CAN Controller 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'	Loop Back 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 controller 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 '1' on the S_AXI_ARESETN input. The core continues to be in reset as long as S_AXI_ARESETN is '1'. The core enters Configuration mode after S_AXI_ARESETN is negated to '0'.

Following are the Configuration Mode features

- CAN controller 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 controller does not receive any new messages.
- CAN controller does not transmit any messages. Messages in the TX FIFO and the TX high priority buffer are pended. 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 TXBFL and TXFL, and the Status Register bits TXBFL and TXFL, 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 controller 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 controller waits for a sequence of 11 recessive bits before exiting Configuration mode.

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

Normal Mode

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

For Normal mode, the CAN controller 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 controller 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 controller enters Sleep mode only when there are no pending transmission requests from either the TX FIFO or the TX High Priority Buffer.

The CAN controller 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 controller 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 controller exits Sleep mode. Interrupts are generated when the CAN controller enters Sleep mode or wakes up from Sleep mode. From sleep mode, the CAN controller can enter either the Configuration or Normal modes.

The CAN controller 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

Loop Back Mode

In Loop Back mode, the CAN controller transmits a recessive bitstream on to the CAN Bus. Any message transmitted is looped back to the RX line and is acknowledged. The CAN controller 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 Loop Back mode, the CAN controller can only enter Configuration mode. The CAN controller enters Configuration mode when any of the configuration conditions are satisfied.

The CAN controller enters Loop Back mode from the Configuration mode if the LBACK bit in MSR is '1' and the CEN bit in SRR is '1.'

Standards

CAN 4.2 core conforms to the ISO 11898 -1, CAN 2.0A, and CAN 2.0B standards

Performance

Maximum Frequencies

The range of CAN_CLK clock is 8-24 MHZ.

Resource Utilization

Resources required for the CAN v4.2 core have been estimated for the following devices.

- Virtex®-7 FPGA (xc7v285t-1-ffg1157) [Table 2-2](#)
- Kintex™-7 FPGA (xc7k410t-1-ffg900) [Table 2-3](#)
- Artix™-7 FPGA (xc7a355tdie-3) [Table 2-4](#)
- Spartan®-6 FPGA (xc6slx45t-2-ffg484) [Table 2-5](#)
- Virtex-6 FPGA (xc6vlx130t-2-ff484) [Table 2-6](#)

These values were generated using ISE® Design Suite 14.4 and Vivado™ Design Suite 2012.4. The results are post PAR results. For Zynq™-7000, estimated resources depend upon the fabric used.

Table 2-2: Performance and Resource Utilization Benchmarks for Virtex-7 FPGA (xc7v285t-1-ffg1157)

Parameter Values			Device Resources			FMAX (MHz)
C_CAN_RX_DPTH	C_CAN_TX_DPTH	C_CAN_NUM_ACF	Slices	Slice Flip- Flops	LUTs	AXI FMAX
2	2	0	326	557	757	142
2	2	1	304	650	844	145
2	2	2	280	653	852	137
2	2	3	309	656	840	141
2	2	4	317	659	847	142
4	4	0	301	565	773	141
4	4	1	291	666	856	134
4	4	2	342	669	864	128
4	4	3	353	672	866	139
4	4	4	313	675	869	134
8	8	0	319	589	796	136
8	8	1	337	682	870	129
8	8	2	338	685	869	129
8	8	3	340	688	870	130
8	8	4	341	691	867	137
16	16	0	316	605	813	132
16	16	1	334	698	884	134
16	16	2	383	701	886	134
16	16	3	347	680	872	131
16	16	4	350	707	901	124

Table 2-2: Performance and Resource Utilization Benchmarks for Virtex-7 FPGA (xc7v285t-1-ffg1157) (Cont'd)

Parameter Values			Device Resources			FMAX (MHz)
C_CAN_RX_DPTH	C_CAN_TX_DPTH	C_CAN_NUM_ACF	Slices	Slice Flip- Flops	LUTs	AXI FMAX
32	32	0	333	621	819	137
32	32	1	364	714	888	136
32	32	2	324	717	895	149
32	32	3	309	720	911	149
32	32	4	355	723	906	138
64	64	0	342	637	831	132
64	64	1	363	730	934	132
64	64	2	347	733	927	136
64	64	3	348	736	927	133
64	64	4	314	739	936	134

Table 2-3: Performance and Resource Utilization Benchmarks for Kintex-7 FPGA (xc7k410t-1-ffg900)

Parameter Values			Device Resources			FMAX (MHz)
C_CAN_RX_DPTH	C_CAN_TX_DPTH	C_CAN_NUM_ACF	Slices	Slice Flip-Flops	LUTs	AXI FMAX
2	2	0	339	557	758	148
2	2	1	372	650	844	145
2	2	2	362	653	851	144
2	2	3	356	656	849	134
2	2	4	369	659	854	146
4	4	0	341	565	767	151
4	4	1	338	666	861	122
4	4	2	366	669	858	144
4	4	3	359	672	867	141
4	4	4	355	675	862	125
8	8	0	330	589	788	106
8	8	1	345	682	861	140
8	8	2	355	685	874	149
8	8	3	352	688	870	142
8	8	4	342	691	879	154
16	16	0	339	605	803	134
16	16	1	350	698	886	142
16	16	2	349	701	899	142
16	16	3	341	680	868	139
16	16	4	370	707	894	143

Table 2-3: Performance and Resource Utilization Benchmarks for Kintex-7 FPGA (xc7k410t-1-ffg900) (Cont'd)

Parameter Values			Device Resources			FMAX (MHz)
C_CAN_RX_DPTH	C_CAN_TX_DPTH	C_CAN_NUM_ACF	Slices	Slice Flip- Flops	LUTs	AXI FMAX
32	32	0	337	621	822	148
32	32	1	389	714	890	146
32	32	2	375	717	905	133
32	32	3	378	720	898	136
32	32	4	373	723	903	137
64	64	0	345	637	825	140
64	64	1	388	730	920	148
64	64	2	373	733	929	140
64	64	3	384	736	925	137
64	64	4	365	739	927	125

Table 2-4: Performance and Resource Utilization Benchmarks for Artix-7 FPGA (xc7a355tdie-3)

Parameter Values			Device Resources			FMAX (MHz)
C_CAN_RX_DPTH	C_CAN_TX_DPTH	C_CAN_NUM_ACF	Slices	Slice Flip- Flops	LUTs	AXI FMAX
2	2	0	314	552	754	151
2	2	1	353	644	842	157
2	2	2	344	648	846	141
2	2	3	350	651	861	160
2	2	4	326	653	850	115
4	4	0	277	560	744	131
4	4	1	355	660	844	136
4	4	2	337	664	866	135
4	4	3	333	667	852	123
4	4	4	329	669	859	145
8	8	0	305	584	760	131
8	8	1	339	677	868	152
8	8	2	338	680	868	130
8	8	3	349	683	879	139
8	8	4	337	685	878	133
16	16	0	308	600	800	102
16	16	1	343	693	874	130
16	16	2	335	696	879	140
16	16	3	330	675	876	134
16	16	4	347	701	895	139

Table 2-4: Performance and Resource Utilization Benchmarks for Artix-7 FPGA (xc7a355tdie-3)

Parameter Values			Device Resources			FMAX (MHz)
C_CAN_RX_DPTH	C_CAN_TX_DPTH	C_CAN_NUM_ACF	Slices	Slice Flip-Flops	LUTs	AXI FMAX
32	32	0	299	6161	799	114
32	32	1	348	709	893	115
32	32	2	364	712	901	117
32	32	3	323	715	902	133
32	32	4	364	717	889	123
64	64	0	341	632	829	133
64	64	1	342	727	931	134
64	64	2	355	730	922	141
64	64	3	359	733	914	132
64	64	4	359	735	923	143

**Table 2-5: Performance and Resource Utilization Benchmarks for the Spartan-6
FPGA (xc6slx45t-2-fgg484)**

Parameter values			Device resources			F _{MAX} (MHz)
C_CAN_RX_DPTH	C_CAN_TX_DPTH	C_CAN_NUM_ACF	Slices	Slice Flip-Flops	LUTs	AXI F _{MAX}
2	2	0	374	553	794	75
2	2	1	405	648	866	68
2	2	2	311	651	896	80
2	2	3	394	654	878	79
2	2	4	386	648	864	72
4	4	0	378	561	805	62
4	4	1	398	664	893	76
4	4	2	385	667	897	73
4	4	3	389	670	891	75
4	4	4	393	664	886	74
8	8	0	368	585	826	71
8	8	1	403	680	895	75
8	8	2	384	683	909	71
8	8	3	392	686	912	83
8	8	4	365	680	902	77
16	16	0	391	601	848	70
16	16	1	402	696	919	84
16	16	2	411	699	925	75
16	16	3	401	678	905	73
16	16	4	430	696	905	70
32	32	0	371	617	845	66
32	32	1	419	712	926	75
32	32	2	430	715	924	75
32	32	3	401	718	935	81
32	32	4	372	712	911	84
64	64	0	364	633	889	76
64	64	1	428	728	962	74

Table 2-5: Performance and Resource Utilization Benchmarks for the Spartan-6 FPGA (Cont'd)(xc6slx45t-2-fgg484) (Cont'd)

Parameter values			Device resources			F _{MAX} (MHz)
C_CAN_RX_DPTH	C_CAN_TX_DPTH	C_CAN_NUM_ACF	Slices	Slice Flip-Flops	LUTs	AXI F _{MAX}
64	64	2	404	731	970	60
64	64	3	428	734	955	73
64	64	4	399	728	948	75

Table 2-6: Performance and Resource Utilization Benchmarks for Virtex-6 FPGA (xc6vlx130t-2-ff484)

Parameter values			Device resources			F _{MAX} (MHz)
C_CAN_RX_DPTH	C_CAN_TX_DPTH	C_CAN_NUM_ACF	Slices	Slice Flip-Flops	LUTs	AXI F _{MAX}
2	2	0	263	555	799	200
2	2	1	287	642	874	181
2	2	2	314	651	879	206
2	2	3	305	654	875	195
2	2	4	311	652	864	172
4	4	0	284	563	805	174
4	4	1	284	658	896	184
4	4	2	312	667	895	187
4	4	3	280	670	893	194
4	4	4	310	668	877	169
8	8	0	290	587	822	187
8	8	1	269	674	901	196
8	8	2	328	683	916	184
8	8	3	299	686	905	197
8	8	4	271	684	897	171

Table 2-6: Performance and Resource Utilization Benchmarks for Virtex-6 FPGA (xc6vlx130t-2-ff484) (Cont'd)

Parameter values			Device resources			F _{MAX} (MHz)
C_CAN_RX_DPTH	C_CAN_TX_DPTH	C_CAN_NUM_ACF	Slices	Slice Flip-Flops	LUTs	AXI F _{MAX}
16	16	0	284	602	843	201
16	16	1	294	690	922	184
16	16	2	327	699	925	179
16	16	3	295	678	899	197
16	16	4	280	700	918	176
32	32	0	271	618	860	194
32	32	1	257	706	906	165
32	32	2	296	715	920	190
32	32	3	311	718	924	195
32	32	4	325	717	914	167
64	64	0	269	634	884	201
64	64	1	305	722	949	174
64	64	2	293	731	942	192
64	64	3	309	734	944	192
64	64	4	299	732	934	183

Port Descriptions

The external interface of the CAN controller is a AXI4-Lite Interface.

Table 2-7 defines CAN controller interface signalling.

Table 2-7: CAN Controller External I/Os

No.	Signal Name	I/O	Default Value	Description
1	S_AXI_ACLK	I	N/A	Advanced eXtensible Interface (AXI) Clock
2	S_AXI_ARESETN	I	N/A	AXI Reset (active-Low)
3	S_AXI_AWADDR[C_S_AXI_ADDR_WIDTH-1:0]	I	N/A	AXI Write address. The write address bus gives the address of the write transaction.
4	S_AXI_AWVALID	I	N/A	Write address valid. This signal indicates that valid write address and control information are available.
5	S_AXI_AWREADY	O	N/A	Write address ready. This signal indicates that the slave is ready to accept an address and associated control signals.
6	S_AXI_WDATA[C_S_AXI_DATA_WIDTH-1:0]	I	N/A	Write Data
7	S_AXI_WSTB[C_S_AXI_DATA_WIDTH/8-1:0]	I	N/A	Write strobes. This signal indicates which byte lanes to update in memory.
8	S_AXI_WVALID	I	N/A	Write valid. This signal indicates that valid write data and strobes are available.
9	S_AXI_WREADY	O	0x0	Write ready. This signal indicates that the slave can accept the write data.
10	S_AXI_BRESP[1:0]	O	0x0	Write response. This signal indicates the status of the write transaction. "00"- OKAY "10"- SLVERR "11"- DECERR
11	S_AXI_BVALID	O	0x0	Write response valid. This signal indicates that a valid write response is available
12	S_AXI_BREADY	I	0x1	Response ready. This signal indicates that the master can accept the response information
13	S_AXI_ARADDR[C_S_AXI_ADDR_WIDTH-1:0]	I	N/A	Read address. The read address bus gives the address of a read transaction.

Table 2-7: CAN Controller External I/Os (Cont'd)

No.	Signal Name	I/O	Default Value	Description
14	S_AXI_ARVALID	I	N/A	Read address valid. This signal indicates, when HIGH, that the read address and control information is valid and remains stable until the address acknowledgement signal, S_AXI_ARREADY, is high.
15	S_AXI_ARREADY	O	0x1	Read address ready. This signal indicates that the slave is ready to accept an address and associated control signals.
16	S_AXI_RDATA[C_S_AXI_DATA_WIDTH-1:0]	O	0x0	Read data
17	S_AXI_RRESP[1:0]	O	0x0	Read response. This signal indicates the status of the read transfer. "00"- OKAY "10"- SLVERR "11"- DECERR
18	S_AXI_RVALID	O	0x0	Read valid. This signal indicates that the required read data is available and the read transfer can complete
19	S_AXI_RREADY	I	0x1	Read ready. This signal indicates that the master can accept the read data and response information.
20	IP2Bus_IntrEvent	O	0x0	Active-High Interrupt line. ^a
21	CAN_CLK	I		24 MHz oscillator clock input.
22	CAN_PHY_TX	O	1	CAN bus transmit signal to PHY.
23	CAN_PHY_RX	I	N/A	CAN bus receive signal from PHY.

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

Xilinx CAN Controller Configuration Register Descriptions

Table 2-8 defines the CAN controller configuration registers. Each of these registers is 32-bits wide and is represented in big endian format. Because the controller supports 32-bit word access, the `S_AXI_AWADDR/S_AXI_ARADDR` is appended with `2'b00` internally. Any read operations to reserved bits or bits that are not used return '0.' A '0' should be written to reserved bits and bit fields not used. Writes to reserved locations are ignored.

Table 2-8: CAN Controller Configuration Register

Register Name		Address	Access
Control Registers			
Software Reset Register (SRR)		0x000	Read/Write
Mode Select Register (MSR)		0x004	Read/Write
Transfer Layer Configuration Registers			
Baud Rate Prescaler Register (BRPR)		0x008	Read/Write
Bit Timing Register (BTR)		0x00C	Read/Write
Error Indication Registers			
Error Counter Register (ECR)		0x010	Read
Error Status Register (ESR)		0x014	Read/Write to Clear
CAN Status Registers			
Status Register (SR)		0x018	Read
Interrupt Registers			
Interrupt Status Register (ISR)		0x01C	Read
Interrupt Enable Register (IER)		0x020	Read/Write
Interrupt Clear Register (ICR)		0x024	Write
Reserved			
Reserved Locations		0x028 to 0x02C	Reads Return 0/ Write has no affect
Messages			
Transmit Message FIFO (TX FIFO)			
	ID	0x030	Write
	DLC	0x034	Write
	Data Word 1	0x038	Write
	Data Word 2	0x03C	Write
Transmit High Priority Buffer (TX HPB)			

Table 2-8: CAN Controller Configuration Register (Cont'd)

Register Name		Address	Access
	ID	0x040	Write
	DLC	0x044	Write
	Data Word 1	0x048	Write
	Data Word 2	0x04C	Write
Receive Message FIFO (RX FIFO)			
	ID	0x050	Read
	DLC	0x054	Read
	Data Word 1	0x058	Read
	Data Word 2	0x05C	Read
Acceptance Filtering			
Acceptance Filter Register (AFR)		0x060	Read/Write
Acceptance Filter Mask Register 1 (AFMR1)		0x064	Read/Write
Acceptance Filter ID Register 1 (AFIR1)		0x068	Read/Write
Acceptance Filter Mask Register 2(AFMR2)		0x06C	Read/Write
Acceptance Filter ID Register 2 (AFIR2)		0x070	Read/Write
Acceptance Filter Mask Register 3(AFMR3)		0x074	Read/Write
Acceptance Filter ID Register 3 (AFIR3)		0x078	Read/Write
Acceptance Filter Mask Register 4(AFMR4)		0x07C	Read/Write
Acceptance Filter ID Register 4 (AFIR4)		0x080	Read/Write
Reserved			
Reserved Locations		0x084 to 0x0FC	Reads Return 0/ Write has no affect

Control Registers

Software Reset Register (0x000)

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

Use these steps to configure the CAN controller 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 \(0x008\)](#) and [Bit Timing Register \(0x00C\)](#).

2. Do one of the following:
 - For Loop Back mode, write '1' to LBACK bit in the MSR.
 - For Sleep mode, write '1' to the SLEEP bit in the MSR.
See [Table 2-7 defines CAN controller interface signalling](#). for information about operational modes.
3. Set the set the CEN bit in the SRR to 1.

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

[Table 2-9](#) defines the bit positions in the Software Reset Register (SRR) and [Table 2-10](#) defines the Software Reset Register bits.

Table 2-9: Software Reset Register BIT Positions

0 – 29	30	31
Reserved	CEN	SRST

Table 2-10: Software Reset Register Bits

Bit(s)	Name	Access	Default Value	Description
0–29	Reserved	Read/Write	0	Reserved Reserved for future expansion.
30	CEN	Read/Write	0	Can Enable The Enable bit for the CAN controller. '1' = The CAN controller is in Loop Back, Sleep or Normal mode depending on the LBACK and SLEEP bits in the MSR. '0' = The CAN controller is in the Configuration mode.
31	SRST	Read/Write	0	Reset The Software reset bit for the CAN controller. '1' = CAN controller is reset. If a '1' is written to this bit, all the CAN controller configuration registers (including the SRR) are reset. Reads to this bit always return a '0.'

Mode Select Register (0x004)

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

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

Table 2-11 shows the bit positions in the MSR and Table 2-12 describes the MSR bits.

Table 2-11: Mode Select Register Bit Positions

0–29	30	31
Reserved	LBACK	SLEEP

Table 2-12: Mode Select Register Bits

Bit(s)	Name	Access	Default Value	Description
0–29	Reserved	Read/Write	0	Reserved Reserved for future expansion.
30	LBACK	Read/Write	0	Loop Back Mode Select The Loop Back Mode Select bit. '1' = CAN controller is in Loop Back mode. '0' = CAN controller is in Normal, Configuration, or Sleep mode. This bit can be written to only when CEN bit in SRR is '0.'
31	SLEEP	Read/Write	0	Sleep Mode Select The Sleep Mode select bit. '1' = CAN controller is in Sleep mode. '0' = CAN controller is in Normal, Configuration or Loop Back mode. This bit is cleared when the CAN controller 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 (0x008)

The CAN clock for the CAN controller is divided by (prescaler+1) to generate the quantum clock needed for sampling and synchronization. Table 2-13 shows the bit positions in the BRPR, and Table 2-14 defines the BRPR bits.

Table 2-13: Baud Rate Prescaler Register Positions

0 – 23	24 – 31
Reserved	BPR [7.0]

Table 2-14: Baud Rate Prescaler Register Bits

Bit(s)	Name	Access	Default Value	Description
0–23	Reserved	Read/Write	0	Reserved Reserved for future expansion.
24–31	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 1 more than the value written into the register.

The CAN quantum clock can be calculated using this equation:

$$tq = tosc * (BRP + 1)$$

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

Note: 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* specifications.

Bit Timing Register (0x00C)

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 2-15](#) shows the bit positions in the BTR and [Table 2-16](#) defines the BTR bits.

Table 2-15: Bit Timing Register BIT Positions

0–22	23–24	25–27	28–31
Reserved	SJW[1..0]	TS2[2..0]	TS1[3..0]

Table 2-16: Bit Timing Register Bits

Bit(s)	Name	Access	Default Value	Description
0–22	Reserved	Read/Write	0	Reserved Reserved for future expansion.
23–24	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.
25–27	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.
28–31	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:

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

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

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

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

Note: 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* specifications.

Error Indication Registers

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

Error Counter Register (0x010)

The 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 controller enters Bus Off state
- During Bus Off recovery when the CAN controller enters Error Active state after 128 occurrences of 11 consecutive recessive bits

When in Bus Off recovery, the Receive Error counter is advanced by 1 when a sequence of 11 consecutive recessive bits is seen.

Table 2-17 shows the bit positions in the ECR and Table 2-18 defines the ECR bits.

Table 2-17: Error Count Register BIT Positions

0 –15	16 – 23	24 – 31
Reserved	REC[7..0]	TEC[7..0]

Table 2-18: Error Count Register Bits

Bit(s)	Name	Access	Default Value	Description
0–15	Reserved	Read Only	0	Reserved Reserved for future expansion.
16–23	REC[7..0]	Read Only	0	Receive Error Counter Indicates the Value of the Receive Error Counter.
24–31	TEC[7..0]	Read Only	0	Transmit Error Counter Indicates the Value of the Transmit Error Counter.

Error Status Register (0x014)

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 2-19 shows the bit positions in the ESR and Table 2-20 describes the ESR bits. All the bits in the ESR are cleared when '0' is written to the CEN bit in the SRR.

Table 2-19: Error Status Register BIT Positions

0 — 26	27	28	29	30	31
Reserved	ACKER	BERR	STER	FMER	CRCER

Table 2-20: Error Status Register Bits

Bit(s)	Name	Access	Default Value	Description
0—26	Reserved	Read/Write	0	Reserved Reserved for future expansion.
27	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.
28	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.
29	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.
30	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.
31	CRCER	Write to Clear	0	CRC Error ^a 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.

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

CAN Status Register (0x018)

The CAN Status Register provides a status of all conditions of the core. Specifically, FIFO status, Error State, Bus State and Configuration mode are reported.

Table 2-21 shows the SR bit positions in the SR and Table 2-22 provides SR bit descriptions.

Table 2-21: Status Register BIT Positions

0 — 19	20	21	22	23 — 24	25
Reserved	ACFBSY	TXFLL	TXBFLL	ESTAT[1..0]	ERRWRN
26	27	28	29	30	31
BBSY	BIDLE	NORMAL	SLEEP	LBACK	CONFIG

Table 2-22: Status Register Bits

Bit(s)	Name	Access	Default Value	Description
0—19	Reserved	Read/Write	0	Reserved Reserved for future expansion.
20	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.
21	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.
22	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.
23–24	ESTAT[1..0]	Read Only	0	Error Status Indicates the error status of the CAN controller. "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.

Table 2-22: Status Register Bits (Cont'd)

Bit(s)	Name	Access	Default Value	Description
25	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 greater than or equal to 96. '0' = Neither of the error counters has a value greater than or equal to 96.
26	BBSY	Read Only	0	Bus Busy Indicates the CAN bus status. '1' = Indicates that the CAN controller is either receiving a message or transmitting a message. '0' = Indicates that the CAN controller is either in Configuration mode or the bus is idle.
27	BIDLE	Read Only	0	Bus Idle Indicates the CAN bus status. '1' = Indicates no bus communication is taking place. '0' = Indicates the CAN controller is either in Configuration mode or the bus is busy.
28	NORMAL	Read Only	0	Normal Mode Indicates the CAN controller is in Normal Mode. '1' = Indicates the CAN controller is in Normal Mode. '0' = Indicates the CAN controller is not in Normal mode.
29	SLEEP	Read Only	0	Sleep Mode Indicates the CAN controller is in Sleep mode. '1' = Indicates the CAN controller is in Sleep mode. '0' = Indicates the CAN controller is not in Sleep mode.
30	LBACK	Read Only	0	Loop Back Mode Indicates the CAN controller is in Loop Back mode. '1' = Indicates the CAN controller is in Loop Back mode. '0' = Indicates the CAN controller is not in Loop Back mode.
31	CONFIG	Read Only	1	Configuration Mode Indicator Indicates the CAN controller is in Configuration mode. '1' = Indicates the CAN controller is in Configuration mode. '0' = Indicates the CAN controller is not in Configuration mode.

Interrupt Registers

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

Interrupt Status Register (0x01C)

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 2-23 shows the bit positions in the ISR and Table 2-24 describes the ISR bits.

Table 2-23: Interrupt Status Register BIT Positions

0 — 19	20	21	22	23	24	25
Reserved	WKUP	SLP	BSOFF	ERROR	RXNEMP	RXOFLW
26	27	28	29	30	31	
RXUFLW	RXOK	TXBFL	TXFLL	TXOK	ARBLST	

Table 2-24: Interrupt Status Register Bits

Bit(s)	Name	Access	Default Value	Description
0–19	Reserved	Read/Write	0	Reserved Reserved for future expansion.
20	WKUP	Read Only	0	Wake up Interrupt A '1' indicates that the CAN controller 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.
21	SLP	Read Only	0	Sleep Interrupt A '1' indicates that the CAN controller 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.
22	BSOFF	Read Only	0	Bus Off Interrupt A '1' indicates that the CAN controller 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 2-24: Interrupt Status Register Bits (Cont'd)

Bit(s)	Name	Access	Default Value	Description
23	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.
24	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.
25	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.
26	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.
27	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.
28	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.
29	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.

Table 2-24: Interrupt Status Register Bits (Cont'd)

Bit(s)	Name	Access	Default Value	Description
30	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.
31	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.

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

Interrupt Enable Register (0x020)

The Interrupt Enable Register (IER) is used to enable interrupt generation. Table 2-25 shows the bit positions in the IER and Table 2-26 describes the IER bits.

Table 2-25: Interrupt Enable Register Bit Positions

0 – 19	20	21	22	23	24	25
Reserved	EWKUP	ESLP	EBSOFF	EERROR	ERXNEMP	ERXOFLW
26	27	28	29	30	31	
ERXUFLW	ERXOK	ETXBFL	ETXFLL	ETXOK	EARBLST	

Table 2-26: Interrupt Enable Register Bits

Bit(s)	Name	Access	Default Value	Description
0–19	Reserved	Read/Write	0	Reserved Reserved for future expansion.
20	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.
21	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.

Table 2-26: Interrupt Enable Register Bits (Cont'd)

Bit(s)	Name	Access	Default Value	Description
22	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.
23	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.
24	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.
25	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.
26	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.
27	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.
28	ETXBFL	Read/Write	0	Enable High Priority Transmit Buffer Full Interrupt Writes to this bit enable or disable interrupts when the TXBFL bit in the ISR is set. '1' = Enable interrupt generation if TXBFL bit in ISR is set. '0' = Disable interrupt generation if TXBFL bit in ISR is set.
29	ETXFLL	Read/Write	0	Enable Transmit FIFO Full Interrupt Writes to this bit enable or disable interrupts when the 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 2-26: Interrupt Enable Register Bits (Cont'd)

Bit(s)	Name	Access	Default Value	Description
30	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.
31	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 (0x024)

The Interrupt Clear Register (ICR) is used to clear interrupt status bits. Table 2-27 shows the bit positions in the ICR and Table 2-28 describes the ICR bits.

Table 2-27: Interrupt Clear Register Bit Positions

0 – 19	20	21	22	23	24	25
Reserved	CWKUP	CSLP	CBSOFF	CERROR	CRXNEMP	CRXOFLW
26	27	28	29	30	31	
CRXUFLW	CRXOK	CTXBLL	CTXFLL	CTXOK	CARBLST	

Table 2-28: Interrupt Clear Register Bit Descriptions

Bit(s)	Name	Access	Default Value	Description
0–19	Reserved	Read/Write	0	Reserved Reserved for future expansion.
20	CWKUP	Write Only	0	Clear Wake up Interrupt Writing a '1' to this bit clears the WKUP bit in the ISR.
21	CSLP	Write Only	0	Clear Sleep Interrupt Writing a '1' to this bit clears the SLP bit in the ISR.
22	CBSOFF	Write Only	0	Clear Bus Off Interrupt Writing a '1' to this bit clears the BSOFF bit in the ISR.
23	CERROR	Write Only	0	Clear Error Interrupt Writing a '1' to this bit clears the ERROR bit in the ISR.

Table 2-28: Interrupt Clear Register Bit Descriptions (Cont'd)

Bit(s)	Name	Access	Default Value	Description
24	CRXNEMP	Write Only	0	Clear Receive FIFO Not Empty Interrupt Writing a '1' to this bit clears the RXNEMP bit in the ISR.
25	CRXOFLW	Write Only	0	Clear RX FIFO Overflow Interrupt Writing a '1' to this bit clears the RXOFLW bit in the ISR.
26	CRXUFLW	Write Only	0	Clear RX FIFO Underflow Interrupt Writing a '1' to this bit clears the RXUFLW bit in the ISR.
27	CRXOK	Write Only	0	Clear New Message Received Interrupt Writing a '1' to this bit clears the RXOK bit in the ISR.
28	CTXBFLL	Write Only	0	Clear High Priority Transmit Buffer Full Interrupt Writing a '1' to this bit clears the TXBFLL bit in the ISR.
29	CTXFLL	Write Only	0	Clear Transmit FIFO Full Interrupt Writing a '1' to this bit clears the TXFLL bit in the ISR.
30	CTXOK	Write Only	0	Clear Transmission Successful Interrupt Writing a '1' to this bit clears the TXOK bit in the ISR.
31	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 controller 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 controller has a configurable Transmit FIFO (TX FIFO) that can store up to 64 messages. The CAN controller 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 rules apply regarding 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 controller 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 controller enters Bus off state or Configuration mode.

Message Structure

Each message is 16 bytes. Byte ordering for CAN message structure is shown in [Table 2-29](#) through [Table 2-32](#).

Table 2-29: Message Identifier [IDR]

0 – 10	11	12	13 – 30	31
ID [28..18]	SRR/RTR	IDE	ID[17..0]	RTR

Table 2-30: Data Length Code [DLCR]

0 – 3	4 – 31
DLC [3..0]	Reserved

Table 2-31: Data Word 1 [DW1R]

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

Table 2-32: Data Word 2 [DW2R]

0 – 7	8 – 15	16 – 23	24 – 31
DB4[7..0]	DB5[7..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 (4 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 8 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 (4 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 controller 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 8 data bytes. Reads 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 4 to 31 in the DLCR
- 0s must be written to Identifier of Received Message (IDR) [13 to 31] 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 2-33](#) provides bit descriptions for the Identifier Word. [Table 2-34](#) describes the DLC Word bits. [Table 2-35](#) describes the Data Word 1 and Data Word 2 bits.

Table 2-33: Identifier Word Bits

Bit(s)	Name	Access	Default Value	Description
0–10	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.
11	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.
12	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.
13–30	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
31	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 2-34: DLC Word Bits

Bit(s)	Name	Access	Default Value	Description
0–3	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.
4–31	Reserved	Read/Write		Reads from this field return 0s. Writes to this field should be 0s.

Table 2-35: Data Word 1 and Data Word 2 Bits

Register	Field	Access	Default Value	Description
DW1R [0..7]	DB0[7..0]	Read/Write	0	Data Byte 0 Reads from this field return invalid data if the message has no data.
DW1R [8..15]	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 [16..23]	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 [24..31]	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 [0..7]	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 [8..15]	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 [16..23]	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 [24..31]	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 greater than or equal to 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 2-36 shows the bit positions in the AFR and Table 2-37 describes the AFR bits.

Table 2-36: Acceptance Filter Register Bit Positions

0 — 27	28	29	30	31
Reserved	UAF4	UAF3	UAF2	UAF1

Table 2-37: Acceptance Filter Register Bits

Bit(s)	Name	Access	Default Value	Description
0–27	Reserved	Read/Write	0	Reserved Reserved for future expansion.
28	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.
29	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.

Table 2-37: Acceptance Filter Register Bits (Cont'd)

Bit(s)	Name	Access	Default Value	Description
30	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.
31	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 Registers

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 greater than or equal to 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 2-38 shows the bit positions in the AFMR and Table 2-39 describes the AFMR bits.

Table 2-38: Acceptance Filter Mask Registers Bit Positions

0 — 10	11	12	13 — 30	31
AMID[28..18]	AMSRR	AMIDE	AMID[17..0]	AMRTR

Table 2-39: Acceptance Filter Mask Bit Descriptions

Bit(s)	Name	Access	Default Value	Description
0–10	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.
11	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.

Table 2-39: Acceptance Filter Mask Bit Descriptions (Cont'd)

Bit(s)	Name	Access	Default Value	Description
12	AMIDE	Read/Write	0	<p>Identifier Extension Mask Used for masking the IDE bit in CAN frames.</p> <p>'1' = Indicates the corresponding bit in Acceptance Mask ID Register is used when comparing the incoming message identifier.</p> <p>'0' = Indicates the corresponding bit in Acceptance Mask ID Register is not used when comparing the incoming message identifier.</p> <p>If AMIDE = '1' and the AIIDE bit in the corresponding Acceptance ID register is '0', this mask is applicable to only Standard frames.</p> <p>If AMIDE = '1' and the AIIDE bit in the corresponding Acceptance ID register is '1', this mask is applicable to only extended frames.</p> <p>If AMIDE = '0' this mask is applicable to both Standard and Extended frames.</p>
13–30	AMID[17..0]	Read/Write	0	<p>Extended Message ID Mask These bits are used for masking the Identifier in an Extended Frame.</p> <p>'1' = Indicates the corresponding bit in Acceptance Mask ID Register is used when comparing the incoming message identifier.</p> <p>'0' = Indicates the corresponding bit in Acceptance Mask ID Register is not used when comparing the incoming message identifier.</p>
31	AMRTR	Read/Write	0	<p>Remote Transmission Request Mask. This bit is used for masking the RTR bit in an Extended Frame.</p> <p>'1' = Indicates the corresponding bit in Acceptance Mask ID Register is used when comparing the incoming message identifier.</p> <p>'0' = Indicates the corresponding bit in Acceptance Mask ID Register is not used when comparing the incoming message identifier.</p>

Acceptance Filter ID Registers

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 greater than or equal to 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 ACFSY 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 2-40 shows AFIR bit positions, and Table 2-41 describes the AFIR bits.

Table 2-40: Acceptance Filter ID Registers Bit Positions

0 — 10	11	12	13 — 30	31
AIID[28..18]	AISRR	AIIDE	AIID[17..0]	AIRTR

Table 2-41: Acceptance Filter ID Registers Bits

Bit(s)	Name	Access	Default Value	Description
0–10	AIID [28..18]	Read/Write	0	Standard Message ID Standard Identifier
11	AISRR	Read/Write	0	Substitute Remote Transmission Request. Indicates the Remote Transmission Request bit for Standard frames
12	AIIDE	Read/Write	0	Identifier Extension Differentiates between Standard and Extended frames
13–30	AIID[17..0]	Read/Write	0	Extended Message ID Extended Identifier
31	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. The chapter contains the following sections.

- [Configuring the CAN Controller](#)
- [Clocking](#)
- [Resets](#)
- [Interrupts](#)
- [Xilinx CAN Controller Design Parameters](#)

Configuring the CAN Controller

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 Loop Back 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 '0's 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 controller 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](#).

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](#).

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.

Extra Design Consideration

The CAN and Xilinx Platform Studio (XPS) CAN cores require 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 and XPS CAN cores.

Clocking

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

- CAN_CLK can be 8 to 24 MHz in frequency
- S_AXI_CLK can be 8 to 100 MHz in frequency
- CAN_CLK and S_AXI_CLK 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

You can specify the operating frequency for S_AXI_ACLK; using a DCM to generate S_AXI_ACLK is optional.

CAN_CLK

The range of CAN_CLK clock is 8–24 MHz.

You determine 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 controller. The S_AXI_ARESETN input mentioned in [Table 2-7](#) 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](#).)

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 '1' 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 '1.'

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 IP core uses a hard-vector interrupt mechanism. It has a single interrupt line (`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).

Xilinx CAN Controller Design Parameters

To obtain a CAN controller tailored to meet your minimum system requirements, specific features are parameterized. This results in a design using only the required resources, providing the best possible performance. Table 3-1 shows the CAN controller features that can be parameterized.

Table 3-1: CAN Controller Design Parameters

Feature Description	Parameter Name	Allowable Values	Default Value
Target FPGA Family	C_FAMILY	Spartan-6 and Virtex-6	
Depth of the RX FIFO	C_CAN_RX_DPTH	2, 4, 8, 16, 32, 64	2
Depth of the TX FIFO	C_CAN_TX_DPTH	2, 4, 8, 16, 32, 64	2
Number of Acceptance Filters used	C_CAN_NUM_ACF	0 - 4	0
Based Address of Xilinx CAN controller	C_S_AXI_BASEADDR	Valid Address	See note (1) and (2)
High Address of CAN Xilinx controller	C_S_AXI_HIGHADDR	Valid Address	See note (1) and (2)
AXI Address bus width	C_S_AXI_ADDR_WIDTH	32	32
AXI Data bus width	C_S_AXI_DATA_WIDTH	32	32

1. Address range is specified by C_S_AXI_BASEADDR and C_S_AXI_HIGHADDR must be at least 0x100 and must be power of 2. C_S_AXI_BASEADDR must be multiple of the range, where the range is C_S_AXI_HIGHADDR - C_S_AXI_BASEADDR + 1. Also make sure that LSB 8 bits of the C_S_AXI_BASEADDR to be zero.
2. No default value is specified to ensure that the actual value is set, that is, if the value is not set, a compiler error is generated. The address range must be at least 0x00FF. For example, C_S_AXI_BASEADDR = 0x80000000, C_S_AXI_HIGHADDR = 0x800000FF

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 bit in the ISR that is '1' (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 assertion condition remains true.

SECTION II: VIVADO DESIGN SUITE

Customizing and Generating the Core

Constraining the Core

Detailed Example Design

Customizing and Generating the Core

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

CAN Graphical User Interface

The CAN graphical user interface (GUI) provides a single screen for configuring the CAN core.

Figure 4-1 shows the main CAN customization screens, which you use to set the component name and core options, described in the following sections.

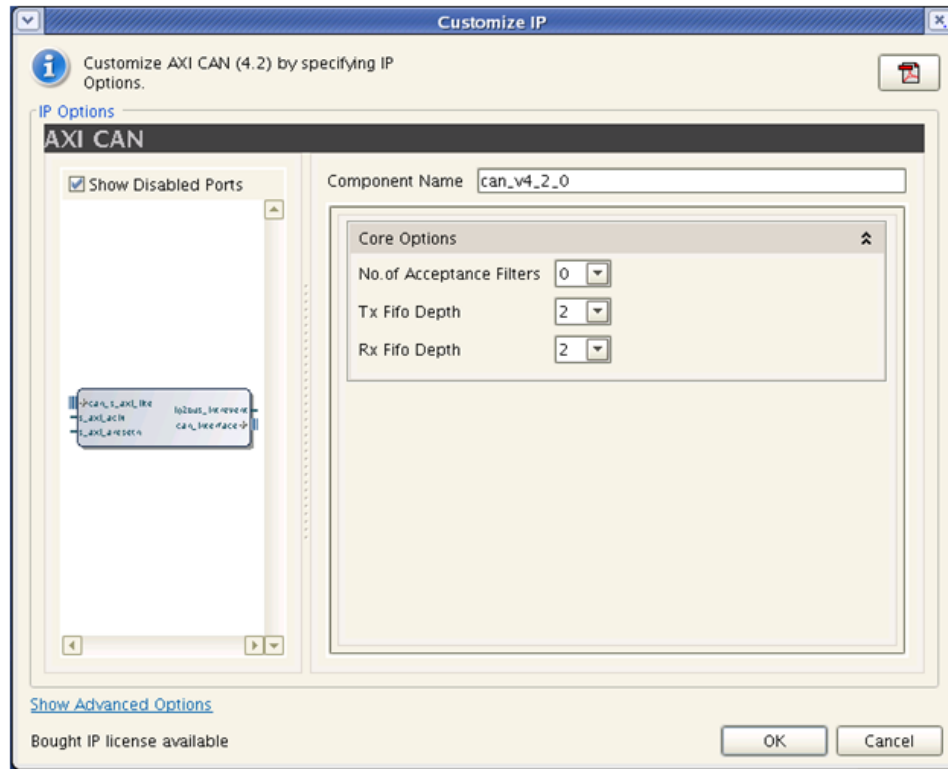


Figure 4-1: Vivado Main Screen

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 controller. 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, TX FIFO with a depth of 2 can hold at most 2 CAN messages.

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

RX FIFO Depth

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

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

Parameter Values in the XCI File

Table 4-1: Parameter Values in the XCI File

Parameter	Value
C_CAN_RX_DPTH	2
C_CAN_TX_DPTH	2
C_CAN_NUM_ACF	0
C_FAMILY	virtex7

Output Generation

This section provides detailed information about the example design, including a description of files and the directory structure generated by the Xilinx Vivado Design Suite, the purpose and contents of the provided scripts, the contents of the example HDL wrappers, and the operation of the demonstration test bench.

In the IP Catalog project, clicking Open IP Example Design in GUI or typing the command

```
open_example_project [get_ips <component_name>]
```

in the TCL console invokes a separate example design project. In this new project `<component_name>_exdes` is the top module for synthesis, and `<component_name>_tb` is the top module for simulation. The implementation or simulation of the example design can be run from the example project.

Directory and File Contents

 `<project_name>/<project_name>.srcs/sources_1/ip/<component name>`

Top-level project directory; name is user-defined

 `<project_name>/<project_name>.srcs/sources_1/ip/<component name>`

Core release notes file

 `<component_name>example design`




Verilog and VHDL design files

 `<component_name>/implement`

Implementation script files

 `<component_name>/implement/results`

Results directory, created after implementation scripts are run, and contains implement script results

- 
 <component_name>/simulation
Simulation scripts
- 
 <component_name>/simulation/functional
Functional simulation files
- 
 simulation/timing
Simulation files

The directory structure for a Vivado design suite project under <project_name>/<project_name>.srcs/sources_1/ip/<component name> is same as <project_directory>/<component name> for a CORE Generator™ tools project.

<project_name>/<project_name>.srcs/sources_1/ip/<component name>

This directory contains all the Vivado Design Suite project files.

Table 4-2: Project Directory

Name	Description
project_name>/<project_name>.srcs/sources_1/ip/<component name>	
<component_name>.xci	Vivado device tools project-specific option file; can be used as an input to the Vivado Design Suite
<component_name>.{veo vho}	VHDL or Verilog instantiation template
<component_name>.xdc	Constraints file for core.

<project_directory>/<component name>

The <component name> directory contains the release notes file provided with the core, which can include last-minute changes and updates.

Table 4-3: Component Name Directory

Name	Description
<project_dir>/<component_name>	
can_release_notes.txt	Core name release notes file.

<component_name>example design

The example design directory contains the example design files provided with the core.

Table 4-4: Example Design Directory

Name	Description
<project_dir>/<component_name>/example_design	
<component_name>_top.ucf	Provides example constraints necessary for processing the CAN core using the Xilinx implementation tools.
<component_name>_top.v[hd]	The VHDL or Verilog top-level file for the example design; it instantiates the CAN core.
<component_name>.v	Top-level file for the example design. Only generated when Verilog design flow is selected.

<component_name>/implement

The implement directory contains the core implementation script files. Generated for Full-System Hardware Evaluation and Full license types.

Table 4-5: Implement Directory

Name	Description
<project_dir>/<component_name>/implement	
implement.{bat sh}	A Windows (.bat) or Linux script that processes the example design.
xst.prj	The XST project file for the example design that lists all of the source files to be synthesized. Only available when the CORE Generator system project option is set to ISE or Other.
xst.scr	The XST script file for the example design used to synthesize the core. Only available when the CORE Generator system. Vendor project option is set to ISE or Other.

<component_name>/implement/results

The results directory is created by the implement script, after which the implement script results are placed in the results directory.

Table 4-6: Results Directory

Name	Description
<project_dir>/<component_name>/implement/results	
Implement script result files.	

<component_name>/simulation

The simulation directory contains the simulation scripts provided with the core.

Table 4-7: Simulation Directory

Name	Description
<project_dir>/<component_name>/simulation	
gbl.v	Verilog test file provided with the demonstration test bench.
can_v4_2_tb.v[hd]	Verilog/VHDL test file provided with the demonstration test bench.

<component_name>/simulation/functional

The functional directory contains functional simulation scripts provided with the core.

Table 4-8: Functional Directory

Name	Description
<project_dir>/<component_name>/simulation/functional	
simulate_mti.do	A macro file for ModelSim that compiles the HDL sources and runs the simulation.
simulate_ncsim.sh	A macro file for Cadence IES that compiles the HDL sources and runs the simulation in a Linux environment.
simulate_ncsim.bat	A macro file for Cadence IES that compiles the HDL sources and runs the simulation in a Windows environment.
wave.do	A macro file for ModelSim that opens a wave window and adds key signals to the wave viewer. This file is called by the simulate_mti.do file and is displayed after the simulation is loaded.
wave.sv	A macro file for Cadence IES that opens a wave window and adds key signals to the wave viewer.

simulation/timing

The timing simulation directory is generated only for Full-System Hardware Evaluation and Full-license types.

Table 4-9: Timing Directory

Name	Description
<project_dir>/<component_name>/simulation/timing	
simulate_mti.do	A macro file for ModelSim that compiles the post-par timing netlist, demonstration test bench files, and runs the simulation.
simulate_ncsim.sh	A macro file for Cadence IES that compiles the post-par timing netlist, demonstration test bench files, and runs the simulation in a Linux environment.
simulate_ncsim.bat	A macro file for Cadence IES that compiles the post-par timing netlist, demonstration test bench files, and runs the simulation in a Windows environment.
wave.do	A macro file for ModelSim that opens a wave window and adds key signals to the wave viewer. This file is called by the simulate_mti.do file and is displayed after the simulation is loaded.
wave.sv	A macro file for Cadence IES that opens a wave window and adds key signals to the wave viewer.

Constraining the Core

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

Required Constraints

```
set_false_path -from [all_registers -clock $clk_domain_a] -to [all_registers  
-clock $clk_domain_b]
```

```
set_false_path -from [all_registers -clock $clk_domain_b] -to [all_registers  
-clock $clk_domain_a]
```

Clock Frequencies

- CAN_CLK can be 8 to 24 MHz in frequency.
- S_AXI_CLK can be 8 to 100 MHz in frequency.
- CAN_CLK and S_AXI_CLK can be asynchronous or can be clocked from the same source.

Detailed Example Design

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

Figure 6-1 illustrates the CAN example design.

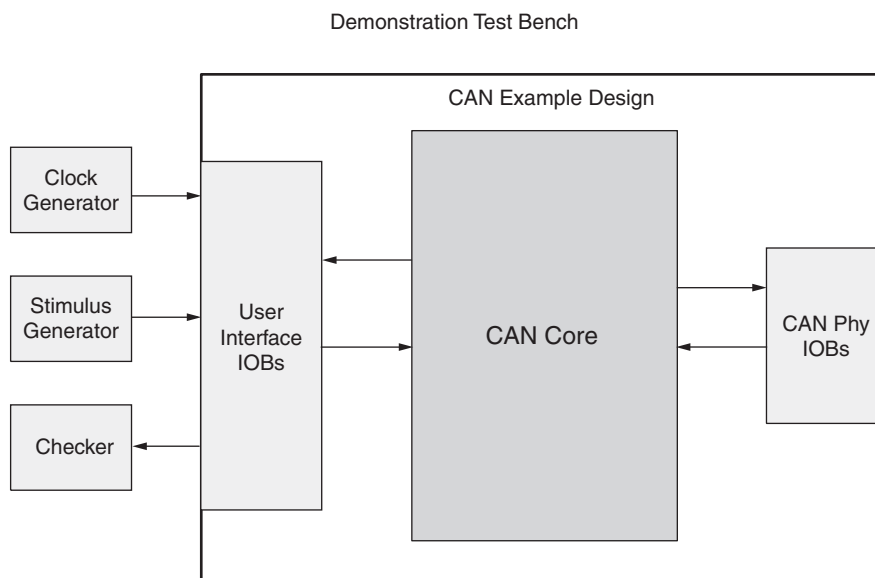


Figure 6-1: Example Design

The CAN example design consists of the following:

- CAN netlist
- HDL wrapper which instantiates the CAN netlist
- Demonstration test bench that is customized to simulate the example design

The CAN example design has been tested with Xilinx ISE® tools 14.4, Vivado Design Suite 2012.4, and the Mentor Graphics ModelSim v10.1a simulator.

Generating the Core

Vivado IP Catalog

This section describes how to generate a CAN core with default values using Vivado Design Suite.

To generate the core:

1. Start the Vivado Design Suite.
2. Choose **File > New Project**.
3. Click **Next** and select project name and project location.
4. Keep default settings for the succeeding pages until "Default Part" page.
5. Select the desired part.
Note: If an unsupported silicon family is selected, the CAN core will not appear in the taxonomy tree.
6. Click Finish to create the project.
7. After creating the project, locate the CAN core in the taxonomy tree under **Automotive & Industrial > Automotive** or **Embedded Processing > AXI Peripheral > Low Speed Peripheral > AXI CAN**.
8. Double-click the core to display the main configuration screen.

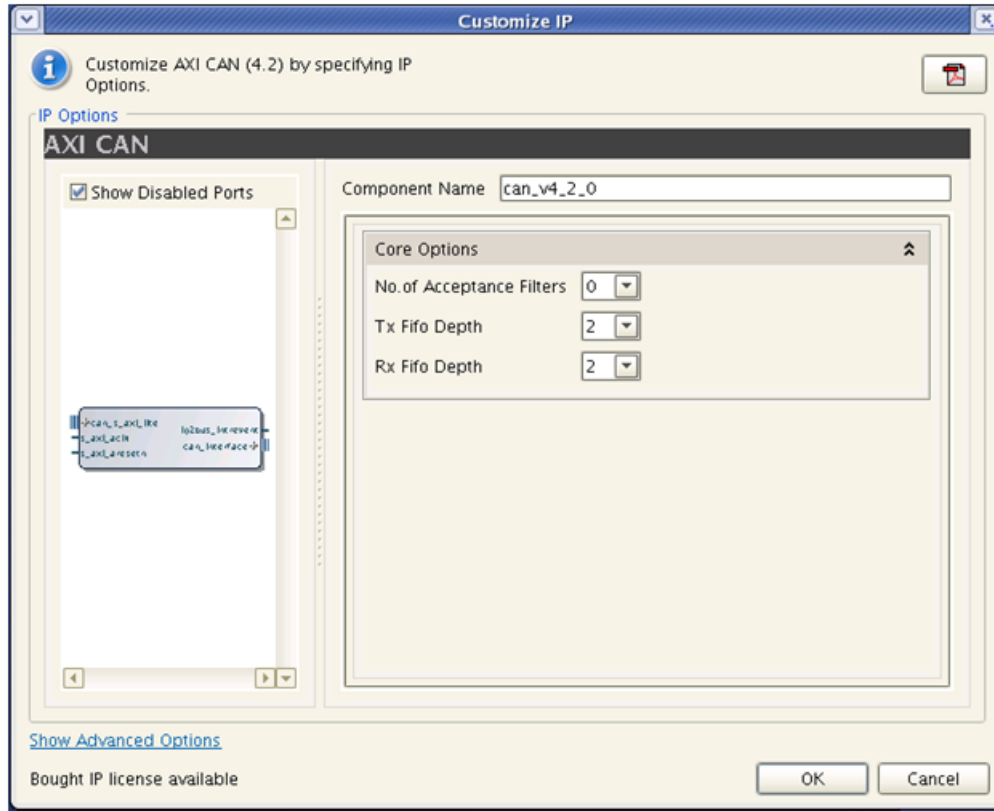


Figure 6-2: Vivado Main Screen

9. In the Component Name field, enter a name for the core instance.
10. After selecting the parameters from the GUI screen, click **Finish**.

Field Descriptions

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 controller. 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, TX FIFO with a depth of 2 can hold at most 2 CAN messages.

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

RX FIFO Depth

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

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

Implementing the Example Design

After generating a core with either a Full-System Hardware Evaluation or Full license, the netlists and example design can be processed by the Xilinx implementation tools. The generated output files include scripts to assist you in running the Xilinx software.

To implement the CAN example design, open a command prompt or terminal window and type these commands:

For Windows:

```
ms-dos> cd <proj_dir>\quickstart\implement
ms-dos> implement.bat
```

For Linux:

```
Linux-shell% cd <proj_dir>/quickstart/implement
Linux-shell% ./implement.sh
```

These commands execute a script that synthesizes, builds, maps, and places-and-routes the example design. The script then generates a post-par simulation model for use in timing simulation. The resulting files are placed in the results directory.

For Vivado design tools:

1. Right-click on generated core, select "Open IP Example Design."
This creates a new project with the example design.
2. Click "Run Implementation" to implement the example design.

Note: Equivalent tcl commands can be found on the "tcl console" of the Vivado Design Suite.

Simulating the Example Design

The CAN core provides a quick way to simulate and observe the behavior of the core by using the provided example design. There are two different simulation types: functional and timing. The simulation models provided are either in VHDL or Verilog, depending on the IP catalog system Design Entry project option.

Setting up for Simulation

The Xilinx UNISIM and SIMPRIM libraries must be mapped into the simulator. If the UNISIM or SIMPRIM libraries are not set for your environment, go to the *Synthesis and Simulation Guide* in the [Xilinx Software Manuals](#) for assistance compiling Xilinx simulation models. Simulation scripts are provided for ModelSim.

Functional Simulation

This section provides instructions for running a functional simulation of the CAN core using either VHDL or Verilog. Functional simulation models are provided when the core is generated. Implementing the core before simulating the functional models is not required.

To run a VHDL or Verilog functional simulation of the example design:

1. Set the current directory to:

```
<quickstart>/simulation/functional/
```

2. Launch the simulation script.

```
ModelSim: vsim -do simulate_mti.do
```

```
ncsim (ms-dos>): simulate_ncsim.bat
```

```
ncsim (Linux-shell%): ./simulate_ncsim.sh
```

The simulation script compiles the functional simulation models and demonstration test bench, adds relevant signals to the wave window, and runs the simulation. To observe the operation of the core, inspect the simulation transcript and the waveform.

For Vivado design tools:

1. Right-click on generated core, select "Open IP Example Design."
This creates a new project with the example design.
2. Click "Run simulation" to simulate the example design.

Note: Equivalent tcl commands can be found on the "tcl console" of the Vivado design suite.

Timing Simulation

Timing simulation is supported only for the Full-System Hardware Evaluation and Full license types, as the core cannot be implemented using a Simulation Only Evaluation license. This section contains instructions for running a timing simulation of the CAN core using either VHDL or Verilog. A timing simulation model is generated when the core is run through the Xilinx tools using the implement script. It is a requirement that the core is implemented before attempting to run timing simulation.

To run a VHDL or Verilog functional simulation of the example design:

1. Set the current directory to:

```
<quickstart>/simulation/timing/
```

2. Launch the simulation script:

```
ModelSim: vsim -do simulate_mti.do
```

```
ncsim (ms-dos>) : simulate_ncsim.bat
```

```
ncsim (Linux-shell%) : ./simulate_ncsim.sh
```

The simulation script compiles the timing simulation model and the demonstration test bench, adds relevant signals to the wave window, and runs the simulation. To observe the operation of the core, inspect the simulation transcript and the waveform.

For Vivado design tools:

1. Right-click on generated core, select "Open IP Example Design"
This creates a new project with the example design.
2. Click "Run implementation" to implement the design.
3. Copy the implementation results to the directory specified in [Directory and File Contents in Chapter 3](#) for Vivado design tools.
4. Use the scripts in simulation/timing directory to run timing simulations.

Note: Equivalent tcl commands can be found on the "tcl console" of the Vivado design suite.

Directory and File Contents

See [Directory and File Contents in Chapter 4](#).

Example Design Configuration

Figure 6-3 illustrates the example design configuration.

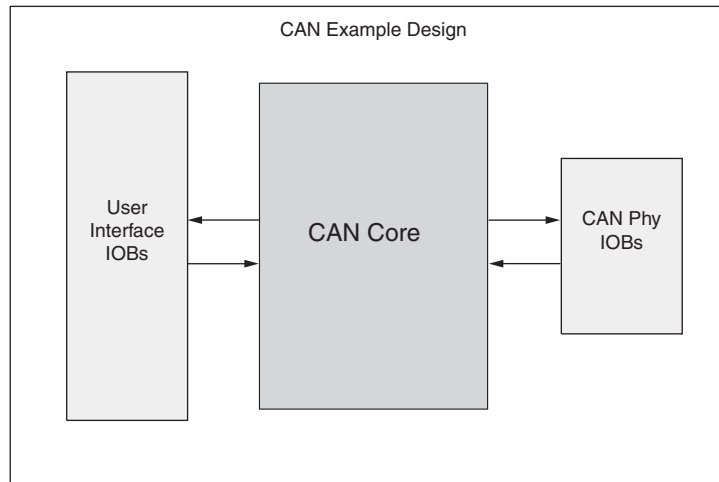


Figure 6-3: Example Design Configuration

The example design contains the following:

- An instance of the CAN core

During simulation, the CAN core is instantiated as a black box and replaced with the netlist during implementation and the gate-level simulation model.

- Input and output buffers for top-level port signal

Demonstration Test Bench

Figure 6-4 illustrates the demonstration test bench.

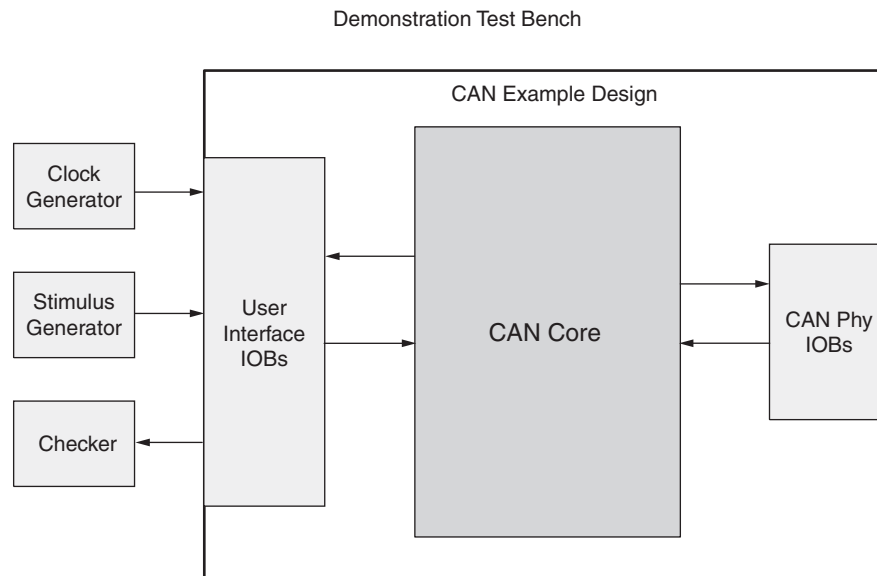


Figure 6-4: **Demonstration Test Bench**

Test Bench Functionality

The demonstration test bench is a straightforward VHDL or Verilog file to exercise the example design and the core itself.

The test bench consists of the following:

- Clock Generators
- Procedure to write to a Configuration Register memory location
- Procedure to read from a Configuration Register memory location
- Procedure to display the bits set in the Interrupt Status Register (ISR)

Core with No Acceptance Filtering

The demonstration test bench performs the following tasks:

- Input clock signals are generated.
- A reset is applied to the example design.
- The Baud Rate Prescaler register and Bit Timing registers are written to. These registers are read from and the values read are compared with the values written.

- The Interrupt Enable Register is written to enable interrupts for TXBFL and RXOK bits. This register is read from and the value read is compared with the value written.
- The Mode Select Register is written to select Loop Back mode of operation. This register is read from and the value read is compared with the value written.
- The Software Reset Register is written to enable CEN bit. This register is read from and the value written is compared with the value read.
- Five messages are written in sequence:
 1. The first message is written to the TXHPB and is a standard data frame.
 2. The second message is written to the TX FIFO and is a standard data frame.
 3. The third message is written to the TX FIFO and is a standard remote frame.
 4. The fourth message is written to the TX FIFO and is an extended data frame.
 5. 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 the interrupt line. When the interrupt line is asserted, the following conditions occur:

- The bits set in the ISR are displayed.
- The RX FIFO is read if the RXOK bit is set. The message received is compared with the message previously transmitted.
- The ICR is written to. This clears the bits in the ISR that are set.

With no acceptance filtering, all five messages are received in the RX FIFO.

Core with Acceptance Filtering

The demonstration test bench performs the following tasks:

- Input clock signals are generated.
- A reset is applied to the example design.
- The Baud Rate Prescaler register and Bit Timing registers are written to. These registers are read from and the values read are compared with the values written.
- The Interrupt Enable Register is written to enable interrupts for TXBFL and RXOK bits. This register is read from and the value read is compared with the value written.
- Acceptance Filter ID Register 1 and Acceptance Filter Mask Register 1 are written to. These registers are read from and the values read are compared with the values written.
- The Acceptance Filter Register is written to enable Acceptance Filter pair 1. This register is read from and the value read is compared with the value written.
- The Mode Select Register is written to select Loop Back mode. This register is read from and the value read is compared with the value written.

- The Software Reset Register is written to enable CEN bit. This register is read from and the value written is compared with the value read.
- Five messages are written in a sequence.
 1. The first message is written to the TXHPB and is a standard data frame.
 2. The second message is written to the TX FIFO and is a standard data frame.
 3. The third message is written to the TX FIFO and is a standard remote frame.
 4. The fourth message is written to the TX FIFO and is an extended data frame.
 5. 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 interrupt line to be asserted. When the interrupt line is asserted, these conditions occur:

- The bits in the ISR that are set are displayed.
- The RX FIFO is read if the RXOK bit is set. The message that is received is compared with the message that was transmitted.
- The ICR is written to. This clears the bits in the ISR that are set.
- After the fourth message is transmitted and received, the Interrupt Enable Register is written to enable interrupts for TXOK, RXOK and TXBFLL. This register is read from and the value read is compared with the value written.
- The fifth message does not pass acceptance filtering. Only the TXOK bit in the ISR is set when the ISR is asserted.

Customizing the Demonstration Test Bench

This section describes the variety of demonstration test bench customization options that can be used for individual system requirements.

Changing the Data

You can change the contents of the message written to the TX FIFO / TX HPB by changing the procedure call that writes to the TX FIFO and the TX HPB memory locations. The relevant fields in the checkers must also be changed to ensure that the message read from the RX FIFO matches the message that was transmitted.

Changing the CAN Parameters

The values written to the BRPR and the BTR registers can be changed for appropriate bit timing values. The test bench operates in the Loop Back mode of operation.

Changing the Test Bench Structure

You can add messages using these steps.

1. Write the message to the TX FIFO.
2. Wait for an interrupt and process the interrupt.
3. Read the received message from the RX FIFO.

Implementation

To implement the example design, select **Run Implementation** in the Vivado Project Manager window. For further details on setting up the implementation, see the *Vivado Design Suite User Guide, Implementation* ([UG911](#)).

SECTION III: ISE DESIGN SUITE

Customizing and Generating the Core

Constraining the Core

Detailed Example Design

Customizing and Generating the Core

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

GUI

The CAN graphical user interface (GUI) provides a single screen for configuring the CAN core.

Figure 7-1 shows the main CAN customization screens, which you use to set the component name and core options, described in the following sections.

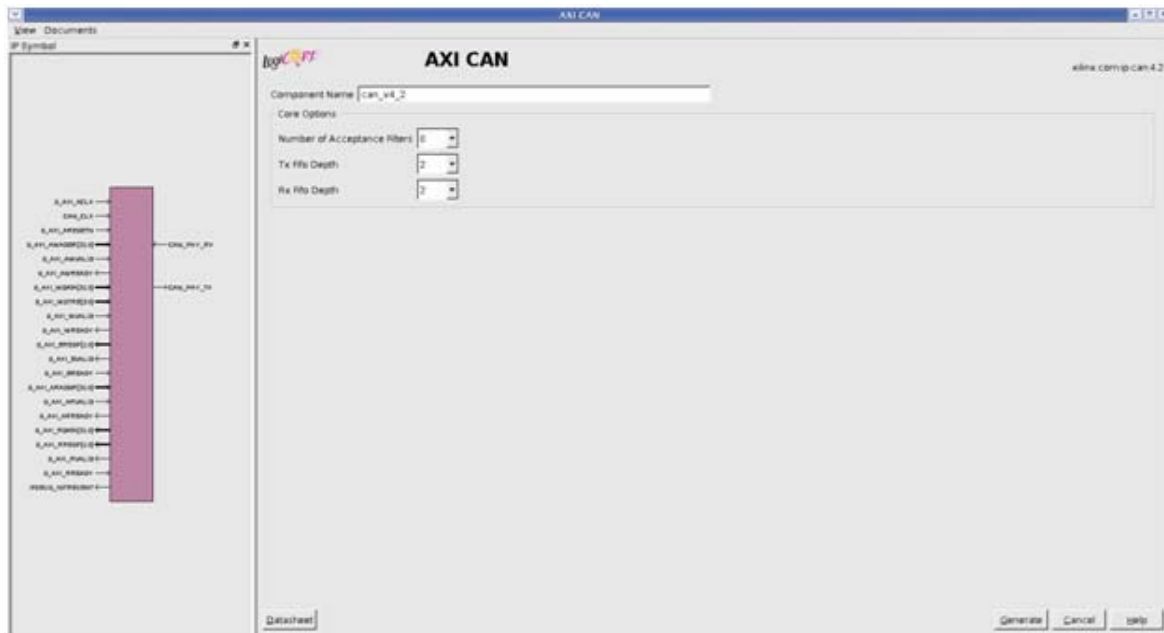


Figure 7-1: CORE Generator Main Screen

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 controller. 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, TX FIFO with a depth of 2 can hold at most 2 CAN messages.

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

RX FIFO Depth

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

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

Parameter Values in the XCO File

Table 7-1: Parameter Values in the XCO File

component_name	can_v4_2
number_of_acceptance_filters	0
rx_fifo_depth	2
tx_fifo_depth	2

Output Generation

Directory and File Contents

The CAN v4.2 core directories and their associated files are defined in the following sections.

 **<project directory>**

Top-level project directory; name is user-defined

 **<project_directory>/<component name>**

Core release notes file

 **<component_name>/doc**

Product documentation

 **<component_name>example design**

Verilog and VHDL design files

 **<component_name>/implement**

Implementation script files

 **<component_name>/implement/results**

Results directory, created after implementation scripts are run, and contains implement script results

 **<component_name>/simulation**

Simulation scripts

 **<component_name>/simulation/functional**

Functional simulation files

 **simulation/timing**

Simulation files

<project directory>

The <project directory> contains all the CORE Generator™ tools project files.

Table 7-2: Project Directory

Name	Description
<project_dir>	
<component_name>.ngc	Top-level netlist
<component_name>.v[hd]	Verilog or VHDL simulation model
<component_name>.xco	CORE Generator tool project-specific option file; can be used as an input to the CORE Generator system.
<component_name>_flist.txt	List of files delivered with the core.
<component_name>.{veo vho}	VHDL or Verilog instantiation template.

[Back to Top](#)

<project_directory>/<component name>

The <component name> directory contains the release notes file provided with the core, which can include last-minute changes and updates.

Table 7-3: Component Name Directory

Name	Description
<project_dir>/<component_name>	
can_release_notes.txt	Core name release notes file.

[Back to Top](#)

<component_name>example design

The example design directory contains the example design files provided with the core.

Table 7-4: Example Design Directory

Name	Description
<project_dir>/<component_name>/example_design	
<component_name>_top.ucf	Provides example constraints necessary for processing the CAN core using the Xilinx implementation tools.
<component_name>_top.v[hd]	The VHDL or Verilog top-level file for the example design; it instantiates the CAN core.
<component_name>.v	Top-level file for the example design. Only generated when Verilog design flow is selected.

[Back to Top](#)

<component_name>/doc

The doc directory contains the PDF documentation provided with the core.

Table 7-5: Doc Directory

Name	Description
<project_dir>/<component_name>/doc	
pg096-can.pdf	product guide

[Back to Top](#)

<component_name>/implement

The implement directory contains the core implementation script files. Generated for Full-System Hardware Evaluation and Full license types.

Table 7-6: Implement Directory

Name	Description
<project_dir>/<component_name>/implement	
implement.{bat sh}	A Windows (.bat) or Linux script that processes the example design.
xst.prj	The XST project file for the example design that lists all of the source files to be synthesized. Only available when the CORE Generator system project option is set to ISE or Other.
xst.scr	The XST script file for the example design used to synthesize the core. Only available when the CORE Generator system. Vendor project option is set to ISE or Other.

[Back to Top](#)

<component_name>/implement/results

The results directory is created by the implement script, after which the implement script results are placed in the results directory.

Table 7-7: Results Directory

Name	Description
<project_dir>/<component_name>/implement/results	
Implement script result files.	

[Back to Top](#)

<component_name>/simulation

The simulation directory contains the simulation scripts provided with the core.

Table 7-8: Simulation Directory

Name	Description
<project_dir>/<component_name>/simulation	
gbl.v	Verilog test file provided with the demonstration test bench.
can_v4_2_tb.v[hd]	Verilog/VHDL test file provided with the demonstration test bench.

[Back to Top](#)

<component_name>/simulation/functional

The functional directory contains functional simulation scripts provided with the core.

Table 7-9: Functional Directory

Name	Description
<project_dir>/<component_name>/simulation/functional	
simulate_mti.do	A macro file for ModelSim that compiles the HDL sources and runs the simulation.
simulate_ncsim.sh	A macro file for Cadence IES that compiles the HDL sources and runs the simulation in a Linux environment.
simulate_ncsim.bat	A macro file for Cadence IES that compiles the HDL sources and runs the simulation in a Windows environment.
wave.do	A macro file for ModelSim that opens a wave window and adds key signals to the wave viewer. This file is called by the simulate_mti.do file and is displayed after the simulation is loaded.
wave.sv	A macro file for Cadence IES that opens a wave window and adds key signals to the wave viewer.

[Back to Top](#)

simulation/timing

The timing simulation directory is generated only for Full-System Hardware Evaluation and Full-license types.

Table 7-10: Timing Directory

Name	Description
<project_dir>/<component_name>/simulation/timing	
simulate_mti.do	A macro file for ModelSim that compiles the post-par timing netlist, demonstration test bench files, and runs the simulation.
simulate_ncsim.sh	A macro file for Cadence IES that compiles the post-par timing netlist, demonstration test bench files, and runs the simulation in a Linux environment.
simulate_ncsim.bat	A macro file for Cadence IES that compiles the post-par timing netlist, demonstration test bench files, and runs the simulation in a Windows environment.
wave.do	A macro file for ModelSim that opens a wave window and adds key signals to the wave viewer. This file is called by the simulate_mti.do file and is displayed after the simulation is loaded.
wave.sv	A macro file for Cadence IES that opens a wave window and adds key signals to the wave viewer.

[Back to Top](#)

Constraining the Core

This chapter contains information about constraining the core in the ISE® Design Suite.

Device and Package Selection

The CAN controller can be implemented in Virtex®-6, XA Spartan®-6 and Spartan-6 devices. Ensure that the device used has these attributes:

- The device is large enough to accommodate the core.
 - The device contains a sufficient number of Input Output Blocks (IOBs.)
-

Location Constraints

No specific I/O location constraints.

Placement Constraints

No specific placement constraints.

Timing Constraints

The core has two different clock domains: S_AXI_ACLK and CAN_CLK. The following constraints can be used with the CAN Controller.

PERIOD Constraints for Clock Nets

CAN_CLK



IMPORTANT: *The clock provided to CAN_CLK must be constrained for a clock frequency of less than or equal to 24 MHz, based on the input oscillator frequency.*

```
# Set the CAN_CLK constraints
NET "CAN_CLK" TNM_NET = "CAN_CLK";
TIMESPEC "TS_CAN_CLK" = PERIOD "CAN_CLK" 40 ns HIGH 50%;
```

S_AXI_ACLK



IMPORTANT: *The clock provided to S_AXI_ACLK must be constrained for a clock frequency of 100 MHz or less.*

```
# Set the S_AXI_ACLK constraints
# This can be relaxed based on the actual frequency
NET "S_AXI_ACLK" TNM_NET = "S_AXI_ACLK";
TIMESPEC "TS_S_AXI_ACLK" = PERIOD "S_AXI_ACLK" 10 ns HIGH 50%;
```

Timing Ignore Constraints



IMPORTANT: *A timing ignore (TIG) constraint must be specified on all the signals that cross clock domains.*

```
# Timing Ignore constraint on all signals that cross from CAN_CLK domain to
S_AXI_ACLK domain
TIMESPEC "TS_CAN_SYS_TIG" = FROM "CAN_CLK" TO "S_AXI_ACLK" TIG;
# Timing Ignore constraint on all signals that cross from S_AXI_ACLK domain to
CAN_CLK domain
TIMESPEC "TS_SYS_CAN_TIG" = FROM "S_AXI_ACLK" TO "CAN_CLK" TIG;
```

I/O Constraints

These constraints ensure that the flops associated with the external I/O signals are placed in IOBs.

```
# These constraints need to be given if the CAN controller is used in a stand-alone
mode.
INST "S_AXI_ARESETN" ION = true;
INST "S_AXI_AWADDR*" IOB = true;
INST "S_AXI_AWVALID" IOB = true;
INST "S_AXI_AWREADY" IOB = true;
INST "S_AXI_WDATA*" IOB = true;
INST "S_AXI_WSTB*" IOB = true;
INST "S_AXI_WVALID" IOB = true;
INST "S_AXI_WREADY" IOB = true;
INST "S_AXI_BRESP" IOB = true;
INST "S_AXI_BVALID" IOB = true;
INST "S_AXI_BREADY" IOB = true;
INST "S_AXI_ARADDR*" IOB = true;
INST "S_AXI_ARVALID" IOB = true;
INST "S_AXI_ARREADY" IOB = true;
INST "S_AXI_RDATA*" IOB = true;
INST "Ip2Bus_IntrEvent" IOB = true;
INST "S_AXI_ARESETNIOB" = true;
```

I/O Standards

The pins that interface to the CAN PHY device have a 3.3 volt signal level interface. These constraints can be used provided the device I/O Banking rules are followed:

```
# Select the I/O standards for the interface to the CAN PHY
INST "CAN_PHY_TX"          IOSTANDARD = "LVTTTL"
INST "CAN_PHY_RX"          IOSTANDARD = "LVTTTL"
```

Detailed Example Design

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

Figure 9-1 illustrates the CAN example design.

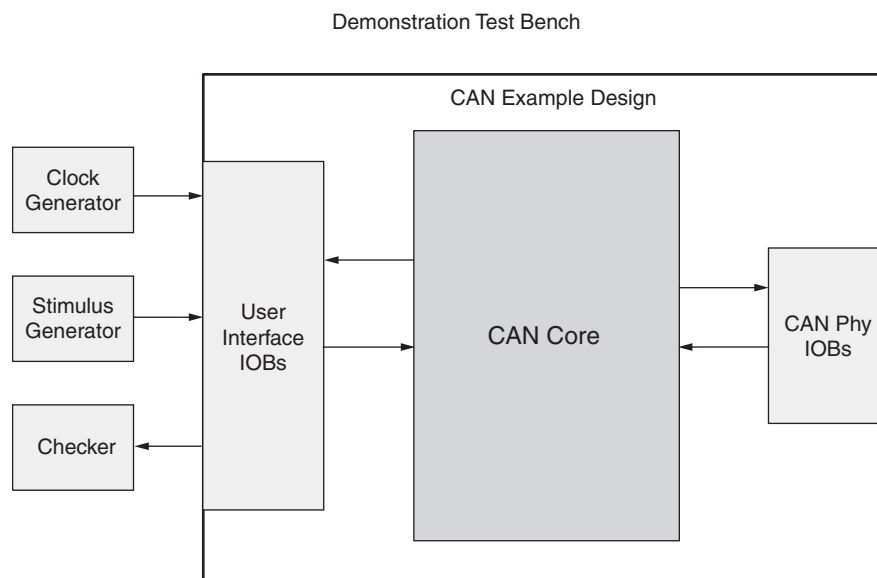


Figure 9-1: Example Design

The CAN example design consists of the following:

- CAN netlist
- HDL wrapper which instantiates the CAN netlist
- Demonstration test bench that can be customized to simulate the example design

The CAN example design has been tested with Xilinx ISE tools 14.4, Vivado™ Design Suite 2012.4, and the Mentor Graphics ModelSim v10.1a simulator.

Generating the Core

This section describes how to generate a CAN core with default values using the Xilinx CORE Generator™ tool.

To generate the core:

1. Start the CORE Generator tool.
2. Choose **File > New Project**.
3. Type a directory name.

This example uses the directory name *design*.

4. Do the following to set project options:
 - Part Options
 - From Target Architecture, select the desired family. For a list of supported families, see [IP Facts](#).
 - Generation Options
 - For Design Entry, select either VHDL or Verilog.
5. After creating the project, locate the CAN core in the taxonomy tree under **Automotive & Industrial > Automotive > CAN**.
6. Double-click the core to display the main CAN configuration screen

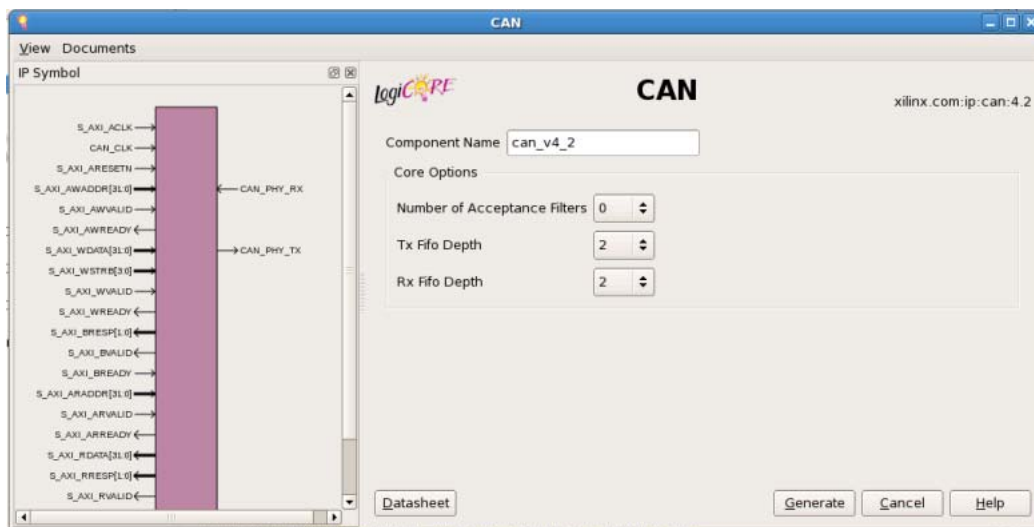


Figure 9-2: CORE Generator System Main Screen

7. In the Component Name field, enter a name for the core instance.

This example uses the name *quickstart*.

8. After selecting the parameters from the GUI screens, click **Finish**.

The core and its supporting files, including the example design, are generated in the project directory. For detailed information about the example design files and directories see [Chapter 3, Detailed Example Design](#).

Field Descriptions

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 controller. 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, TX FIFO with a depth of 2 can hold at most 2 CAN messages.

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

RX FIFO Depth

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

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

Implementing the Example Design

After generating a core with either a Full-System Hardware Evaluation or Full license, the netlists and example design can be processed by the Xilinx implementation tools. The generated output files include scripts to assist you in running the Xilinx software.

To implement the CAN example design, open a command prompt or terminal window and type these commands:

For Windows:

```
ms-dos> cd <proj_dir>\quickstart\implement
ms-dos> implement.bat
```

For Linux:

```
Linux-shell% cd <proj_dir>/quickstart/implement
Linux-shell% ./implement.sh
```

These commands execute a script that synthesizes, builds, maps, and places-and-routes the example design. The script then generates a post-par simulation model for use in timing simulation. The resulting files are placed in the results directory.

Simulating the Example Design

The CAN core provides a quick way to simulate and observe the behavior of the core by using the provided example design. There are two different simulation types: functional and timing. The simulation models provided are either in VHDL or Verilog, depending on the CORE Generator system Design Entry project option.

Setting up for Simulation

The Xilinx UNISIM and SIMPRIM libraries must be mapped into the simulator. If the UNISIM or SIMPRIM libraries are not set for your environment, go to the *Synthesis and Simulation Guide* in the [Xilinx Software Manuals](#) for assistance compiling Xilinx simulation models. Simulation scripts are provided for ModelSim.

Functional Simulation

This section provides instructions for running a functional simulation of the CAN core using either VHDL or Verilog. Functional simulation models are provided when the core is generated. Implementing the core before simulating the functional models is not required.

To run a VHDL or Verilog functional simulation of the example design:

1. Set the current directory to:

```
<quickstart>/simulation/functional/
```

2. Launch the simulation script.

```
ModelSim: vsim -do simulate_mti.do
```

```
ncsim (ms-dos>) : simulate_ncsim.bat
```

```
ncsim (Linux-shell%) : ./simulate_ncsim.sh
```

The simulation script compiles the functional simulation models and demonstration test bench, adds relevant signals to the wave window, and runs the simulation. To observe the operation of the core, inspect the simulation transcript and the waveform.

Timing Simulation

Timing simulation is supported only for the Full-System Hardware Evaluation and Full license types, as the core cannot be implemented using a Simulation Only Evaluation license. This section contains instructions for running a timing simulation of the CAN core using either VHDL or Verilog. A timing simulation model is generated when the core is run through the Xilinx tools using the implement script. It is a requirement that the core is implemented before attempting to run timing simulation.

To run a VHDL or Verilog functional simulation of the example design:

1. Set the current directory to:

```
<quickstart>/simulation/timing/
```

2. Launch the simulation script:

```
ModelSim: vsim -do simulate_mti.do
```

```
ncsim (ms-dos>) : simulate_ncsim.bat
```

```
ncsim (Linux-shell%) : ./simulate_ncsim.sh
```

The simulation script compiles the timing simulation model and the demonstration test bench, adds relevant signals to the wave window, and runs the simulation. To observe the operation of the core, inspect the simulation transcript and the waveform.

Directory and File Contents

See [Directory and File Contents in Chapter 7](#).

Implementation Scripts

Note: Present only with a Full license.

The implementation script is either a shell script(.sh) or batch file (.bat) that processes the example design through the Xilinx tool flow. It is located at:

Linux

```
<project_dir>/<component_name>/implement/implement.sh
```

Windows

```
<project_dir>/<component_name>/implement/implement.bat
```

When the CORE Generator system is run with the Full System Hardware Evaluation, or Full license types, the implement script performs these steps:

- Synthesizes the HDL example design files using XST
- Runs NGDBuild to consolidate the core netlist and the example design netlist into the NGD file containing the entire design
- Maps the design to the target technology
- Place-and-routes the design on the target device
- Performs static timing analysis on the routed design using Timing Analyzer (TRCE)
- Generates a bitstream
- Enables Netgen to run on the routed design to generate a VHDL or Verilog netlist (as appropriate for the Design Entry project setting) and timing information in the form of SDF files.

The Xilinx tool flow generates several output and report files. These are saved in the following directory which is created by the implement script:

```
<project_dir>/<component_name>/implement/results
```

Simulation Scripts

This subsection is only applicable for CORE Generator tools.

Functional Simulation

The test scripts are ModelSim macros that automate the simulation of the test bench. They are available from the following location:

```
<project_dir>/<component_name>/simulation/functional/
```

The test script performs these tasks:

- Compiles the structural UNISIM simulation model
- Compiles HDL Example Design source code
- Compiles the demonstration test bench
- Starts a simulation of the test bench
- Opens a Wave window and adds signals of interest (wave_mti.do/wave_ncsim.sv)
- Runs the simulation to completion

Timing Simulation

Note: Present only with a Full license.

The test scripts are a ModelSim or a Cadence IES macro that automates the simulation of the test bench. They are located in:

```
<project_dir>/<component_name>/simulation/timing/
```

The test script performs these tasks:

- Compiles the SIMPRIM based gate level netlist simulation model
- Compiles the demonstration test bench
- Starts a simulation of the test bench
- Opens a Wave window and adds signals of interest (wave_mti.do/wave_ncsim.sv)
- Runs the simulation to completion

Example Design Configuration

Figure 9-3 illustrates the example design configuration.

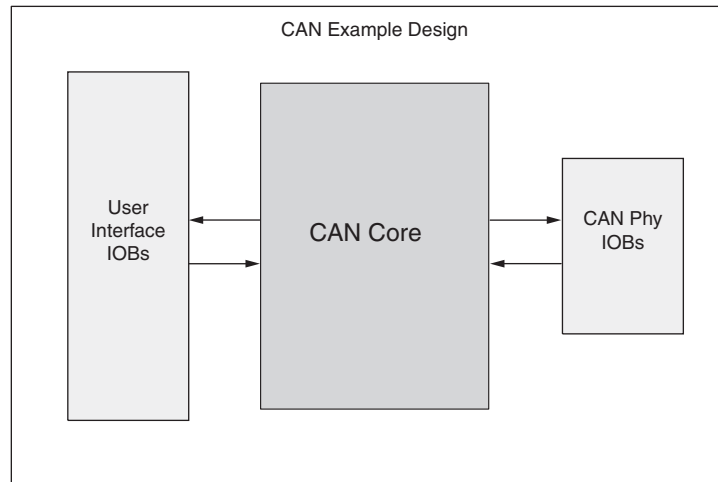


Figure 9-3: Example Design Configuration

The example design contains the following:

- An instance of the CAN core

During simulation, the CAN core is instantiated as a black box and replaced with the CORE Generator system netlist during implementation and the gate-level simulation model.

- Input and output buffers for top-level port signal

Demonstration Test Bench

Figure 9-4 illustrates the demonstration test bench.

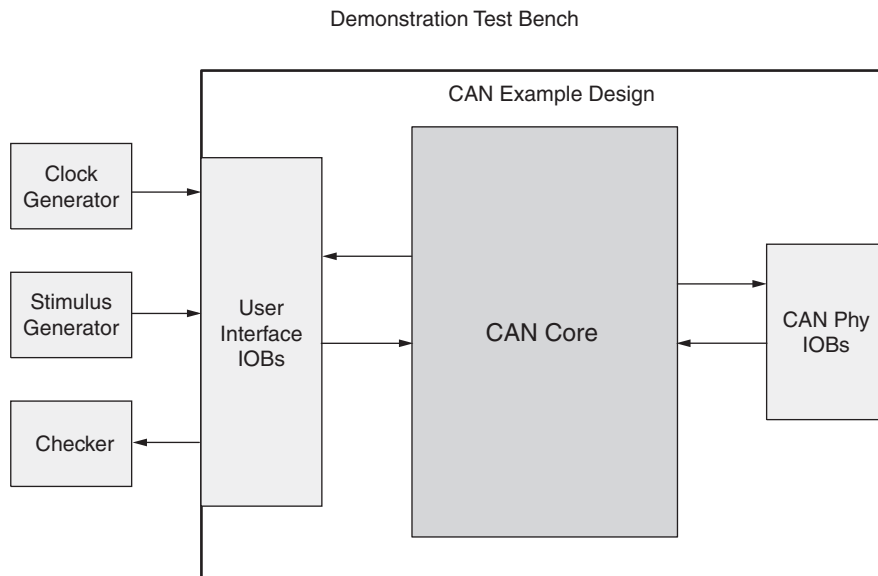


Figure 9-4: Demonstration Test Bench

Test Bench Functionality

The demonstration test bench is a straightforward VHDL or Verilog file to exercise the example design and the core itself.

The test bench consists of the following:

- Clock Generators
- Procedure to write to a Configuration Register memory location
- Procedure to read from a Configuration Register memory location
- Procedure to display the bits set in the Interrupt Status Register (ISR)

Core with No Acceptance Filtering

The demonstration test bench performs the following tasks:

- Input clock signals are generated.
- A reset is applied to the example design.
- The Baud Rate Prescaler register and Bit Timing registers are written to. These registers are read from and the values read are compared with the values written.

- The Interrupt Enable Register is written to enable interrupts for TXBFL and RXOK bits. This register is read from and the value read is compared with the value written.
- The Mode Select Register is written to select Loop Back mode of operation. This register is read from and the value read is compared with the value written.
- The Software Reset Register is written to enable CEN bit. This register is read from and the value written is compared with the value read.
- Five messages are written in sequence:
 1. The first message is written to the TXHPB and is a standard data frame.
 2. The second message is written to the TX FIFO and is a standard data frame.
 3. The third message is written to the TX FIFO and is a standard remote frame.
 4. The fourth message is written to the TX FIFO and is an extended data frame.
 5. 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 the interrupt line. When the interrupt line is asserted, the following conditions occur:

- The bits set in the ISR are displayed.
- The RX FIFO is read if the RXOK bit is set. The message received is compared with the message previously transmitted.
- The ICR is written to. This clears the bits in the ISR that are set.

With no acceptance filtering, all five messages are received in the RX FIFO.

Core with Acceptance Filtering

The demonstration test bench performs the following tasks:

- Input clock signals are generated.
- A reset is applied to the example design.
- The Baud Rate Prescaler register and Bit Timing registers are written to. These registers are read from and the values read are compared with the values written.
- The Interrupt Enable Register is written to enable interrupts for TXBFL and RXOK bits. This register is read from and the value read is compared with the value written.
- Acceptance Filter ID Register 1 and Acceptance Filter Mask Register 1 are written to. These registers are read from and the values read are compared with the values written.
- The Acceptance Filter Register is written to enable Acceptance Filter pair 1. This register is read from and the value read is compared with the value written.
- The Mode Select Register is written to select Loop Back mode. This register is read from and the value read is compared with the value written.

- The Software Reset Register is written to enable CEN bit. This register is read from and the value written is compared with the value read.
- Five messages are written in a sequence.
 1. The first message is written to the TXHPB and is a standard data frame.
 2. The second message is written to the TX FIFO and is a standard data frame.
 3. The third message is written to the TX FIFO and is a standard remote frame.
 4. The fourth message is written to the TX FIFO and is an extended data frame.
 5. 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 interrupt line to be asserted. When the interrupt line is asserted, these conditions occur:

- The bits in the ISR that are set are displayed.
- The RX FIFO is read if the RXOK bit is set. The message that is received is compared with the message that was transmitted.
- The ICR is written to. This clears the bits in the ISR that are set.
- After the fourth message is transmitted and received, the Interrupt Enable Register is written to enable interrupts for TXOK, RXOK and TXBFL. This register is read from and the value read is compared with the value written.
- The fifth message does not pass acceptance filtering. Only the TXOK bit in the ISR is set when the ISR is asserted.

Customizing the Demonstration Test Bench

This section describes the variety of demonstration test bench customization options that can be used for individual system requirements.

Changing the Data

You can change the contents of the message written to the TX FIFO / TX HPB by changing the procedure call that writes to the TX FIFO and the TX HPB memory locations. The relevant fields in the checkers must also be changed to ensure that the message read from the RX FIFO matches the message that was transmitted.

Changing the CAN Parameters

The values written to the BRPR and the BTR registers can be changed for appropriate bit timing values. The test bench operates in the Loop Back mode of operation.

Changing the Test Bench Structure

You can add messages using these steps.

1. Write the message to the TX FIFO.
2. Wait for an interrupt and process the interrupt.
3. Read the received message from the RX FIFO.

SECTION IV: APPENDICES

Verification, Compliance, and Interoperability

Migrating

Debugging

Additional Resources

Verification, Compliance, and Interoperability

Compliance Testing

Xilinx VHDL OPB CAN Core Version 1.00.a passed ISO CAN Conformance Tests.

The same compliance results are applicable for the current version of the core.

Migrating

This appendix describes migrating from older versions of the IP to the current IP release.

For information on migrating to the Vivado™ Design Suite, see the *Vivado Design Suite User Guide Product Guide* ([UG911](#))

Parameter Changes in the XCO File

Table B-1: Parameter Changes in the XCO File

component_name	can_v4_2
number_of_acceptance_filters	0
rx_fifo_depth	2
tx_fifo_depth	2

Parameter Values in the XCI File

Table B-2: Parameter Values in the XCI File

Parameter	Value
C_CAN_RX_DPTH	2
C_CAN_TX_DPTH	2
C_CAN_NUM_ACF	0
C_FAMILY	virtex7

Debugging

This appendix includes details about resources available on the Xilinx Support website and debugging tools. In addition, this appendix provides a step-by-step debugging process and a flow diagram to guide you through debugging the CAN core.

The following topics are included in this appendix:

- [Finding Help on Xilinx.com](#)
- [Debug Tools](#)
- [Simulation Debug](#)
- [Interface Debug](#)

Finding Help on Xilinx.com

To help in the design and debug process when using the CAN core, the [Xilinx Support web page](http://www.xilinx.com/support) (www.xilinx.com/support) contains key resources such as product documentation, release notes, answer records, information about known issues, and links for opening a Technical Support WebCase.

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 (www.xilinx.com/support) or by using the Xilinx Documentation Navigator.

You can download the Xilinx Documentation Navigator from the Design Tools tab on the Downloads page (www.xilinx.com/download). For more information about this tool and the features available, open the online help after installation.

Release Notes

Known issues for all cores, including the CAN core are described in the [IP Release Notes Guide \(XTP025.PDF\)](#).

Solution Centers

See the [Xilinx Solution Centers](#) for support on devices, software tools, and intellectual property at all stages of the design cycle. Topics include design assistance, advisories, and troubleshooting tips.

Known Issues

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 also 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.

Answer Records for the CAN core can be found at:

www.xilinx.com/support/answers/37994.htm

Contacting Technical Support

Xilinx provides premier technical support for customers encountering issues that require additional assistance.

To contact Xilinx Technical Support:

1. Navigate to www.xilinx.com/support.
2. Open a WebCase by selecting the [WebCase](#) link located under Support Quick Links.

When opening a WebCase, include:

- Target FPGA including package and speed grade.
- All applicable Xilinx Design Tools and simulator software versions.
- Additional files based on the specific issue might also be required. See the relevant sections in this debug guide for guidelines about which file(s) to include with the WebCase.

Debug Tools



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

Example Design

The CAN core is delivered with an example design that can be synthesized, complete with functional test benches. Information about the example design can be found in [Chapter 6, Detailed Example Design](#).

ChipScope Pro Debugging Tool

The ChipScope™ Pro debugging tool inserts logic analyzer, bus analyzer, and virtual I/O cores directly into your design. The ChipScope Pro debugging tool allows you to set trigger conditions to capture application and integrated block port signals in hardware. Captured signals can then be analyzed through the ChipScope Pro Logic Analyzer tool. For detailed information for using the ChipScope Pro debugging tool, see www.xilinx.com/tools/cspro.htm.

License Checkers

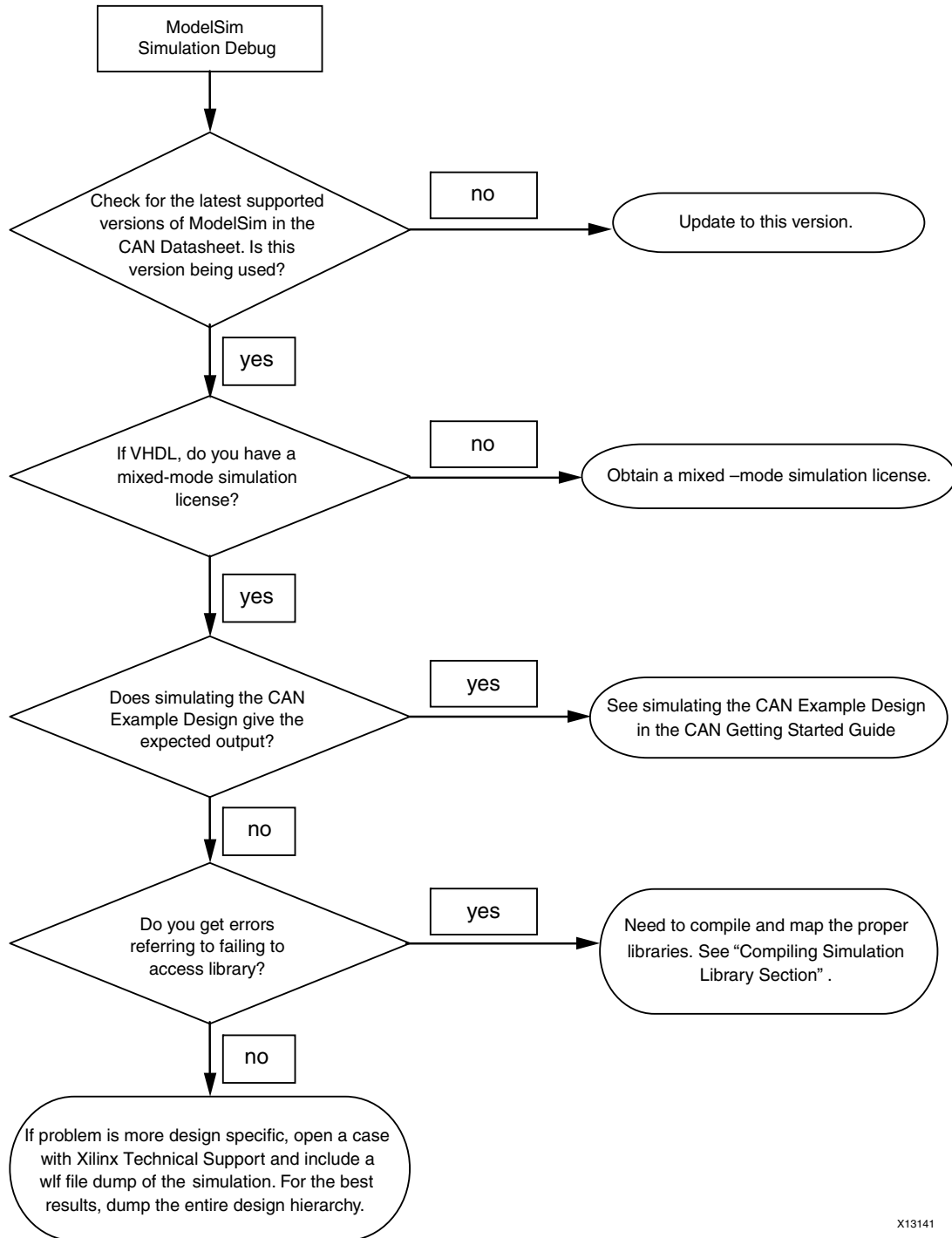
If the IP requires a license key, the key must be verified. The ISE and Vivado design tool flows have several license check points for gating licensed IP through the flow. If the license check succeeds the IP can continue generation, otherwise generation halts with error. License checkpoints are enforced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

- ISE flow: XST, NgdBuild, Bitgen
- Vivado flow: 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.*

Simulation Debug

The simulation debug flow for ModelSim is illustrated below. A similar approach can be used with other simulators.



X13141

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.

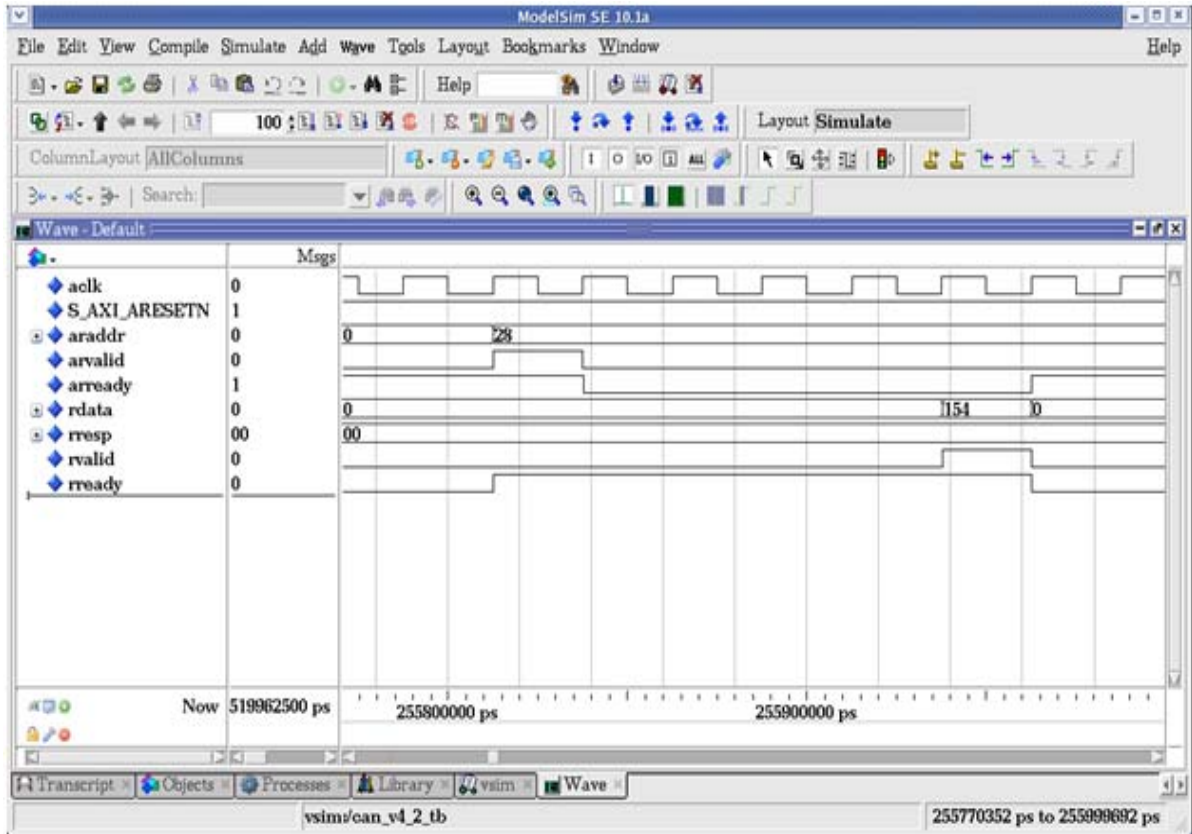


Figure C-2: Read Timing Diagram

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 been run, verify in simulation and/or a ChipScope debugging tool capture that the waveform is correct for accessing the AXI4-Lite interface.

Additional Resources

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see the Xilinx Support website at:

www.xilinx.com/support.

For a glossary of technical terms used in Xilinx documentation, see:

www.xilinx.com/company/terms.htm.

References

Unless otherwise noted, IP references are for the product documentation page. 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 documentation (www.xilinx.com/cgi-bin/docs/rdoc?v=2012.4;t=vivado+docs)
 4. *Vivado Design Suite User Guide, Implementation* ([UG911](#))
-

Technical Support

Xilinx provides technical support at www.xilinx.com/support for this LogiCORE™ IP product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support of product if implemented in devices that are not defined in the documentation, if customized beyond that allowed in the product documentation, or if changes are made to any section of the design labeled DO NOT MODIFY.

See the IP Release Notes Guide ([XTP025](#)) for more information on this core. For each core, there is a master Answer Record that contains the Release Notes and Known Issues list for the core being used. The following information is listed for each version of the core:

- New Features
- Resolved Issues
- Known Issues

Revision History

Date	Version	Revision
12/18/12	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. <ul style="list-style-type: none"> • 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.

Notice of Disclaimer

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of the Limited Warranties which can be viewed at <http://www.xilinx.com/warranty.htm>; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in Critical Applications: <http://www.xilinx.com/warranty.htm#critapps>.

© Copyright 2012 Xilinx, Inc. Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners.