MAX3108

SPI/I²C UART with 128-Word FIFOs in WLP

General Description

The MAX3108 small form factor universal asynchronous receiver-transmitter (UART) with 128 words each of receive and transmit FIFOs is controlled through a serial I²C or SPI controller interface. Auto-sleep and shutdown modes help reduce power consumption during periods of inactivity. A low 500µA (max) supply current and tiny 25-bump WLP (2.1mm x 2.1mm) package make the MAX3108 ideal for low-power portable devices. The MAX3108 operates from a supply voltage of 1.71V to 3.6V.

Baud rates up to 24Mbps make the MAX3108 suitable for today's high data rate applications. A phase-locked loop (PLL), predivider, and fractional baud-rate generator allow high-resolution baud-rate programming and minimize the dependency of baud rate on reference clock frequency.

Four GPIOs can be used as inputs, outputs, or interrupt inputs. When configured as outputs, they can be programmed to be open-drain outputs and sink up to 20mA of current.

The MAX3108 is ideal for portable and handheld devices, is available in a 25-bump (2.1mm x 2.1mm) 0.4mm pitch WLP package, and is specified over the -40°C to +85°C extended temperature range.

Applications

Portable Communication Devices
Mobile Internet Devices
Low-Power Handheld Devices
Medical Systems
Point-of-Sale Systems

IrDA is a service mark of Infrared Data Association Corporation.

Features

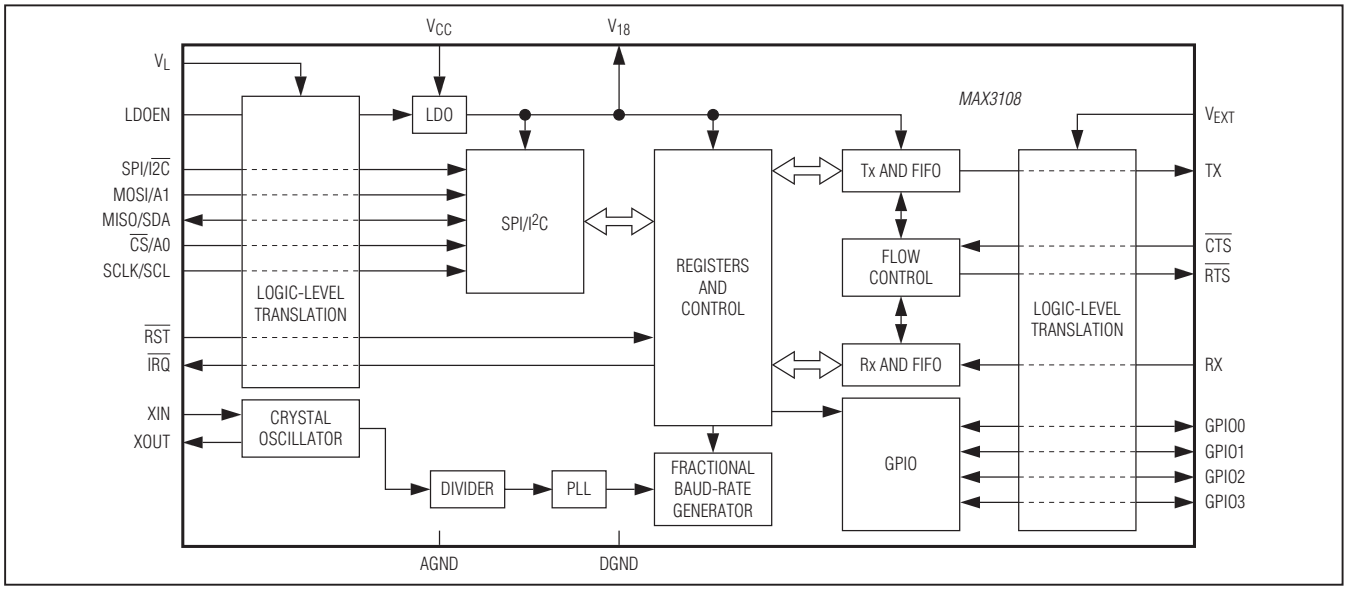
- **24Mbps (max) Baud Rate**
- **Integrated PLL and Divider**
- **1.71V to 3.6V Supply Range**
- **High-Resolution Programmable Baud Rate**
- **SPI Up to 26MHz Clock Rate**
- **Fast Mode Plus I²C Up to 1MHz**
- **Automatic $\overline{RTS}$ and $\overline{CTS}$ Hardware Flow Control**
- **Automatic XON/XOFF Software Flow Control**
- **Special Character Detection**
- **9-Bit Multidrop Mode Data Filtering**
- **SIR- and MIR-Compliant IrDA℠ Encoder/Decoder**
- **Flexible Logic Levels on the Controller and Transceiver Interfaces**
- **Four Flexible GPIOs**
- **Line Noise Indication**
- **Shutdown and Auto-Sleep Modes**
- **Low 35µA (max) V_{CC} Shutdown Current**
- **Register Compatible with the MAX3107**
- **Tiny 25-Bump WLP Package (2.1mm x 2.1mm)**

Ordering Information

PART	TEMP RANGE	PIN-PACKAGE
MAX3108EWA+T	-40°C to +85°C	25 WLP

+Denotes a lead (Pb)-free/RoHS-compliant package.

Functional Diagram



For pricing, delivery, and ordering information, please contact Maxim Direct at 1-888-629-4642, or visit Maxim's website at www.maximintegrated.com.

19-5723; Rev 2; 1/13

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

LIST OF FIGURES

LIST OF TABLES

LIST OF REGISTERS

ABSOLUTE MAXIMUM RATINGS

(Voltages referenced to AGND.)

V_L, V_{CC}, V_{EXT}, XIN ..... -0.3V to +4.0V
XOUT ..... -0.3V to (V_{CC} + 0.3V)
V_{18} ..... -0.3V to the lesser of (V_{CC} + 0.3V) and 2.0V
$\overline{RST}$, $\overline{IRQ}$, MOSI/A1, $\overline{CS}$/A0, SCLK/SCL, MISO/SDA, LDOEN, SPI/$\overline{I2C}$ ..... -0.3V to (V_L + 0.3V)
TX, RX, $\overline{RTS}$, $\overline{CTS}$, GPIO_ ..... -0.3V to (V_{EXT} + 0.3V)
DGND ..... -0.3V to +0.3V
Continuous Power Dissipation (T_A = +70°C)
WLP (derate 19.2mW/°C above +70°C) 1536mW
Operating Temperature Range ..... -40°C to +85°C
Maximum Junction Temperature ..... +150°C
Storage Temperature Range ..... -65°C to +150°C
Soldering Temperature (reflow) ..... +260°C

PACKAGE THERMAL CHARACTERISTICS (Note 1)

WLP
Junction-to-Ambient Thermal Resistance (θ_{JA}) ..... 52°C/W
Junction-to-Case Thermal Resistance (θ_{JC}) ..... 11°C/W

Note 1: Package thermal resistances were obtained using the method described in JEDEC specification JESD51-7, using a four-layer board. For detailed information on package thermal considerations, refer to **www.maximintegrated.com/thermal-tutorial**.

Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated in the operational sections of the specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

DC ELECTRICAL CHARACTERISTICS

(V_{CC} = 1.71V to 3.6V, V_L = 1.71V to 3.6V, V_{EXT} = 1.71V to 3.6V, T_A = -40°C to +85°C, unless otherwise noted. Typical values are at V_{CC} = 2.8V, V_L = 1.8V, V_{EXT} = 2.5V, T_A = +25°C.) (Notes 2, 3)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
Digital Interface Supply Voltage	V_L		1.71		3.6	V
Analog Supply Voltage	V_{CC}	Internal PLL disabled and bypassed	1.71		3.6	V
		Internal PLL enabled	2.35		3.6	
UART Interface Logic Supply Voltage	V_{EXT}		1.71		3.6	V
Logic Supply Voltage	V_{18}		1.65		1.95	V
CURRENT CONSUMPTION						
V_{CC} Supply Current	I_{CC}	1.8MHz crystal oscillator active, PLL disabled, SPI/I²C interface idle, UART interfaces idle, LDOEN = high			500	µA
		Baud rate = 1Mbps, 20MHz external clock, SPI/I²C interface idle, PLL disabled, UART in loopback mode, LDOEN = low			500	
V_{CC} Shutdown Supply Current	I_{CCSHDN}	Shutdown mode, LDOEN = low, $\overline{RST}$ = low, all inputs and outputs are idle			35	µA
V_L Shutdown or Sleep Supply Current	I_L	Shutdown or sleep mode, all inputs and outputs are idle			15	µA
V_{EXT} Shutdown Supply Current	I_{EXT}	Shutdown mode, $\overline{RST}$ = low, all inputs and outputs are idle			10	µA
V_{18} Input Power-Supply Current in Shutdown Mode	I_{18SHDN}	Shutdown mode, LDOEN = low, $\overline{RST}$ = low, all inputs and outputs are idle			50	µA

DC ELECTRICAL CHARACTERISTICS (continued)

(V_{CC} = 1.71V to 3.6V, V_L = 1.71V to 3.6V, V_{EXT} = 1.71V to 3.6V, T_A = -40°C to +85°C, unless otherwise noted. Typical values are at V_{CC} = 2.8V, V_L = 1.8V, V_{EXT} = 2.5V, T_A = +25°C.) (Notes 2, 3)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
V_{18} Input Power-Supply Current	I_{18}	Baud rate = 1Mbps, 20MHz external clock, PLL disabled, UART in loopback mode, LDOEN = low (Note 4)			2	mA
SCLK/SCL, MISO/SDA						
MISO/SDA Output Logic-Low Voltage in I²C Mode	V_{OLI2C}	Sink current = 3mA, V_L > 2V			0.4	V
		Sink current = 3mA, V_L < 2V			0.2 x V_L	
MISO/SDA Output Logic-Low Voltage in SPI Mode	V_{OLSPI}	Sink current = 2mA			0.4	V
MISO/SDA Output Logic-High Voltage in SPI Mode	V_{OHSPI}	Source current = 2mA	V_L - 0.4			V
Input Logic-Low Voltage	V_{IL}	SPI and I²C mode			0.3 x V_L	V
Input Logic-High Voltage	V_{IH}	SPI and I²C mode	0.7 x V_L			V
Input Hysteresis	V_{HYST}	SPI and I²C mode		0.05 x V_L		V
Input Leakage Current	I_{IL}	V_{IN} = 0 to V_L, SPI and I²C mode	-1		+1	µA
Input Capacitance	C_{IN}	SPI and I²C mode		5		pF
SPI/$\overline{I2C}$, $\overline{CS}$/A0, MOSI/A1 INPUTS						
Input Logic-Low Voltage	V_{IL}	SPI and I²C mode			0.3 x V_L	V
Input Logic-High Voltage	V_{IH}	SPI and I²C mode	0.7 x V_L			V
Input Hysteresis	V_{HYST}	SPI and I²C mode		50		mV
Input Leakage Current	I_{IL}	V_{IN} = 0 to V_L, SPI and I²C mode	-1		+1	µA
Input Capacitance	C_{IN}	SPI and I²C mode		5		pF
$\overline{IRQ}$ OUTPUT (OPEN DRAIN)						
Output Logic-Low Voltage	V_{OL}	Sink current = 2mA			0.4	V
Output Leakage Current	I_{OL}	$V_{\overline{IRQ}}$ = 0 to V_L, $\overline{IRQ}$ is not asserted	-1		+1	µA
LDOEN AND $\overline{RST}$ INPUTS						
Input Logic-Low Voltage	V_{IL}				0.3 x V_L	V
Input Logic-High Voltage	V_{IH}		0.7 x V_L			V
Input Hysteresis	V_{HYST}			50		mV
Input Leakage Current	I_{IL}	V_{IN} = 0 to V_L	-1		+1	µA

DC ELECTRICAL CHARACTERISTICS (continued)

(V_{CC} = 1.71V to 3.6V, V_L = 1.71V to 3.6V, V_{EXT} = 1.71V to 3.6V, T_A = -40°C to +85°C, unless otherwise noted. Typical values are at V_{CC} = 2.8V, V_L = 1.8V, V_{EXT} = 2.5V, T_A = +25°C.) (Notes 2, 3)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
UART INTERFACE						
$\overline{RTS}$, TX OUTPUTS						
Output Logic-Low Voltage	V_{OL}	Sink current = 2mA			0.4	V
Output Logic-High Voltage	V_{OH}	Source current = 2mA	0.7 x V_{EXT}			V
Input Leakage Current	I_{IL}	Output is three-stated, $V_{\overline{RTS}}$ = 0 to V_{EXT}	-1		+1	µA
Input Capacitance	C_{IN}	High-Z mode		5		pF
$\overline{CTS}$, RX INPUTS						
Input Logic-Low Voltage	V_{IL}				0.3 x V_{EXT}	V
Input Logic-High Voltage	V_{IH}		0.7 x V_{EXT}			V
Input Hysteresis	V_{HYST}			50		mV
$\overline{CTS}$ Input Leakage Current	I_{IL}	$V_{\overline{CTS}}$ = 0 to V_{EXT}	-1		+1	µA
RX Pullup Current	I_{PU}	V_{RX} = 0V, V_{EXT} = 3.6V	-7.5	-5.5	-3.5	µA
Input Capacitance	C_{IN}			5		pF
GPIO_ INPUTS/OUTPUTS						
Output Logic-Low Voltage	V_{OL}	Sink current = 20mA, push-pull or open-drain output type, V_{EXT} > 2.3V			0.45	V
		Sink current = 20mA, push-pull or open-drain output type, V_{EXT} < 2.3V			0.55	
Output Logic-High Voltage	V_{OH}	Source current = 5mA, push-pull output type	V_{EXT} - 0.4V			V
Input Logic-Low Voltage	V_{IL}	GPIO_ is configured as an input			0.4	V
Input Logic-High Voltage	V_{IH}	GPIO_ is configured as an input	2/3 x V_{EXT}			V
Pulldown Current	I_{PD}	$V_{GPIO\_}$ = V_{EXT} = 3.6V, GPIO_ is configured as an input	3.5	5.5	7.5	µA
XIN						
Input Logic-Low Voltage	V_{IL}				0.6	V
Input Logic-High Voltage	V_{IH}		1.2			V
Input Capacitance	C_{XIN}			16		pF
XOUT						
Input Capacitance	C_{XOUT}			16		pF

AC ELECTRICAL CHARACTERISTICS

(V_{CC} = 1.71V to 3.6V, V_L = 1.71V to 3.6V, V_{EXT} = 1.71V to 3.6V, T_A = -40°C to +85°C, unless otherwise noted. Typical values are at V_{CC} = 2.8V, V_L = 1.8V, V_{EXT} = 2.5V, T_A = +25°C.) (Note 2)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
External Cystal Frequency	f_{XOSC}		1		4	MHz
External Clock Frequency	f_{CLK}		0.5		35	MHz
External Clock Duty Cycle		(Note 5)	45		55	%
Baud-Rate Generator Clock Input Frequency	f_{REF}				96	MHz
I²C BUS: TIMING CHARACTERISTICS (Figure 1)						
SCL Clock Frequency	f_{SCL}	Standard mode			100	kHz
		Fast mode			400	
		Fast mode plus			1000	
Bus Free Time Between a STOP and START Condition	t_{BUF}	Standard mode	4.7			µs
		Fast mode	1.3			
		Fast mode plus	0.5			
Hold Time for START Condition and Repeated START Condition	$t_{HD:STA}$	Standard mode	4.0			µs
		Fast mode	0.6			
		Fast mode plus	0.26			
Low Period of the SCL Clock	t_{LOW}	Standard mode	4.7			µs
		Fast mode	1.3			
		Fast mode plus	0.5			
High Period of the SCL Clock	t_{HIGH}	Standard mode	4.0			µs
		Fast mode	0.6			
		Fast mode plus	0.26			
Data Hold Time	$t_{HD:DAT}$	Standard mode	0		0.9	µs
		Fast mode	0		0.9	
		Fast mode plus	0			
Data Setup Time	$t_{SU:DAT}$	Standard mode	250			ns
		Fast mode	100			
		Fast mode plus	50			
Setup Time for Repeated START Condition	$t_{SU:STA}$	Standard mode	4.7			µs
		Fast mode	0.2			
		Fast mode plus	0.26			
Rise Time of Incoming SDA and SCL Signals	t_R	Standard mode (0.3 x V_L to 0.7 x V_L) (Note 6)	20 + 0.1C_B		1000	ns
		Fast mode (0.3 x V_L to 0.7 x V_L) (Note 6)	20 + 0.1C_B		300	
		Fast mode plus			120	
Fall Time of SDA and SCL Signals	t_F	Standard mode (0.3 x V_L to 0.7 x V_L) (Note 6)	20 + 0.1C_B		1000	ns
		Fast mode (0.3 x V_L to 0.7 x V_L) (Note 6)	20 + 0.1C_B		300	
		Fast mode plus			120	

AC ELECTRICAL CHARACTERISTICS (continued)

(V_{CC} = 1.71V to 3.6V, V_L = 1.71V to 3.6V, V_{EXT} = 1.71V to 3.6V, T_A = -40°C to +85°C, unless otherwise noted. Typical values are at V_{CC} = 2.8V, V_L = 1.8V, V_{EXT} = 2.5V, T_A = +25°C.) (Note 2)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
Setup Time for STOP Condition	$t_{SU:STO}$	Standard mode	4.7			µs
		Fast mode	0.6			
		Fast mode plus	0.26			
Capacitive Load for SDA and SCL	C_B	Standard mode (Note 5)			400	pF
		Fast mode (Note 5)			400	
		Fast mode plus (Note 5)			550	
SCL and SDA I/O Capacitance	$C_{I/O}$	(Note 5)			10	pF
Pulse Width of Spike Suppressed	t_{SP}				50	ns
SPI BUS: TIMING CHARACTERISTICS (Figure 2)						
SCLK Clock Period	$t_{CH}+t_{CL}$		38.4			ns
SCLK Pulse Width High	t_{CH}		16			ns
SCLK Pulse Width Low	t_{CL}		16			ns
$\overline{CS}$ Fall to SCLK Rise Time	t_{CSS}		0			ns
MOSI Hold Time	t_{DH}		3			ns
MOSI Setup Time	t_{DS}		5			ns
Output Data Propagation Delay	t_{DO}				20	ns
MISO Rise and Fall Times	t_{FT}				10	ns
$\overline{CS}$ Hold Time	t_{CSH}		30			ns

Note 2: All units are production tested at T_A = +25°C. Specifications over temperature are guaranteed by design.
Note 3: Currents entering the IC are positive and currents exiting the IC are negative.
Note 4: When V_{18} is powered by an external voltage supply, it must have current capability above or equal to I_{18}.
Note 5: Guaranteed by design; not production tested.
Note 6: C_B is the total capacitance of either the clock or data line of the synchronous bus in pF.

Timing Diagrams

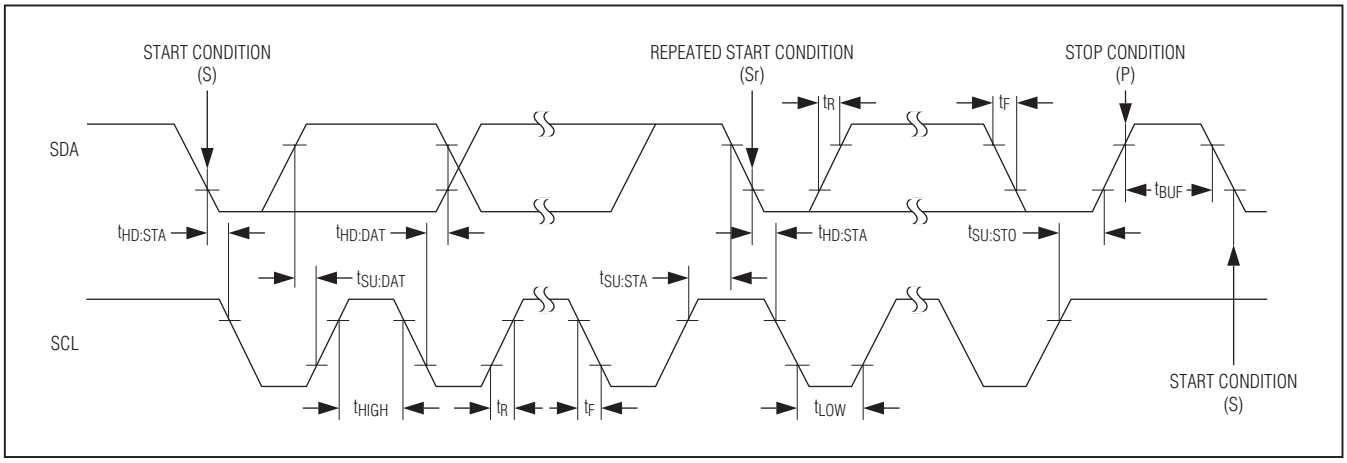


Figure 1. I2C Timing Diagram

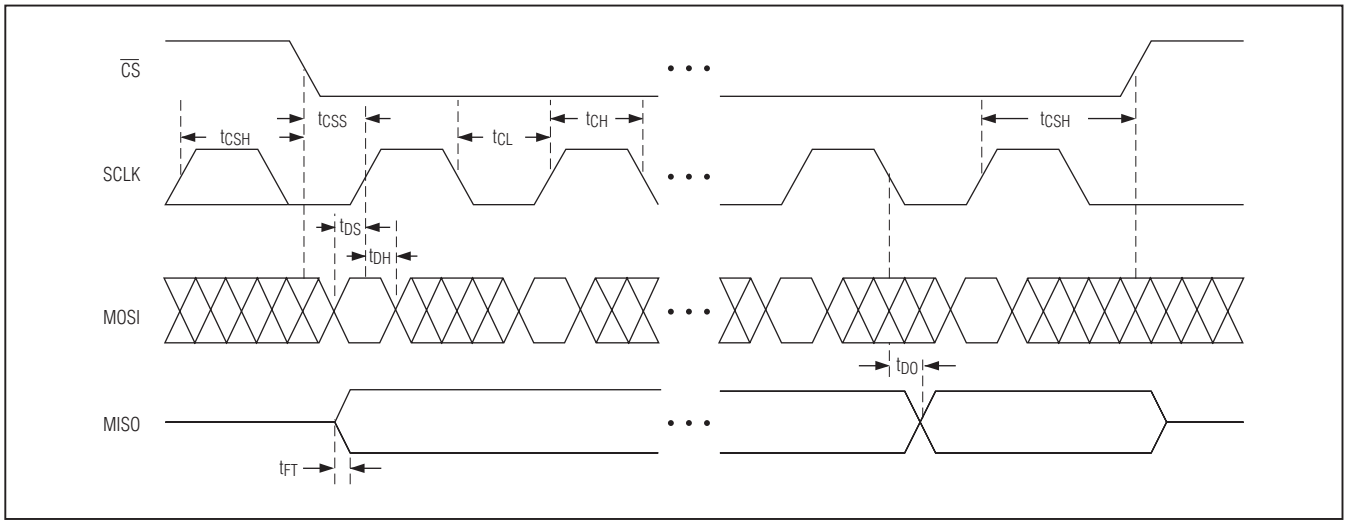
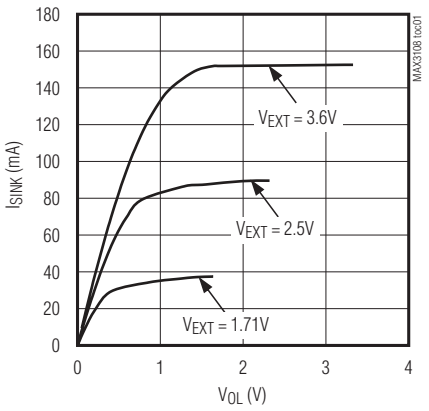


Figure 2. SPI Timing Diagram

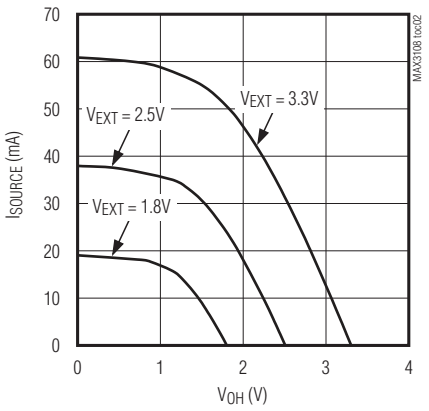
Typical Operating Characteristics

(V_{CC} = 2.5V, V_L = 2.5V, V_{EXT} = 2.5V, V_{LDOEN} = V_L, T_A = +25°C unless otherwise noted.)

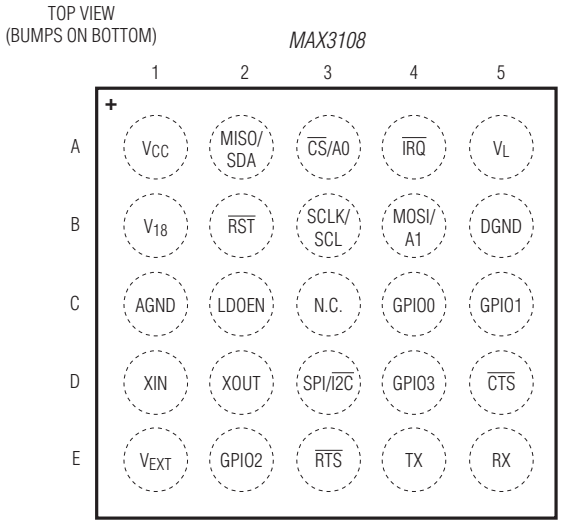
SINK CURRENT (OPEN DRAIN) vs. GPIO_ OUTPUT LOW VOLTAGE



SOURCE CURRENT (PUSH-PULL) vs. GPIO_ OUTPUT HIGH VOLTAGE



Bump Configuration



WLP
(2.1mm × 2.1mm)

Bump Description

BUMP	NAME	FUNCTION
A1	VCC	Analog Power Supply. VCC powers the PLL and internal LDO. Bypass VCC with a 0.1µF ceramic capacitor to AGND.
A2	MISO/SDA	Serial-Data Output. When SPI/$\overline{I2C}$ is high, MISO/SDA functions as the SPI master input, slave output (MISO). When SPI/$\overline{I2C}$ is low, MISO/SDA functions as the SDA, I²C serial-data input/output.
A3	$\overline{CS}$/A0	Active-Low Chip-Select and Address 0 Input. When SPI/$\overline{I2C}$ is high, $\overline{CS}$/A0 functions as the $\overline{CS}$, SPI active-low chip-select. When SPI/$\overline{I2C}$ is low, $\overline{CS}$/A0 functions as the A0 I²C device address programming input. Connect $\overline{CS}$/A0 to DGND, VL, SCL, or SDA when SPI/$\overline{I2C}$ is low.
A4	$\overline{IRQ}$	Active-Low Interrupt Open-Drain Output. $\overline{IRQ}$ is asserted when an interrupt is pending and during initial power-up.
A5	VL	Digital Interface Power Supply. VL powers the internal logic-level translators for $\overline{RST}$, $\overline{IRQ}$, MOSI/A1, $\overline{CS}$/A0, SCLK/SCL, MISO/SDA, LDOEN, and SPI/$\overline{I2C}$. Bypass VL with a 0.1µF ceramic capacitor to DGND.
B1	V18	Internal 1.8V LDO Output and 1.8V Power-Supply Input. Bypass V18 with a 1µF ceramic capacitor to DGND.
B2	$\overline{RST}$	Active-Low Reset Input. Drive $\overline{RST}$ low to force the UART into hardware reset mode. In hardware reset mode, the internal PLL is shut down and there is no clock activity.
B3	SCLK/SCL	Serial-Clock Input. When SPI/$\overline{I2C}$ is high, SCLK/SCL functions as the SCLK SPI serial-clock input (up to 26MHz). When SPI/$\overline{I2C}$ is low, SCLK/SCL functions as the SCL, I²C serial-clock input (up to 1MHz in fast mode plus).

Bump Description (continued)

BUMP	NAME	FUNCTION
B4	MOSI/A1	Serial-Data Input and Address 1 Input. When SPI/$\overline{I2C}$ is high, MOSI/A1 functions as the SPI master output-slave input (MOSI). When SPI/$\overline{I2C}$ is low, MOSI/A1 functions as the A1 I²C device address programming input. Connect MOSI/A1 to DGND, VL, SCL, or SDA when SPI/$\overline{I2C}$ is low.
B5	DGND	Digital Ground
C1	AGND	Analog Ground
C2	LDOEN	LDO Enable Input. Drive LDOEN high to enable the internal 1.8V LDO. Drive LDOEN low to disable the internal LDO. Supply V18 with an external voltage source when LDOEN is low.
C3	N.C.	Not Connected. Internally not connected.
C4	GPIO0	General-Purpose Input/Output 0. GPIO0 is user-programmable as an input or output (push-pull or open drain) or an external event-driven interrupt source. GPIO0 has a weak pulldown resistor to DGND when configured as an input.
C5	GPIO1	General-Purpose Input/Output 1. GPIO1 is user programmable as an input or output (push-pull or open drain) or an external event-driven interrupt source. GPIO1 has a weak pulldown resistor to DGND when configured as an input.
D1	XIN	Crystal/Clock Input. When using an external crystal, connect one end of the crystal to XIN and the other end to XOUT. When using an external clock source, drive XIN with the single-ended external clock.
D2	XOUT	Crystal Output. When using an external crystal, connect one end of the crystal to XOUT and the other end to XIN. When using an external clock source, leave XOUT unconnected.
D3	SPI/$\overline{I2C}$	SPI Selector Input or Active-Low I²C. Drive SPI/$\overline{I2C}$ low to enable I²C. Drive SPI/$\overline{I2C}$ high to enable SPI.
D4	GPIO3	General-Purpose Input/Output 3. GPIO3 is user-programmable as input or output (push-pull or open drain) or an external event-driven interrupt source. GPIO3 has a weak pulldown resistor to DGND when configured as an input.
D5	$\overline{CTS}$	Active-Low Clear-to-Send Input. $\overline{CTS}$ is a flow-control status input.
E1	VEXT	Transceiver Interface Power Supply. VEXT powers the internal logic-level translators for RX, TX, $\overline{RTS}$, $\overline{CTS}$, and GPIO_. Bypass VEXT with a 0.1µF ceramic capacitor to DGND.
E2	GPIO2	General-Purpose Input/Output 2. GPIO2 is user programmable as input or output (push-pull or open drain) or an external event-driven interrupt source. GPIO2 has a weak pulldown resistor to DGND when configured as an input.
E3	$\overline{RTS}$	Active-Low Request-to-Send Output. $\overline{RTS}$ can be set high or low by programming the LCR register.
E4	TX	Serial Transmitting Data Output.
E5	RX	Serial Receiving Data Input. RX has an internal weak pullup resistor to VEXT.

Detailed Description

The MAX3108 universal asynchronous receiver-transmitter (UART) bridges an SPI/MICROWIRE™ or I2C microprocessor bus to an asynchronous serial-data communication link. The MAX3108 contains an advanced UART, a fractional baud-rate generator, and four GPIOs. Eight-bit registers configure and monitor the MAX3108 and are accessed through SPI or I2C, selectable by an external pin. The registers are organized by related function as shown in the *Register Map* section.

The host controller loads data into the Transmit Hold register (**THR**) through the SPI or I2C interface. This data is automatically pushed into the transmit first-in/first-out (FIFO), formatted, and sent out at TX. The MAX3108 adds START, STOP, and parity bits to the data before transmitting at the selected baud rate. The clock configuration registers determine the baud rate, clock source, and clock frequency prescaling.

The MAX3108 receiver detects a START bit as a high-to-low transition on RX. An internal clock samples this data at 16 times the baud rate. The received data is automatically placed in the receive FIFO and can be read out by the host microcontroller through the Receive Hold register (**RHR**).

The MAX3108's register set is compatible with the MAX3107.

Register Set

The MAX3108 has a flat register structure without shadow registers. The registers are 8 bits wide. The MAX3108 registers have some similarities to the 16C550 registers.

Receive and Transmit FIFOs

The UART's receiver and transmitter each have a 128-word-deep FIFO, reducing the number of intervals that the host processor needs to dedicate for high-speed, high-volume data transfer to and from the device. As the data rates of the asynchronous RX/TX interfaces increase and get closer to those of the host controller's SPI/I2C data rates, UART management and flow-control can make up a significant portion of the host's activity. By increasing FIFO size, the host is interrupted less often and can use data block transfers to and from the FIFOs.

FIFO trigger levels can generate interrupts to the host controller, signaling that programmed FIFO fill levels have been reached. The transmitter and receiver trigger levels are programmed through the **FIFOTrgLvl** register with a resolution of eight FIFO locations. The receive FIFO trigger signals to the host either that the receive FIFO has a defined number of words waiting to be read out in a block or that a known number of vacant FIFO locations are available and ready to be filled. The transmit FIFO trigger generates an interrupt when the transmit FIFO fill level is above the programmed trigger level. The host then knows to throttle data writing to the transmit FIFO through **THR**.

The host can read out the number of words present in each of the FIFOs through the **TxFIFOLvl** and **RxFIFOLvl** registers.

The contents of the TxFIFO and RxFIFO are both cleared when the **MODE2**[1]: FIFORst bit is set high.

Transmitter Operation

Figure 3 shows the structure of the transmitter with the TxFIFO. The transmit FIFO can hold up to 128 words of data that are added by writing to the **THR** register.

The current number of words in the TxFIFO can be manually read out by the host controller through the **TxFIFOLvl** register. The transmit FIFO fill level can be

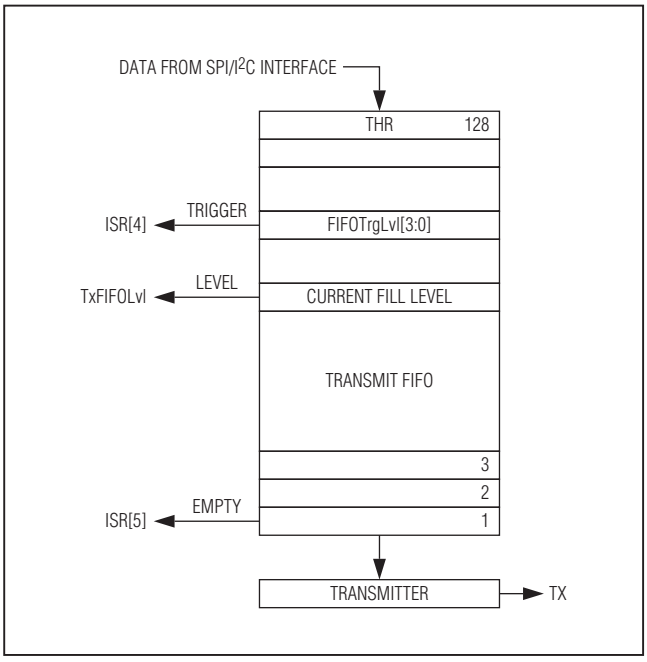


Figure 3. Transmit FIFO Signals

MICROWIRE is a trademark of National Semiconductor Corp.

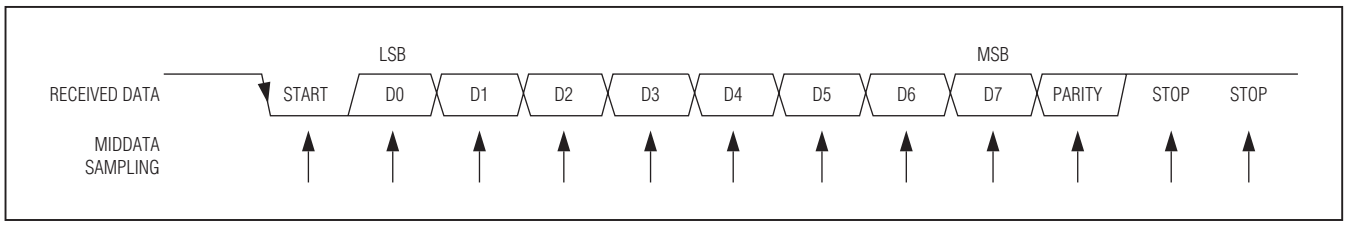


Figure 4. Receive Data Format

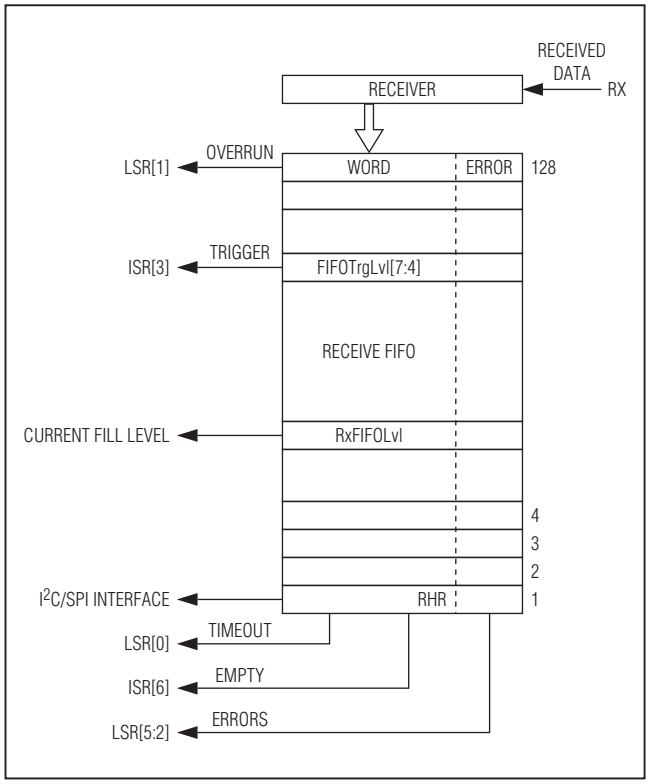


Figure 5. Receive FIFO

programmed to generate an interrupt when more than a programmed number of words are present in the TxFIFO through the **FIFOTrgLvl** register. This TxFIFO interrupt trigger level is selectable by the **FIFOTrgLvl**[3:0] bits. When the transmit FIFO fill level increases beyond the programmed trigger level, an interrupt is generated in **ISR**[4]: TxTrgInt.

An interrupt is generated in **ISR**[5]: TFifoEmptyInt when the transmit FIFO is empty. **ISR**[5] goes high when the transmitter starts transmitting the last word in the TxFIFO. An additional interrupt is generated in **STSInt**[7]: TxEmptyInt when the transmitter completes transmitting the last word.

To halt transmission, set the **MODE1**[1]: TxDisabl bit high. After TxDisabl is set, the transmitter completes the transmission of the current character and then ceases transmission. Turn the transmitter off prior to enabling auto software flow control and AutoRTS flow control.

The TX output logic can be inverted through the **IrDA**[5]: TxInv bit. Unless otherwise noted, all transmitter logic described in this data sheet assumes that TxInv is set low.

Receiver Operation

The receiver expects the format of the data at RX to be as shown in Figure 4. The quiescent logic state is logic-high and the first bit (the START bit) is logic-low. The 8-bit data word expected to be received LSB first. The receiver samples the data near the midbit instant (Figure 4). The received words and their associated errors are deposited into the receive FIFO. Errors and status information are stored for every received word (Figure 5). The host reads the data out of the receive FIFO by reading **RHR**, which comes out oldest data first. The status and error information for the word most recently read from **RHR** is located in the Line Status register (**LSR**). After a word is read out of **RHR**, **LSR** contains the status information for that word.

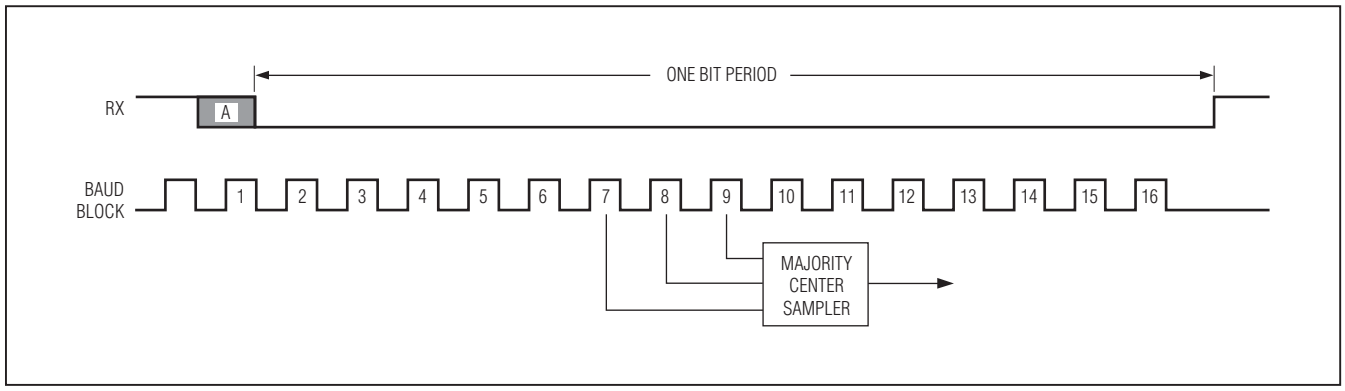


Figure 6. Midbit Sampling

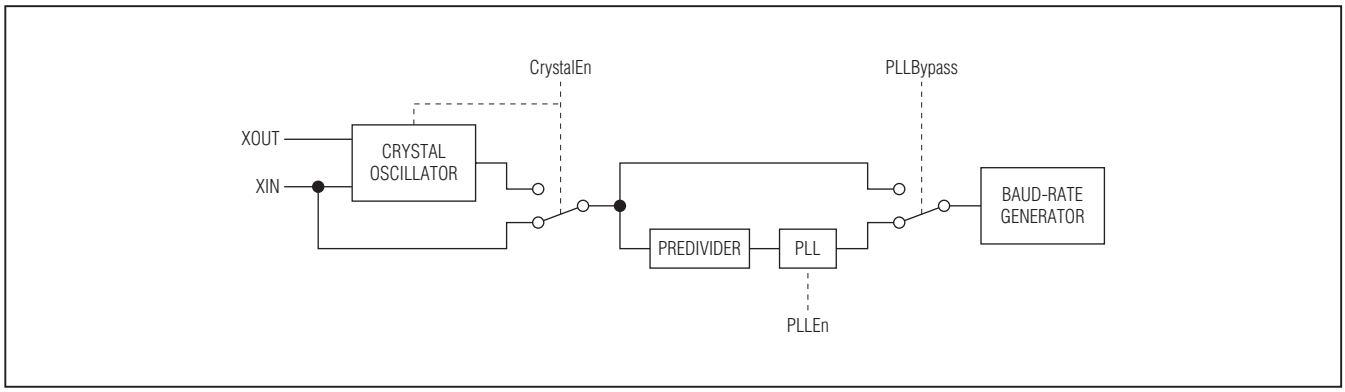


Figure 7. Clock Selection Diagram

The following three error conditions are checked for each received word: parity error, frame error, and noise on the line. Parity errors are detected by calculating either even or odd parity of the received word as programmed by register settings. Framing errors are detected when the received data frame does not match the expected frame format in length. Line noise is detected by checking the logical congruency of the three samples taken of each bit (Figure 6).

The receiver can be turned off by setting the **MODE1**[0]: RxDisabl bit high. After this bit is set high, the MAX3108 turns the receiver off immediately following the current word and does not receive any further data.

The RX input logic can be inverted by setting the **IrDA**[4]: RxInv bit high. Unless otherwise noted, all receiver logic described in this data sheet assumes that RxInv is set low.

Line Noise Indication

When operating in standard or 2x (i.e., not 4x) rate mode, the MAX3108 checks that the binary logic level of the three samples per received bit are identical. If any of the three samples per received bit have differing logic levels, then noise on the transmission line has affected the received data and it is considered to be noisy. This noise indication is reflected in the **LSR**[5]: RxNoise bit for each received byte. Parity errors are another indication of noise, but are not as sensitive.

Clock Selection

The MAX3108 can be clocked by either an external crystal or an external clock source. Figure 7 shows a simplified diagram of the clock selection circuitry. When the MAX3108 is clocked by a crystal, the **STSInt**[5]: ClkReady bit indicates when the crystal oscillator has reached steady state and the baud-rate generator is ready for stable operation.

The baud-rate clock can be routed to the $\overline{RTS}$ output by setting the **CLKSource**[7]: CLKtoRTS bit high. The clock rate is 16x the baud rate in standard operating mode, 8x the baud rate in 2x rate mode, and 4x the baud rate in 4x rate mode. If the fractional portion of the baud-rate generator is used, the clock is not regular and exhibits jitter.

Crystal Oscillator

The MAX3108 is equipped with a crystal oscillator to provide high baud-rate accuracy and low power consumption. Set the **CLKSource**[1]: CrystalEn bit high to enable and select the crystal oscillator. The on-chip crystal oscillator has integrated load capacitances of 16pF in both the XIN and XOUT pins. Connect only an external crystal or ceramic oscillator between XIN and XOUT.

External Clock Source

Connect an external single-ended clock source to XIN when not using the crystal oscillator. Leave XOUT unconnected. Set the **CLKSource**[1]: CrystalEn bit low to select external clocking.

PLL and Predivider

The internal predivider and PLL allow for compatibility with a wide range of external clock frequencies and baud rates. The PLL can be configured to multiply the input clock rate by a factor of 6, 48, 96, or 144 by the **PLLConfig**[7:6] bits. The predivider is located between the input clock and the PLL and allows division of the input clock by an integer factor between 1 and 63. This value is defined by the **PLLConfig**[5:0] bits. See the **PLLConfig** register description for more information. Use of the PLL requires V_{CC} to be higher than 2.35V.

Fractional Baud-Rate Generator

The internal fractional baud-rate generator provides a high degree of flexibility and high resolution in baud-rate programming. The baud-rate generator has a 16-bit integer divisor and a 4-bit word for the fractional divisor. The fractional baud-rate generator can be used either with the crystal oscillator or external clock source.

The integer and fractional divisors are calculated by the divisor, D:

$$D = \frac{f_{REF} \times RateMode}{16 \times BaudRate}$$

where f_{REF} is the reference frequency input to the baud-rate generator, RateMode is the rate mode multiplier (1x default), BaudRate is the desired baud rate, and D is the ideal divisor. f_{REF} must be less than 96MHz. RateMode is 1 in 1x rate mode, 2 in 2x rate mode, and 4 in 4x rate mode.

The integer divisor portion, DIV, of the divisor, D, is obtained by truncating D:

DIV(decimal) = TRUNC(D)

DIV can be a maximum of 16 bits (65,535) wide and is programmed into the two single-byte-wide registers **DIVMSB** and **DIVLSB**. The minimum allowed value for **DIVLSB** is 1.

The fractional portion of the divisor, FRACT, is a 4-bit nibble that is programmed into **BRGConfig**[3:0]. The maximum value is 15, allowing the divisor to be programmed with a resolution of 0.0625. FRACT is calculated as: FRACT = ROUND(16 x (D - DIV)).

The following is an example of how to calculate the divisor. It is based on a required baud rate of 190kbaud and a reference input frequency of 28.23MHz and 1x (default) rate mode.

The ideal divisor is calculated as:

D = 28,230,000/(16 x 190,000) = 9.286

hence DIV = 9.

FRACT = ROUND(16 x 0.286) = 5

so **DIVMSB** = 0x00, **DIVLSB** = 0x09, and **BRGConfig**[3:0] = 0x05.

The resulting actual baud rate can be calculated as:

$$BR_{ACTUAL} = \frac{f_{REF} \times RateMode}{16 \times D_{ACTUAL}}$$

For this example: D_{ACTUAL} = 9 + 5/16 = 9.3125, RateMode = 1, and

BR_{ACTUAL}= 28,230,000/(16 x 9.3125) = 189,463 baud.

Thus, the actual baud rate is within 0.28% of the ideal rate.

2x and 4x Rate Modes

The MAX3108 offers 2x and 4x rate modes in order to support higher baud rates than possible with standard operation using 16x sampling. In these modes, the reference clock rate only needs to be either 8x or 4x higher than the baud rate, respectively. In 4x rate mode, each received bit is only sampled once at the midbit instant instead of the usual three samples to determine the logic value of the received bit. This reduces the ability to detect line noise on the received data in 4x rate mode. The 2x and 4x rate modes are selectable through **BRGConfig**[5:4]. Note that IrDA encoding and decoding does not operate in 2x and 4x rate modes.

When 2x rate mode is selected, the actual baud rate is twice the rate programmed into the baud-rate generator. If 4x rate mode is enabled, the actual baud rate on the line is quadruple that of the programmed baud rate (Figure 8).

Multidrop Mode

In multidrop mode, also known as 9-bit mode, the data word length is 8 bits and a 9th bit is used for distinguishing between an address word and a data word. Multidrop mode is enabled by the **MODE2**[6]: MultiDrop bit. The MultiDrop bit takes the place of the parity bit in the data word structure. Parity checking is disabled and an interrupt is generated in **SpclCharInt**[5]: MultiDropInt when an address (9th bit is 1) is received while in multidrop mode.

It is up to the host processor to filter out the data intended for its address. Alternatively, the auto data-filtering feature can be used to automatically filter out the data not intended for the station's specific 9-bit mode address.

Auto Data Filtering in Multidrop Mode

In multidrop mode, the MAX3108 can be configured to automatically filter out data that is not meant for its address. The address is user-definable either by programming a register value or a combination of a register value and GPIO hardware inputs. Use either the entire **XOFF2** register or the **XOFF2**[7:4] bits in combination with GPIO_ inputs to define the address.

Enable multidrop mode by setting the **MODE2**[6]: MultiDrop bit high and enable auto data filtering by setting the **MODE2**[4]: SpecialChr bit high.

When using register bits in combination with GPIO_ inputs to define the address, the MSB of the address is written to the **XOFF2**[7:4] bits, while the LSBs of the address are defined by the GPIOs. To enable this address-definition method along with auto data filtering, set the **FlowCtrl**[2]: GPIAddr bit high in addition to the **MODE2**[4]: SpecialChr and **MODE2**[6]: MultiDrop bits. The GPIO_ inputs are automatically read when the **FlowCtrl**[2]: GPIAddr bit is set high, and the address is automatically updated on logic changes to any GPIO pin.

When using auto data filtering, the MAX3108 checks each received address against the programmed station address. When an address is received that matches the station's address, received data is stored in the RxFIFO. When an address is received that does not match the station's address, received data is discarded. Addresses are not stored into the FIFO but an interrupt is still generated in **SpclCharInt**[5]: MultiDropInt upon receiving an address. An additional interrupt is generated in **SpclCharInt**[3]: XOFF2Int when the station address is received.

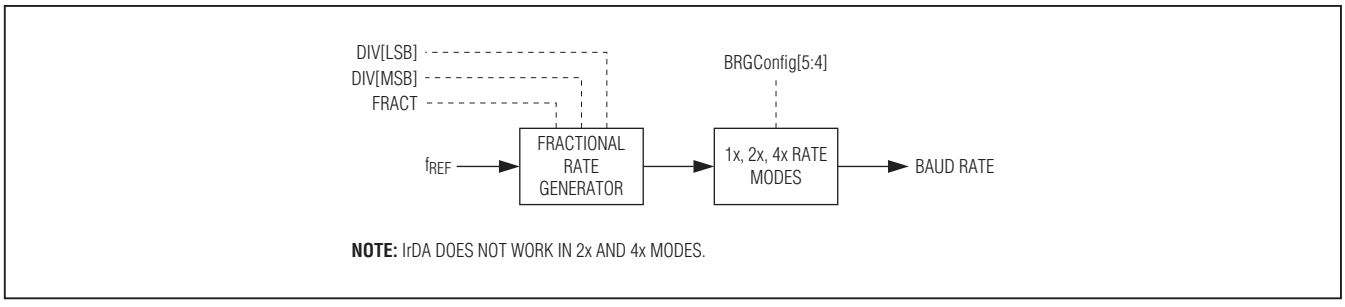


Figure 8. 2x and 4x Baud Rates

Auto Transceiver Direction Control

In some half-duplex communication systems, the transceiver's transmitter must be turned off when data is being received in order to not load the bus. This is the case in half-duplex RS-485 communication. Similarly, in full-duplex multidrop communication such as RS-485 or RS-422 V.11, only one transmitter can be enabled at any one time while the others must be disabled. The MAX3108 can be configured to automatically enable/disable a transceiver's transmitter and/or receiver at the hardware level by controlling its DE and $\overline{RE}$ pins. This feature relieves the host processor of this time-critical task.

The $\overline{RTS}$ output is used to control the transceivers' transmit-enable input and is automatically set high when the MAX3108's transmitter starts transmission. This occurs as soon as data is present in the transmit FIFO. Auto transceiver direction control is enabled by the **MODE1**[4]: TrnscvCtrl bit. Figure 9 shows a typical MAX3108 connection in an RS-485 application using the auto transceiver direction control feature.

The $\overline{RTS}$ output can be set high in advance of TX transmission by a programmable time period called the setup time (Figure 10). The setup time is programmed by the **HDplxDelay**[7:4]: Setupx bits. Similarly, the $\overline{RTS}$ output can be held high for a programmable period after the transmitter has completed transmission called the hold time. The hold time is programmed by the **HDplxDelay**[3:0]: Holdx bits.

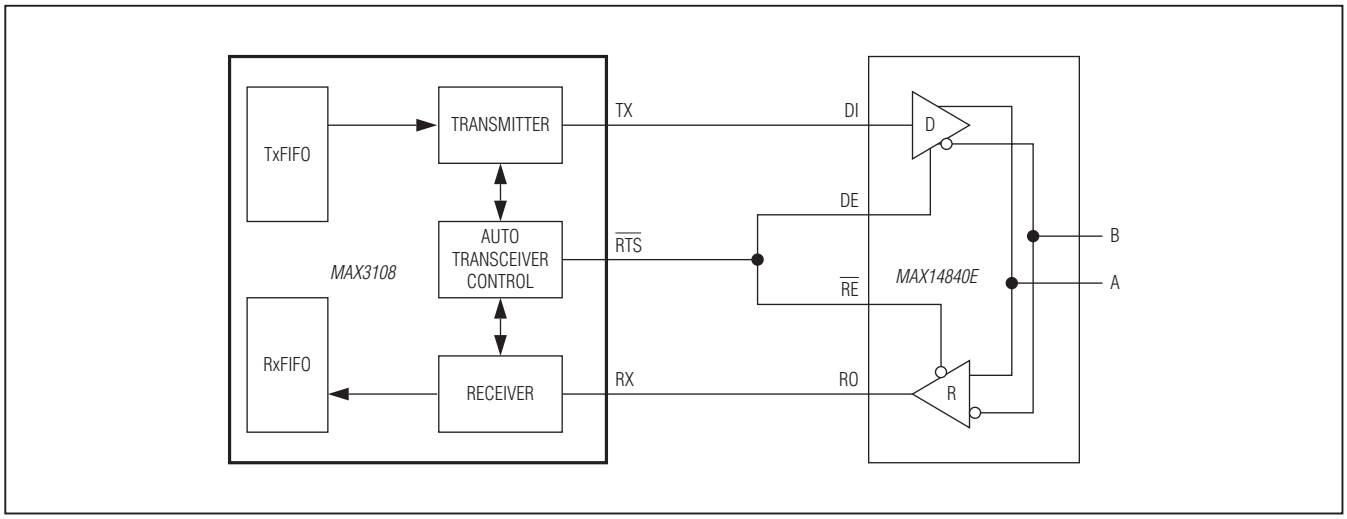


Figure 9. Auto Transceiver Direction Control

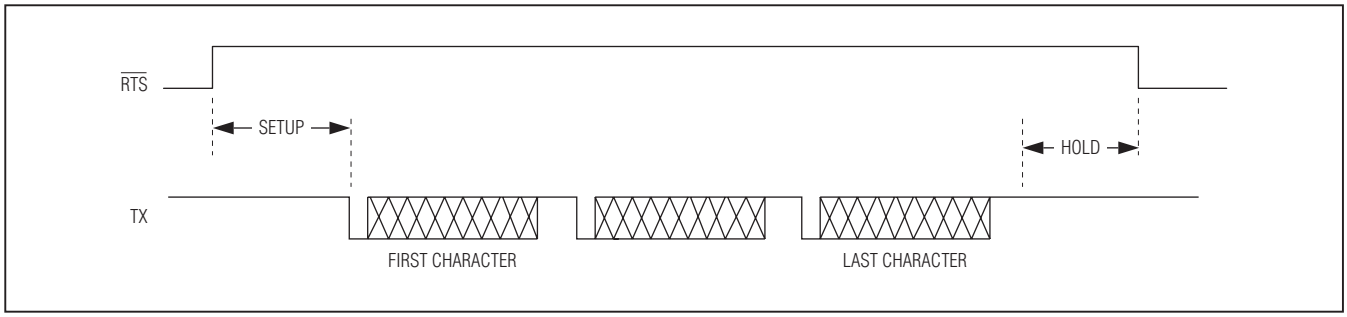


Figure 10. Setup and Hold times in Auto Transceiver Direction Control

Echo Suppression

The MAX3108 can suppress echoed data that is sometimes found in half-duplex communication networks, such as RS-485 and IrDA. If the transceiver's receiver is not turned off while the transceiver is transmitting, copies (echoes) of the transmitted data are received by the UART. The MAX3108's receiver can block the reception of this echoed data by enabling echo suppression. Figure 11 shows a typical RS-485 application using the echo suppression feature. Set the **MODE2**[7]: EchoSuprs bit high to enable echo suppression.

The MAX3108 can also block echoes with a long round trip delay by disabling the transceiver's receiver with the $\overline{RTS}$ output while the MAX3108 is transmitting. The transmitter can be configured to remain enabled after the end of the transmission for a programmable period of time called the hold time delay (Figure 12). The hold time delay is set by the **HDplxDelay**[3:0] bits. See the **HDplxDelay** description in the *Detailed Register Descriptions* section for more information.

Echo suppression can operate simultaneously with auto transceiver direction control.

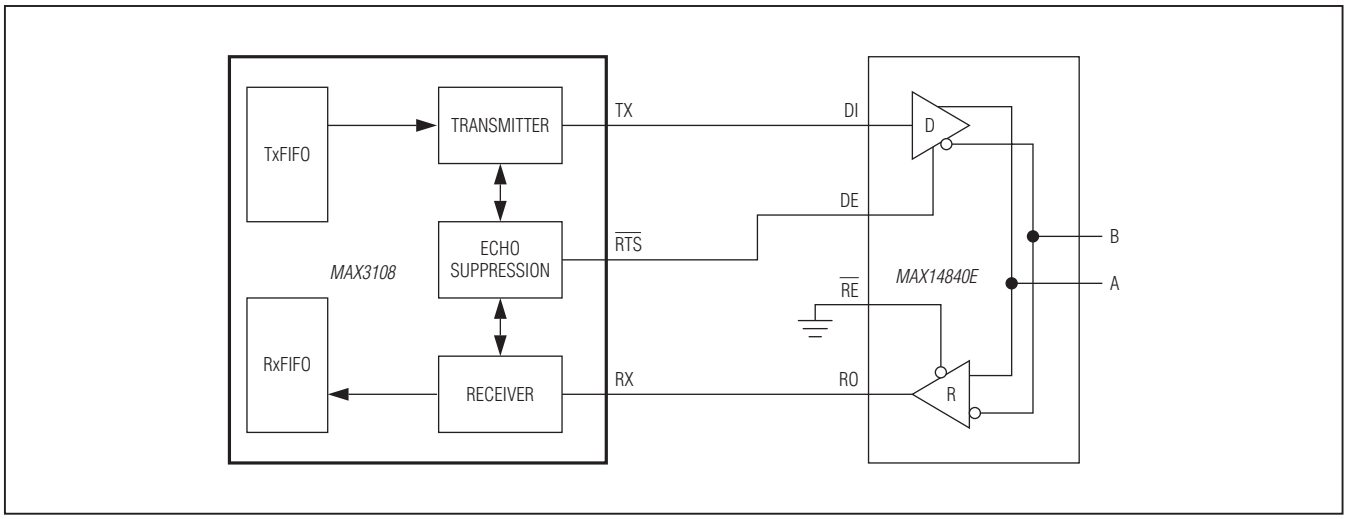


Figure 11. Half-Duplex with Echo Suppression

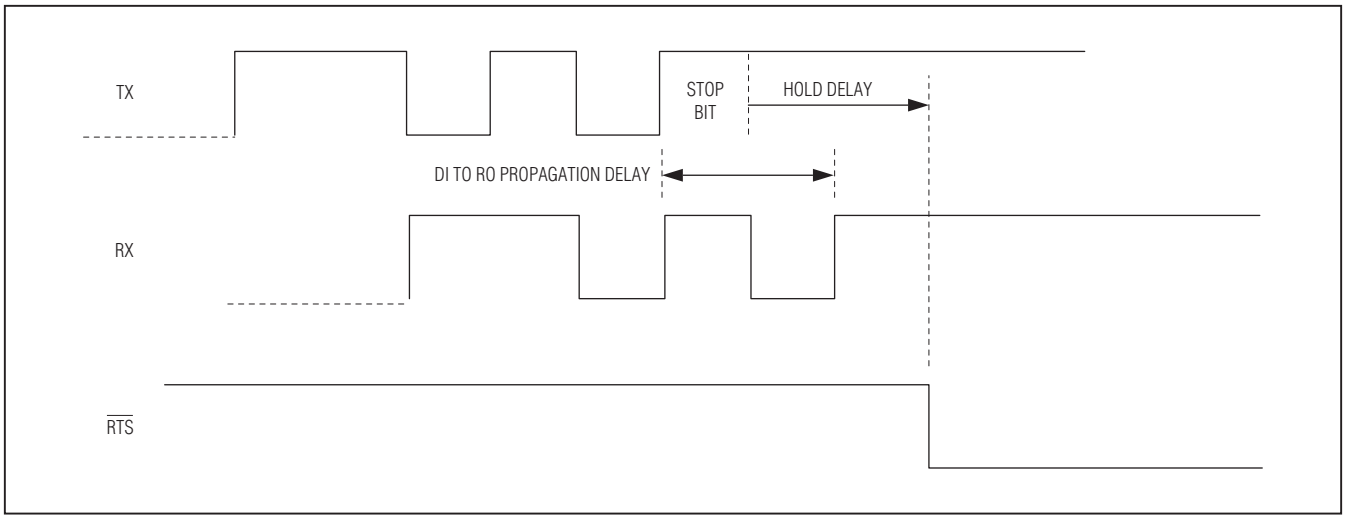


Figure 12. Echo Suppression Timing

Auto Hardware Flow Control

The MAX3108 is capable of auto hardware ($\overline{RTS}$ and $\overline{CTS}$) flow control without the need for host processor intervention. When AutoRTS control is enabled, the MAX3108 automatically controls the $\overline{RTS}$ handshake without the need for host processor intervention. AutoCTS flow control separately turns the MAX3108's transmitter on and off based on the $\overline{CTS}$ input. AutoRTS and AutoCTS flow control modes are independently enabled by the **FlowCtrl**[1:0] bits.

AutoRTS Control

AutoRTS flow control ensures that the receive FIFO does not overflow by signaling to the far-end UART to stop data transmission. The MAX3108 does this automatically by controlling the $\overline{RTS}$ output. AutoRTS flow control is enabled by setting the **FlowCtrl**[0]: AutoRTS bit high. The HALT and RESUME programmable values determine the threshold RxFIFO fill levels at which $\overline{RTS}$ is asserted and deasserted. Set the HALT and RESUME levels in the **FlowLvl** register. With differing HALT and RESUME levels, hysteresis of the RxFIFO level can be defined for $\overline{RTS}$ transitions.

When the RxFIFO is filled to a level higher than the HALT level, the MAX3108 deasserts $\overline{RTS}$ and stops the far-end UART from transmitting any additional data. $\overline{RTS}$ remains deasserted until the RxFIFO is emptied enough so that the number of words falls to below the RESUME level.

Interrupts are not generated when the HALT and RESUME levels are reached. This allows the host controller to be completely disengaged from $\overline{RTS}$ flow control management.

AutoCTS Control

When AutoCTS flow control is enabled, the UART automatically starts transmitting data when the $\overline{CTS}$ input is logic-low and stops transmitting data when $\overline{CTS}$ is logic-high. This frees the host processor from managing this time-critical flow-control task. AutoCTS flow control is enabled by setting the **FlowCtrl**[1]: AutoCTS bit high. The **ISR**[7]: CTSInt interrupt works normally during AutoCTS flow control. Set the **IRQEn**[7]: CTSIntEn bit low to disable routing of $\overline{CTS}$ interrupts to $\overline{IRQ}$ and ensure that the host does not receive interrupts from $\overline{CTS}$ transitions. If $\overline{CTS}$ transitions from low to high during transmission of a data word, the MAX3108 completes the transmission of the current word and halts transmission afterwards.

Turn the transmitter off by setting the **MODE1**[1]: TxDisabl bit high before enabling AutoCTS control.

Auto Software (XON/XOFF) Flow Control

When auto software flow control is enabled, the MAX3108 recognizes and/or sends predefined XON/XOFF characters to control the flow of data across the asynchronous serial link. The XON character signifies that there is enough room in the receive FIFO and transmission of data should continue. The XOFF character signifies that the receive FIFO is nearing overflow and that the transmission of data should stop. Auto software flow control works autonomously and does not require host intervention, similar to auto hardware flow control. To reduce the chance of receiving corrupted data that equals a single-byte XON or XOFF character, the MAX3108 allows for double-wide (16-bit) XON/XOFF characters. The XON and XOFF characters are programmed into the **XON1**, **XON2** and **XOFF1**, **XOFF2** registers.

The **FlowCtrl**[7:3] bits are used for enabling and configuring auto software flow control. An interrupt is generated in **ISR**[1]: SpCharInt whenever an XON or XOFF character is received and details are stored in the **SpclCharInt** register. Set the **IRQEn**[1]: SpclChrIEn bit low to disable routing of the interrupt to $\overline{IRQ}$.

Software flow control consists of transmit flow control and receive flow control, which operate independently of each other.

Receiver Flow Control

When auto receive flow control is enabled by the **FlowCtrl**[7:6] bits, the MAX3108 automatically controls the transmission of data by the far-end UART by sending XOFF and XON control characters. The HALT and RESUME levels determine the threshold RxFIFO fill levels at which the XOFF and XON characters are sent. HALT and RESUME are programmed in the **FlowLvl** register. With differing HALT and RESUME levels, hysteresis can be defined in the RxFIFO fill level for the receiver flow control activity.

When the RxFIFO is filled to a level higher than the HALT level, the MAX3108 sends an XOFF character to stop data transmission. An XON character is sent when the RxFIFO is emptied enough so that the number of words falls to below the RESUME level.

If double-wide (16-bit) XON/XOFF characters are selected by setting the **FlowCtrl**[7:6] bits to 11, then **XON1**/**XOFF1** are transmitted before **XON2**/**XOFF2** whenever a control character is transmitted.

Transmitter Flow Control

If auto transmit control is enabled by the **FlowCtrl**[5:4] bits, the receiver compares all received words with the XOFF and XON characters. When an XOFF character is received, the MAX3108 halts the transmitter from sending further data following any currently transmitting word. The receiver is not affected and continues receiving. Upon receiving an XON character, the transmitter restarts sending data. The received XON and XOFF characters are filtered out and are not stored into the receive FIFO. An interrupt is not generated.

If double-wide (16-bit) XON/XOFF characters are selected by setting the **FlowCtrl**[5:4] bits to 11, then a character matching **XON1**/**XOFF1** must be received before a character matching **XON2**/**XOFF2** in order to be interpreted as a control character.

Turn the transmitter off by setting the **MODE1**[1]: TxDisabl bit high before enabling software transmitter flow control.

FIFO Interrupt Triggering

Receive and transmit FIFO fill-dependent interrupts are generated if FIFO trigger levels are defined. When the number of words in the FIFOs reach or exceed a trigger level programmed in the **FIFOTrgLvl** register, an interrupt is generated in **ISR**[3] or **ISR**[4]. The interrupt trigger levels operate independently from the HALT and RESUME flow control levels in AutoRTS or auto software flow control modes.

The FIFO interrupt triggering can be used, for example, for a block data transfer. The trigger level interrupt gives the host an indication that a given block size of data is available for reading in the receive FIFO or available for transfer to the transmit FIFO. If the HALT and RESUME levels are outside of this range, then the UART continues to transmit or receive data during the block read/write operations for uninterrupted data transmission on the bus.

Low-Power Standby Modes

The MAX3108 has sleep and shutdown modes that reduce power consumption during periods of inactivity. In both sleep and shutdown modes, the UART disables specific functional blocks to reduce power consumption.

After sleep or shutdown mode is exited, the internal clock starts up and a period of time is needed for clock stabilization. The **STSInt**[5]: ClkReady bit indicates when the clocks are stable. When an external clock source is used, the ClkReady bit does not indicate clock stability.

Forced-Sleep Mode

In forced-sleep mode, all UART-related on-chip clocking is stopped. The following blocks are inactive: the crystal oscillator, the PLL, the predivider, the receiver, and the transmitter. The I2C/SPI interface and the registers remain active and the host controller can access them. To force the MAX3108 to enter sleep mode, set the **MODE1**[5]: ForcedSleep bit high. To exit forced-sleep mode, set the ForcedSleep bit low.

Auto-Sleep Mode

The MAX3108 can be configured to operate in auto-sleep mode by setting the **MODE1**[6]: AutoSleep bit high. In auto-sleep mode, the MAX3108 automatically enters sleep mode when all the following conditions are met:

- Both FIFOs are empty.
- There are no pending $\overline{IRQ}$ interrupts.
- There is no activity on any input pins for a period equal to 65,536 UART character lengths.

The same blocks are inactive when the UART is in auto-sleep mode as in forced-sleep mode.

The MAX3108 exits auto-sleep mode as soon as activity is detected on any of the GPIO_, RX, or $\overline{CTS}$ inputs.

To manually exit auto-sleep mode, set the **MODE1**[6]: AutoSleep bit low.

Shutdown Mode

Drive the $\overline{RST}$ input to logic-low to enter shutdown mode. Shutdown mode consumes the least possible amount of power. In shutdown mode, all the MAX3108 circuitry is off except for the 1.8V LDO. This includes the I2C/SPI interface, the registers, the FIFOs, and the clocking circuitry.

When the $\overline{RST}$ input transitions from low to high, the MAX3108 exits shutdown mode and a hardware reset is initiated The chip initialization is complete when the $\overline{IRQ}$ output is logic high.

The MAX3108 needs to be reprogrammed following a shutdown.

Power-Up and $\overline{IRQ}$

The $\overline{IRQ}$ output has two functions. During normal operation (the MODE1{7}. IRQSel bit is 1), $\overline{IRQ}$ operates a hardware active-low interrupt output; $\overline{IRQ}$ is asserted when an interrupt is pending. An $\overline{IRQ}$ interrupt is only possible during normal operation if at least one of the interrupt enable bits in the IRQEn register is set.

During power-up or following a hardware reset, $\overline{IRQ}$ has a different funciton; it is held low during initialization and deasserts when the the MAX3108 is ready for programming. Once $\overline{IRQ}$ goes high, the MAX3108 is ready to be programmed through the I2C/SPI interface. Set the**MODE1**{7}: IRQSel bit high to enable normal IRG interrupt operation following a power-up or reset.

In polled mode, any register with a known reset value can be polled to check whethere the MAX3108 is ready for operation. If the controller gets a valid response from the polled register, the the MAX3108 is ready for operation.

Interrupt Structure

Figure 13 shows the structure of the interrupt. There are four interrupt source registers: **ISR**, **LSR**, **STSInt**, and **SpclCharInt**. The interrupt sources are divided into top-level and low-level interrupts. The top-level interrupts typically occur more often and can be read out by the host controller directly through **ISR**. The low-level interrupts typically occur less often and their specific source can be read out by the host controller through **LSR**, **STSInt**, or **SpclCharInt**. The three LSBs of **ISR** point to the low-level interrupt registers that contain the details of the interrupt source.

Interrupt Enabling

Every interrupt bit of the four interrupt registers can be enabled or masked through an associated interrupt enable register bit. These are the **IRQEn**, **LSRIntEn**, **SpclChrIntEn**, and **STSIntEn** registers. By default, all interrupts are masked.

Interrupt Clearing

When an interrupt is pending (i.e., $\overline{IRQ}$ is asserted) and **ISR** is read, both the **ISR** bits are cleared and the $\overline{IRQ}$ output is deasserted. Low-level interrupt information does not reassert $\overline{IRQ}$ for the same interrupt, but remains stored in the low-level interrupt registers until each is separately cleared. **SpclCharInt** and **STSInt** are clear-on-read (COR). The **LSR** bits are only cleared when the source of the interrupt is removed, not when **LSR** is read.

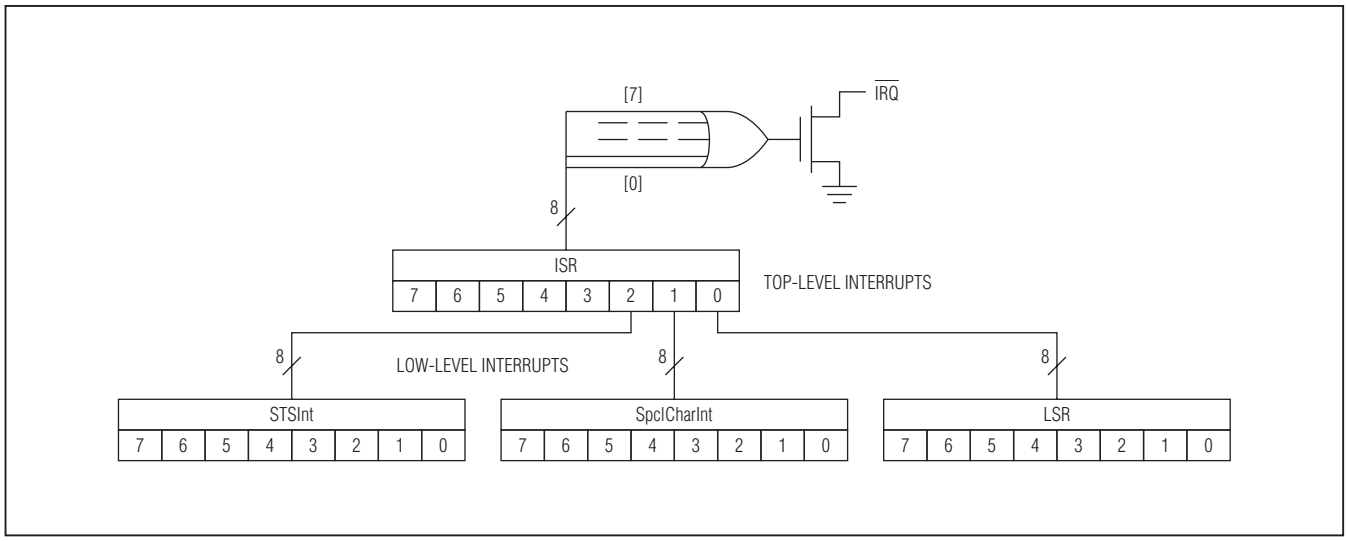


Figure 13. Simplified Interrupt Structure

Register Map

(**Note:** All default reset values are 0x00, unless otherwise noted. All registers are R/W, unless otherwise noted.)

REGISTER	ADDR	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
FIFO DATA									
RHR[1]	0x00	RData7	RData6	RData5	RData4	RData3	RData2	RData1	RData0
THR[1]	0x00	TData7	TData6	TData5	TData4	TData3	TData2	TData1	TData0
INTERRUPTS									
IRQEn	0x01	CTSIEn	RxEmtyIEn	TFifoEmtyIEn	TxTrgIEn	RxTrgIEn	STSIEn	SpChrIEn	LSRErrIEn
ISR[1, 2]	0x02	CTSInt	RFifoEmptyInt	TFifoEmptyInt	TxTrgInt	RxTrigInt	STSInt	SpCharInt	LSRErrInt
LSRIntEn	0x03	—	—	NoiseIntEn	RBreakIEn	FrameErrIEn	ParityIEn	ROverrIEn	RTimoutIEn
LSR[1]	0x04	$\overline{\text{CTS}}$bit	—	RxNoise	RxBreak	FrameErr	RxParityErr	RxOverrun	RTimeout
SpclChrIntEn	0x05	—	—	MltDrpIntEn	BREAKIntEn	XOFF2IntEn	XOFF1IntEn	XON2IntEn	XON1IntEn
SpclCharInt[1]	0x06	—	—	MultiDropInt	BREAKInt	XOFF2Int	XOFF1Int	XON2Int	XON1Int
STSIntEn	0x07	TxEmptyIntEn	SleepIntEn	ClkRdyIntEn	—	GPI3IntEn	GPI2IntEn	GPI1IntEn	GPI0IntEn
STSInt[1]	0x08	TxEmptyInt	SleepInt	ClkReady	—	GPI3Int	GPI2Int	GPI1Int	GPI0Int
UART MODES									
MODE1	0x09	—	AutoSleep	ForcedSleep	TrnscvCtrl	RTSHiZ	TXHiZ	TxDisabl	RxDisabl
MODE2	0x0A	EchoSuprs	MultiDrop	Loopback	SpecialChr	RFifoEmptyInv	RxTrgInv	FIFORst	RST
LCR[2]	0x0B	$\overline{\text{RTS}}$bit	TxBreak	ForceParity	EvenParity	ParityEn	StopBits	Length1	Length0
RxTimeOut	0x0C	TimOut7	TimOut6	TimOut5	TimOut4	TimOut3	TimOut2	TimOut1	TimOut0
HDplxDelay	0x0D	Setup3	Setup2	Setup1	Setup0	Hold3	Hold2	Hold1	Hold0
IrDA	0x0E	—	—	TxInv	RxInv	MIR	—	SIR	IrDAEn
FIFOs CONTROL									
FlowLvl	0x0F	Resume3	Resume2	Resume1	Resume0	Halt3	Halt2	Halt1	Halt0
FIFOTrgLvl[2]	0x10	RxTrig3	RxTrig2	RxTrig1	RxTrig0	TxTrig3	TxTrig2	TxTrig1	TxTrig0
TxFIFOLvl[1]	0x11	TxFL7	TxFL6	TxFL5	TxFL4	TxFL3	TxFL2	TxFL1	TxFL0
RxFIFOLvl[1]	0x12	RxFL7	RxFL6	RxFL5	RxFL4	RxFL3	RxFL2	RxFL1	RxFL0
FLOW CONTROL									
FlowCtrl	0x13	SwFlow3	SwFlow2	SwFlow1	SwFlow0	SwFlowEn	GPIAddr	AutoCTS	AutoRTS
XON1	0x14	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
XON2	0x15	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
XOFF1	0x16	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
XOFF2	0x17	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
GPIOs									
GPIOConfg	0x18	GP3OD	GP2OD	GP1OD	GP0OD	GP3Out	GP2Out	GP1Out	GP0Out
GPIOData	0x19	GPI3Dat	GPI2Dat	GPI1Dat	GPI0Dat	GPO3Dat	GPO2Dat	GPO1Dat	GPO0Dat
CLOCK CONFIGURATION									
PLLConfig[2]	0x1A	PLLFactor1	PLLFactor0	PreDiv5	PreDiv4	PreDiv3	PreDiv2	PreDiv1	PreDiv0
BRGConfig	0x1B	—	—	4xMode	2xMode	FRACT3	FRACT2	FRACT1	FRACT0
DIVLSB[2]	0x1C	Div7	Div6	Div5	Div4	Div3	Div2	Div1	Div0
DIVMSB	0x1D	Div15	Div14	Div13	Div12	Div11	Div10	Div9	Div8
CLKSource[2]	0x1E	CLKtoRTS	—	—	—	PLLBypass	PLLEn	CystalEn	—

1 *Denotes nonread/write mode: RHR = R, THR = W, ISR = COR, LSR = R, SpclCharInt = COR, STSInt = R/COR, TxFIFOLvl = R, RxFIFOLvl = R.*

2 *Denotes nonzero default reset value: ISR = 0x60, LCR = 0x05, FIFOTrgLvl = 0xFF, PLLConfig = 0x01, DIVLSB = 0x01, CLKSource = 0x18.*

Detailed Register Descriptions

The MAX3108 has a flat register structure that does not have shadow registers, which makes programming simple and efficient. All registers are 8 bits wide.

Receive Hold Register (RHR)

ADDRESS:	0x00							
MODE:	R							
BIT	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
NAME	RData7	RData6	RData5	RData4	RData3	RData2	RData1	RData0
RESET	0	0	0	0	0	0	0	0

Bits 7–0: RDatax

The **RHR** is the bottom of the receive FIFO and is the register used for reading data out of the receive FIFO. It contains the oldest (first received) character in the receive FIFO. **RHR**[0] is the LSB of the character received at the RX input. It is the first data bit of the serial-data word received by the receiver. Reading **RHR** removes the read word from the receive FIFO, clearing space for more data to be received.

Transmit Hold Register (THR)

ADDRESS:	0x00							
MODE:	W							
BIT	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
NAME	TData7	TData6	TData5	TData4	TData3	TData2	TData1	TData0
RESET	0	0	0	0	0	0	0	0

Bits 7–0: TDatax

The **THR** is the register that the host controller writes data to for subsequent UART transmission. This data is deposited in the transmit FIFO. **THR**[0] is the LSB. It is the first data bit of the serial-data word that the transmitter sends out, immediately after the START bit.

IRQ Enable Register (IRQEn)

ADDRESS:	0x01							
MODE:	R/W							
BIT	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
NAME	CTSIEn	RxEmtyIEn	TFifoEmtyIEn	TxTrgIEn	RxTrgIEn	STSIEn	SpChrIEn	LSRErrIEn
RESET	0	0	0	0	0	0	0	0

The **IRQEn** register is used to enable the $\overline{IRQ}$ physical interrupt. Any of the eight **ISR** interrupt sources can be enabled to generate an interrupt on $\overline{IRQ}$. The **IRQEn** bits only influence the $\overline{IRQ}$ output and do not have any effect on the **ISR** contents or behavior. Every one of the **IRQEn** bits operates on a corresponding ISR bit.

Bit 7: CTSIEn

The CTSIEn bit enables $\overline{IRQ}$ interrupt generation when the CTSInt interrupt is set in **ISR**[7]. Set CTSIEn low to disable $\overline{IRQ}$ generation from CTSInt.

Bit 6: RxEmtyIEn

The RxEmtyIEn bit enables $\overline{IRQ}$ interrupt generation when the RFifoEmptyInt interrupt is set in **ISR**[6]. Set RxEmtyIEn low to disable $\overline{IRQ}$ generation from RFifoEmptyInt.

Bit 5: TFifoEmtyIEn

The TFifoEmtyIEn bit enables $\overline{IRQ}$ interrupt generation when the TFifoEmptyInt interrupt is set in **ISR**[5]. Set TFifoEmtyIEn low to disable $\overline{IRQ}$ generation from TFifoEmptyInt.

Bit 4: TxTrgIEn

The TxTrgIEn bit enables $\overline{IRQ}$ interrupt generation when the TxTrgInt interrupt is set in **ISR**[4]. Set TxTrgIEn low to disable $\overline{IRQ}$ generation from TxTrgInt.

Bit 3: RxTrgIEn

The RxTrgIEn bit enables $\overline{IRQ}$ interrupt generation when the RxTrigInt interrupt is set in **ISR**[3]. Set RxTrgIEn low to disable $\overline{IRQ}$ generation from RxTrigInt.

Bit 2: STSIEn

The STSIEn bit enables $\overline{IRQ}$ interrupt generation when the STSInt interrupt is set in **ISR**[2]. Set STSIEn low to disable $\overline{IRQ}$ generation from STSInt.

Bit 1: SpChrIEn

The SpChrIEn bit enables $\overline{IRQ}$ interrupt generation when the SpCharInt interrupt is set in **ISR**[1]. Set SpclChrIEn low to disable $\overline{IRQ}$ generation from SpCharInt.

Bit 0: LSRErrIEn

The LSRErrIEn bit enables $\overline{IRQ}$ interrupt generation when the LSRErrInt interrupt is set in **ISR**[0]. Set LSRErrIEn low to disable $\overline{IRQ}$ generation from LSRErrInt.

Interrupt Status Register (ISR)

ADDRESS:	0x02							
MODE:		COR						
BIT	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
NAME	CTSInt	RFifoEmptyInt	TFifoEmptyInt	TxTrgInt	RxTrigInt	STSInt	SpCharInt	LSRErrInt
RESET	0	1	1	0	0	0	0	0

The Interrupt Status register provides an overview of all interrupts generated by the MAX3108. Both the interrupt bits and any pending interrupts on $\overline{IRQ}$ are cleared after reading **ISR**. When the MAX3108 is operated in polled mode, **ISR** can be polled to establish the UART's status. In interrupt-driven mode, $\overline{IRQ}$ interrupts are enabled by the appropriate **IRQEn** bits. The **ISR** contents either give direct information on the cause for the interrupt or point to other registers that contain more detailed information.

Bit 7: CTSInt

The CTSInt interrupt is generated when a logic state transition occurs at the $\overline{CTS}$ input. CTSInt is cleared after **ISR** is read. The current logic state of the $\overline{CTS}$ input can be read out through the **LSR**[7]: $\overline{CTS}$bit bit.

Bit 6: RFifoEmptyInt

The RFifoEmptyInt interrupt is generated when the receive FIFO is empty. RFifoEmptyInt is cleared after **ISR** is read. Its meaning can be inverted by the **MODE2**[3]: RFifoEmptyInv bit.

Bit 5: TFifoEmptyInt

The TFifoEmptyInt interrupt is generated when the transmit FIFO is empty and the transmitter is transmitting the last character. Use **STSInt**[7]: TxEmptyInt to determine when the last character has completed transmission. TFifoEmptyInt is cleared after **ISR** is read.

Bit 4: TxTrgInt

The TxTrgInt interrupt is generated when the number of characters in the transmit FIFO is equal to or greater than the transmit FIFO trigger level defined in **FIFOTrgLvl**[3:0]. TxTrgInt is cleared when the transmit FIFO level falls below the trigger level or after **ISR** is read. TxTrgInt can be used as a warning that the transmit FIFO is nearing overflow.

Bit 3: RxTrigInt

The RxTrigInt interrupt is generated when the receive FIFO fill level reaches the receive FIFO trigger level defined in **FIFOTrgLvl**[7:4]. RxTrigInt can be used as an indication that the receive FIFO is nearing overrun. It can also be used to report that a known number of words are available that can be read out in one block. The meaning of RxTrigInt can be inverted by the **MODE2**[2]: RxTrgInv bit. RxTrigInt is cleared after **ISR** is read.

Bit 2: STSInt

The STSInt interrupt is generated when any interrupt in the **STSInt** register that is enabled by a **STSIntEn** bit is high. STSInt is cleared after **ISR** is read, but the interrupt in **STSInt** that caused this interrupt remains set. See the **STSInt** register description for details about this interrupt.

Bit 1: SpCharInt

The SpCharInt interrupt is generated when a special character is received, a line break is detected, or an address character is received in multidrop mode. SpCharInt is cleared after **ISR** is read, but the interrupt in **SpclCharInt** that caused this interrupt remains set. See the **SpclCharInt** register description for details about this interrupt.

Bit 0: LSRErrInt

The LSRErrInt interrupt is generated when any interrupts in **LSR** that are enabled by corresponding bits in **LSRIntEn** are set. This bit is cleared after **ISR** is read. See the **LSR** register description for details about this interrupt.

Line Status Interrupt Enable Register (LSRIntEn)

ADDRESS:	0x03							
MODE:	R/W							
BIT	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
NAME	—	—	NoiseIntEn	RBreakIEn	FrameErrIEn	ParityIEn	ROverrIEn	RTimoutIEn
RESET	0	0	0	0	0	0	0	0

LSRIntEn allows routing of **LSR** interrupts to **ISR**[0]. The **LSRIntEn** bits only influence the **ISR**[0]: LSRErrInt bit and do not have any effect on the **LSR** contents or behavior. Bits 5 to 0 of the **LSRIntEn** register operate on a corresponding **LSR** bit, while bits 7 and 6 are not used.

Bits 7 and 6: No Function

Bit 5: NoiseIntEn

Set the NoiseIntEn bit high to enable routing the **LSR**[5]: RxNoise interrupt to **ISR**[0]. If NoiseIntEn is set low, RxNoise is not routed to **ISR**[0].

Bit 4: RBreakIEn

Set the RBreakIEn bit high to enable routing the **LSR**[4]: RxBreak interrupt to **ISR**[0]. If RBreakIEn is set low, RxBreak is not routed to **ISR**[0].

Bit 3: FrameErrIEn

Set the FrameErrIEn bit high to enable routing the **LSR**[3]: FrameErr interrupt to **ISR**[0]. If FrameErrIEn is set low, FrameErr is not routed to **ISR**[0].

Bit 2: ParityIEn

Set the ParityIEn bit high to enable routing the **LSR**[2]: RxParityErr interrupt to **ISR**[0]. If ParityIEn is set low, RxParityErr is not routed to **ISR**[0].

Bit 1: ROverrIEn

Set the ROverrIEn bit high to enable routing the **LSR**[1]: RxOverrun interrupt to **ISR**[0]. If ROverrIEn is set low, RxOverrun is not routed to **ISR**[0].

Bit 0: RTimoutIEn

Set the RTimoutIEn bit high to enable routing the **LSR**[0]: RTimeout interrupt to **ISR**[0]. If RTimoutIEn is set low, RTimeout is not routed to **ISR**[0].

Line Status Register (LSR)

ADDRESS:	0x04							
MODE:	R							
BIT	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
NAME	$\overline{\text{CTS}}$bit	—	RxNoise	RxBreak	FrameErr	RxParityErr	RxOverrun	RTimeout
RESET	X	0	0	0	0	0	0	0

LSR contains all error information related to the word most recently read out from the RxFIFO through **RHR**. The **LSR** bits are not cleared after **LSR** is read; these bits stay set until the next character is read out of **RHR**, with the exception of **LSR**[1], which is cleared by reading either **RHR** or **LSR**. **LSR** also contains the current logic state of the $\overline{\text{CTS}}$ input.

Bit 7: $\overline{\text{CTS}}$bit

The $\overline{\text{CTS}}$bit bit reflects the current logic state of the $\overline{\text{CTS}}$ input. This bit is cleared when the $\overline{\text{CTS}}$ input is low and set when it is high. Following a power-up or reset, the logic state of $\overline{\text{CTS}}$bit depends on the state of the $\overline{\text{CTS}}$ input.

Bit 6: No Function

Bit 5: RxNoise

If noise is detected on the RX input during reception of a character, the RxNoise interrupt is generated for that character. **LSR**[5] corresponds to the character most recently read from **RHR**. RxNoise is cleared after the character following the "noisy character" is read out from **RHR**. RxNoise generates an interrupt in **ISR**[0] if enabled by **LSRIntEn**[5].

Bit 4: RxBreak

If a line break (RX input low for a period longer than the programmed character duration) is detected, a break character is put in the RxFIFO and the RxBreak interrupt is generated for this character. A break character is represented by an all-zeros data character. The RxBreak interrupt distinguishes a regular character with all zeros from a break character. **LSR**[4] corresponds to the current character most recently read from **RHR**. RxBreak is cleared after the character following the break character is read out from **RHR**. RxBreak generates an interrupt in **ISR**[0] if enabled by **LSRIntEn**[4].

Bit 3: FrameErr

The FrameErr interrupt is generated when the received data frame does not match the expected frame format in length. A frame error is related to errors in expected STOP bits. **LSR**[3] corresponds to the frame error of the character most recently read from **RHR**. FrameErr is cleared after the character following the affected character is read out from **RHR**. FrameErr generates an interrupt in **ISR**[0] if enabled by **LSRIntEn**[3].

Bit 2: RxParityErr

The RxParityErr interrupt is generated when the parity computed on the character being received does not match the received character's parity bit. **LSR**[2] indicates a parity error for the character most recently read from **RHR**. RxParityErr is cleared when the character following the affected character is read out from **RHR**.

In 9-bit multidrop mode (**MODE2**[6] is logic 1) the receiver does not check parity and the 9th bit (address/data) is stored in **LSR**[2].

RxParityErr generates an interrupt in **ISR**[0] if enabled by **LSRIntEn**[2].

Bit 1: RxOverrun

The RxOverrun interrupt is generated when the receive FIFO is full and additional data is received that does not fit into the receive FIFO. The receive FIFO retains the data that it already contains and discards all new data. RxOverrun is cleared after **LSR** is read or the RxFIFO level falls below its maximum. RxOverrun generates an interrupt in **ISR**[0] if enabled by **LSRIntEn**[1].

Bit 0: RTimeout

The RTimeout interrupt indicates that stale data is present in the receive FIFO. RTimeout is set when all of the characters in the RxFIFO have been present for at least as long as the period programmed into the **RxTimeOut** register.

The timeout counter restarts whenever **RHR** is read or a new character is received by the RxFIFO. If the value in **RxTimeOut** is zero, RTimeout is disabled. RTimeout is cleared after a word is read out of the RxFIFO or a new word is received. RTimeout generates an interrupt in **ISR**[0] if enabled by **LSRIntEn**[0].

Special Character Interrupt Enable Register (SpclChrIntEn)

ADDRESS:	**0x05**							
MODE:	**R/W**							
BIT	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
NAME	—	—	MltDrpIntEn	BREAKIntEn	XOFF2IntEn	XOFF1IntEn	XON2IntEn	XON1IntEn
RESET	0	0	0	0	0	0	0	0

SpclChrIntEn allows routing of **SpclCharInt** interrupts to **ISR**[1]. The **SpclChrIntEn** bits only influence the **ISR**[1]: SpCharInt bit and do not have any effect on the **SpclCharInt** contents or behavior.

Bits 7 and 6: No Function

Bit 5: MltDrpIntEn

Set the MltDrpIntEn bit high to enable routing the **SpclCharInt**[5]: MultiDropInt interrupt to **ISR**[1]. If MltDrpIntEn is set low, MultiDropInt is not routed to **ISR**[1].

Bit 4: BREAKIntEn

Set the BREAKIntEn bit high to enable routing the **SpclCharInt**[4]: BREAKInt interrupt to **ISR**[1]. If BREAKIntEn is set low, BREAKInt is not routed to **ISR**[1].

Bit 3: XOFF2IntEn

Set the XOFF2IntEn bit high to enable routing the **SpclCharInt**[3]: XOFF2Int interrupt to **ISR**[1]. If XOFF2IntEn is set low, XOFF2Int is not routed to **ISR**[1].

Bit 2: XOFF1IntEn

Set the XOFF1IntEn bit high to enable routing the **SpclCharInt**[2]: XOFF1Int interrupt to **ISR**[1]. If XOFF1IntEn is set low, XOFF1Int is not routed to **ISR**[1].

Bit 1: XON2IntEn

Set the XON2IntEn bit high to enable routing the **SpclCharInt**[1]: XON2Int interrupt to **ISR**[1]. If XON2IntEn is set low, XON2Int is not routed to **ISR**[1].

Bit 0: XON1IntEn

Set the XON1IntEn bit high to enable routing the **SpclCharInt**[0]: XON1Int interrupt to **ISR**[1]. If XON1IntEn is set low, XON1Int is not routed to **ISR**[1].

Special Character Interrupt Register (SpclCharInt)

ADDRESS:	0x06							
MODE:	COR							
BIT	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
NAME	—	—	MultiDropInt	BREAKInt	XOFF2Int	XOFF1Int	XON2Int	XON1Int
RESET	0	0	0	0	0	0	0	0

SpclCharInt contains interrupts that are generated when a special character is received, an address is received in multidrop mode, or a line break occurs.

Bits 7 and 6: No Function

Bit 5: MultiDropInt

The MultiDropInt interrupt is generated when the MAX3108 receives an address character in 9-bit multidrop mode, enabled in **MODE2**[6]. MultiDropInt is cleared after **SpclCharInt** is read. MultiDropInt generates an interrupt in **ISR**[1] if enabled by **SpclChrIntEn**[5].

Bit 4: BREAKInt

The BREAKInt interrupt is generated when a line break (RX low for longer than one character length) is detected by the receiver. BREAKInt is cleared after **SpclCharInt** is read. BREAKInt generates an interrupt in **ISR**[1] if enabled by **SpclChrIntEn**[4].

Bit 3: XOFF2Int

The XOFF2Int interrupt is generated when both an XOFF2 special character is received and special character detection is enabled by **MODE2**[4]. XOFF2Int is cleared after **SpclCharInt** is read. XOFF2Int generates an interrupt in **ISR**[1] if enabled by **SpclChrIntEn**[3].

Bit 2: XOFF1Int

The XOFF1Int interrupt is generated when both an XOFF1 special character is received and special character detection is enabled by **MODE2**[4]. XOFF1Int is cleared after **SpclCharInt** is read. XOFF1Int generates an interrupt in **ISR**[1] if enabled by **SpclChrIntEn**[2].

Bit 1: XON2Int

The XON2Int interrupt is generated when both an XON2 special character is received and special character detection is enabled by **MODE2**[4]. XON2Int is cleared after **SpclCharInt** is read. XON2Int generates an interrupt in **ISR**[1] if enabled by **SpclChrIntEn**[1].

Bit 0: XON1Int

The XON1Int interrupt is generated when both an XON1 special character is received and special character detection is enabled by **MODE2**[4]. XON1Int is cleared after **SpclCharInt** is read. XON1Int generates an interrupt in **ISR**[1] if enabled by **SpclChrIntEn**[0].

STS Interrupt Enable Register (STSIntEn)

ADDRESS:	0x07							
MODE:	R/W							
BIT	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
NAME	TxEmptyIntEn	SleepIntEn	ClkRdyIntEn	—	GPI3IntEn	GPI2IntEn	GPI1IntEn	GPI0IntEn
RESET	0	0	0	0	0	0	0	0

STSIntEn allows routing of **STSInt** interrupts to **ISR**[2]. The **STSIntEn** bits only influence the **ISR**[2]: STSInt bit and do not have any effect on the **STSInt** contents or behavior, with the exception of the GPIxIntEn interrupt enable bits, which control the generation of the STSInt[3:0] interrupts.

Bit 7: TxEmptyIntEn

Set the TxEmptyIntEn bit high to enable routing the **STSInt**[7]: TxEmptyInt interrupt to **ISR**[2]. If TxEmptyIntEn is set low, TxEmptyInt is not routed to **ISR**[2].

Bit 6: SleepIntEn

Set the SleepIntEn bit high to enable routing the **STSInt**[6]: SleepInt interrupt to **ISR**[2]. If SleepIntEn is set low, SleepInt is not routed to **ISR**[2].

Bit 5: ClkRdyIntEn

Set the ClkRdyIntEn bit high to enable routing the **STSInt**[6]: ClkReady interrupt to **ISR**[2]. If ClkRdyIntEn is set low, ClkReady is not routed to **ISR**[2].

Bit 4: No Function

Bits 3–0: GPIxIntEn

Set the GPIxIntEn bits high to enable generating the **STSInt**[3:0]: GPIxInt interrupts. If any of the GPIxIntEn bits are set low, the associated GPIxInt interrupts are not generated.

Status Interrupt Register (STSInt)

ADDRESS:		0x08						
MODE:		R/COR						
BIT	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
NAME	TxEmptyInt	SleepInt	ClkReady	—	GPI3Int	GPI2Int	GPI1Int	GPI0Int
RESET	0	0	0	0	0	0	0	0

Bit 7: TxEmptyInt

The TxEmptyInt interrupt is generated when both the TxFIFO is empty and the last character has completed transmission. TxEmptyInt is cleared after **STSInt** is read. TxEmptyInt generates an interrupt in **ISR**[2] if enabled by **STSIntEn**[7].

Bit 6: SleepInt

The SleepInt status bit is generated when the MAX3108 enters sleep mode. SleepInt is cleared when the MAX3108 exits sleep mode. This status bit is cleared when the clock is disabled and is not cleared by reading **STSInt**. SleepInt generates an interrupt in **ISR**[2] if enabled by **STSIntEn**[6].

Bit 5: ClkReady

The ClkReady status bit is generated when the clock, the predivider, and the PLL have settled, signifying that the MAX3108 is ready for data communication. The ClkReady bit only works with the crystal oscillator. It does not work with external clocking through XIN.

ClkReady is cleared when the clock is disabled and is not cleared after **STSInt** is read. ClkReady generates an interrupt in **ISR**[2] if enabled by **STSIntEn**[5].

Bit 4: No Function

Bits 3–0: GPIxInt

The GPIxInt interrupts are generated when a change of logic state occurs on the associated GPIO input. The GPIxInt interrupts are cleared after **STSInt** is read. The GPIxInt interrupts generate an interrupt in **ISR**[2] if enabled by the corresponding bits in **STSIntEn**[3:0].

MODE1 Register

ADDRESS:	0x09							
MODE:	R/W							
BIT	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
NAME	—	AutoSleep	ForcedSleep	TrnscvCtrl	RTSHiZ	TXHiZ	TxDisabl	RxDisabl
RESET	0	0	0	0	0	0	0	0

Bit 7: IRQSel

Depending on the logic level of the IRQSel bit, $\overline{IRQ}$ has different meanings. After a hardware (POR and $\overline{RST}$) or software (**MODE2**{0} reset, the IRQSel bit is set low and, after a short delay, the IRQ output signals the end of the power-up sequence. $\overline{IRQ}$ is low during power-up and transitions to high when the MAX3108 is ready to be programmed.

Set IRQSel high to enable interrupt driven operation after the power-up sequence. Essentially, IRQSel is a global enable for all interrupts that acts in addition to the interrupt enable registers.

Bit 6: AutoSleep

Set the AutoSleep bit high to set the MAX3108 to automatically enter low-power sleep mode after a period of no activity (see the *Auto-Sleep Mode* section). An interrupt is generated in **STSInt**[6]: SleepInt when the MAX3108 enters sleep mode.

Bit 5: ForcedSleep

Set the ForcedSleep bit high to force the MAX3108 into low-power sleep mode (see the *Forced-Sleep Mode* section). The current sleep state can be read out through the ForcedSleep bit, even when the UART is in sleep mode.

Bit 4: TrnscvCtrl

Set the TrnscvCtrl bit high to enable auto transceiver direction control mode. $\overline{RTS}$ automatically controls the transceiver's transmit/receive enable/disable inputs in this mode. $\overline{RTS}$ is logic-low so that the transceiver is in receive mode with the transmitter disabled until the TxFIFO contains data available for transmission, at which point $\overline{RTS}$ is automatically set logic-high before the transmitter sends out the data. Once the transmitter is empty, $\overline{RTS}$ is automatically forced low again.

Setup and hold times for $\overline{RTS}$ with respect to the TX output can be defined through the **HDplxDelay** register. A transmitter empty interrupt is generated in **ISR**[5] when the TxFIFO is empty.

Bit 3: RTSHiZ

Set the RTSHiZ bit high to three-state $\overline{RTS}$.

Bit 2: TXHiZ

Set the TXHiZ bit high to three-state the TX output.

Bit 1: TxDisabl

Set the TxDisabl bit high to disable transmission. If the TxDisabl bit is set high during transmission, the transmitter completes sending out the current character and then ceases transmission. Data still present in the transmit FIFO remains in the TxFIFO. The TX output is set to logic-high after transmission.

In auto transceiver direction control mode, TxDisabl is high when the transmitter is completely empty.

Bit 0: RxDisabl

Set the RxDisabl bit high to disable the receiver so that the receiver stops receiving data. All data present in the receive FIFO remains in the RxFIFO.

MODE2 Register

ADDRESS:	0x0A							
MODE:		R/W						
BIT	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
NAME	EchoSuprs	MultiDrop	Loopback	SpecialChr	RFifoEmptyInv	RxTrgInv	FIFORst	RST
RESET	0	0	0	0	0	0	0	0

Bit 7: EchoSuprs

Set the EchoSuprs bit high to discard any data that the MAX3108 receives when its transmitter is busy transmitting. In half-duplex communication such as RS-485 and IrDA, this allows blocking of the locally echoed data. The receiver can block data for an extended time after the transmitter ceases transmission by programming a hold time in **HDplxDelay**[3:0].

Bit 6: MultiDrop

Set the MultiDrop bit high to enable the 9-bit multidrop mode. If this bit is set, parity checking is not performed by the receiver and parity generation is not done by the transmitter. The address/data indication takes the place of the parity bit in received and transmitted data words. The parity error interrupt in **LSR**[2] has a different meaning in multidrop mode: it represents the 9th bit (address/data indication) that is received with each 9-bit data character.

Bit 5: Loopback

Set the Loopback bit high to enable internal local loopback mode. This internally connects TX to RX and also $\overline{RTS}$ to $\overline{CTS}$. In local loopback mode, the TX output and the RX input are disconnected from the internal transmitter and receiver. The TX output is in three-state. The $\overline{RTS}$ output remains connected to the internal logic and reflects the logic state programmed in **LCR**[7]. The $\overline{CTS}$ input is disconnected from $\overline{RTS}$ and the internal logic. $\overline{CTS}$ thus remains in a high-impedance state.

Bit 4: SpecialChr

Set the SpecialChr bit high to enable special character detection. The receiver can detect up to four special characters, as selected in **FlowCtrl**[5:4] and defined in the **XON1**, **XON2**, **XOFF1**, and/or **XOFF2** registers, optionally in combination with GPIOx inputs if enabled through **FlowCtrl**[2]: GPIAddr. When a special character is received, it is put into the RxFIFO and a special character detect interrupt is generated in **ISR**[1].

Special character detection can be used in addition to auto XON/XOFF flow control if enabled by **FlowCtrl**[3]: SwFlowEn. In this case, XON/XOFF flow control is limited to single byte XON and XOFF characters (**XON1** and **XOFF1**), and only two special characters can be defined (**XON2** and **XOFF2**).

Bit 3: RFifoEmtyInv

Set the RFifoEmtyInv bit high to invert the meaning of the receiver empty interrupt in **ISR**[6]: RFifoEmptyInt. If RFifoEmtyInv is set low, RFifoEmptyInt is generated when the receive FIFO is empty. If RFifoEmtyInv is set high, RFifoEmptyInt is generated when data is put into the empty receive FIFO.

Bit 2: RxTrgInv

Set the RxTrgInv bit high to invert the meaning of the RxFIFO triggering. If the RxTrgInv bit is set low, an interrupt is generated in **ISR**[3]: RxTrigInt when the RxFIFO fill level is filled up to above the trigger level programmed into **FIFOTrgLvl**[7:4]. If RxTrgInv is set high, an interrupt is generated in **ISR**[3] when the RxFIFO is emptied to below the trigger level programmed into **FIFOTrgLvl**[7:4].

Bit 1: FIFORst

Set the FIFORst bit high to clear all data contents from both the receive and transmit FIFOs. After a FIFO reset, set FIFORst low to continue normal operation.

Bit 0: RST

Set the RST bit high to initiate software reset for the MAX3108. The I²C/SPI bus stays active during this reset; communication with the MAX3108 is possible while RST is set. All register bits are reset to their reset state and all FIFOs are cleared during a reset.

Set RST low to continue normal operation after a software reset. The MAX3108 requires reprogramming following a software reset.

Line Control Register (LCR)

ADDRESS:	0x0B							
MODE:	R/W							
BIT	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
NAME	$\overline{\text{RTS}}$bit	TxBreak	ForceParity	EvenParity	ParityEn	StopBits	Length1	Length0
RESET	0	0	0	0	0	1	0	1

Bit 7: $\overline{\text{RTS}}$bit

The $\overline{\text{RTS}}$bit bit provides direct control of the $\overline{\text{RTS}}$ output logic state. If $\overline{\text{RTS}}$bit is logic 1, then $\overline{\text{RTS}}$ is logic 1; if it is logic 0, then $\overline{\text{RTS}}$ is logic 0. $\overline{\text{RTS}}$bit only works when **CLKSource**[7]: CLKtoRTS is set low.

Bit 6: TxBreak

Set the TxBreak bit high to generate a line break whereby the TX output is held low. TX remains low until TxBreak is set low.

Bit 5: ForceParity

The ForceParity bit enables forced parity that overrides normal parity generation. Set both the **LCR**[3]: ParityEn and ForceParity bits high to use forced parity. In forced-parity mode, the parity bit is forced high by the transmitter if the **LCR**[4]: EvenParity bit is low. The parity bit is forced low if EvenParity is high. Forced parity mode enables the transmitter to control the address/data bit in 9-bit multidrop communication.

Bit 4: EvenParity

Set the EvenParity bit high to enable even parity for both the transmitter and receiver. If EvenParity is set low, odd parity is used.

Bit 3: ParityEn

Set the ParityEn bit high to enable the use of a parity bit on the TX and RX interfaces. Set the ParityEn bit low to disable parity usage.

If ParityEn is set low, then no parity bit is generated by the transmitter or expected by the receiver. If ParityEn is set high, the transmitter generates the parity bit whose polarity is defined in **LCR**[4]: EvenParity, and the receiver checks the parity bit according to the same polarity.

Bit 2: StopBits

The StopBits bit defines the number of stop bits and depends on the length of the word programmed in **LCR**[1:0] (Table 1). For example, when StopBits is set high and the word length is 5, the transmitter generates a word with a stop bit length equal to 1.5 baud periods. Under these conditions, the receiver recognizes a stop bit length greater than a one-bit duration.

Bits 1 and 0: Lengthx

The Lengthx bits configure the length of the words that the transmitter generates and the receiver checks for at the asynchronous TX and RX interfaces (Table 2).

Table 1. StopBits Truth Table

StopBits	WORD LENGTH	STOP BIT LENGTH
0	5, 6, 7, 8	1
1	5	1–1.5
1	6, 7, 8	2

Table 2. Lengthx Truth Table

Length1	Length0	WORD LENGTH
0	0	5
0	1	6
1	0	7
1	1	8

Receiver Timeout Register (RxTimeOut)

ADDRESS:		0x0C						
MODE:		R/W						
BIT	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
NAME	TimOut7	TimOut6	TimOut5	TimOut4	TimOut3	TimOut2	TimOut1	TimOut0
RESET	0	0	0	0	0	0	0	0

Bits 7–0: TimOutx

The **RxTimeOut** register allows programming a time delay from after the last (newest) character in the receive FIFO was received until a receive data timeout interrupt is generated in **LSR**[0]. The units of TimOutx are measured in complete character frames, which are dependent on the character length, parity, and STOP bit settings, and baud rate. If the value in **RxTimeOut** equals zero, a timeout interrupt is not generated.

HDplxDelay Register

ADDRESS:		0x0D						
MODE:		R/W						
BIT	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
NAME	Setup3	Setup2	Setup1	Setup0	Hold3	Hold2	Hold1	Hold0
RESET	0	0	0	0	0	0	0	0

The **HDplxDelay** register allows programming setup and hold times between $\overline{RTS}$ transitions and TX output activity in auto transceiver direction control mode, enabled by setting the **MODE1**[4]: TrnscvCtrl bit high. The hold time can also be used to ensure echo suppression in half-duplex communication. **HDplxDelay** functions in 2x and 4x rate modes.

Bits 7–4: Setupx

The Setupx bits define a setup time for $\overline{RTS}$ to transition high before the transmitter starts transmission of its first character in auto transceiver direction control mode, enabled by setting the **MODE1**[4]: TrnscvCtrl bit high. This allows the MAX3109 to account for skew times between the external transmitter's enable delay and propagation delays. Setupx can also be used to fix a stable state on the transmission line prior to the start of transmission.

The resolution of the **HDplxDelay** setup time delay is one bit interval, or one over the baud rate; this delay is baud-rate dependent. The maximum delay is 15 bit intervals.

Bits 3–0: Holdx

The Holdx bits define a hold time for $\overline{RTS}$ to be held high after the transmitter ends transmission of its last character in auto transceiver direction control mode, enabled by setting the **MODE1**[4]: TrnscvCtrl bit high. $\overline{RTS}$ transitions low after the hold time delay, which starts after the last STOP bit was sent. This keeps the external transmitter enabled during the hold time duration.

The Holdx bits also define a delay in echo suppression mode, enabled by setting the **MODE2**[7]: EchoSuprs bit high. See the *Echo Suppression* section for more information.

The resolution of the **HDplxDelay** hold time delay is one bit interval, or one over the baud rate. Thus, this delay is baud-rate dependent. The maximum delay is 15 bit intervals.

IrDA Register

ADDRESS:	0x0E							
MODE:	R/W							
BIT	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
NAME	—	—	TxInv	RxInv	MIR	—	SIR	IrDAEn
RESET	0	0	0	0	0	0	0	0

The **IrDA** register allows selection of IrDA SIR- and MIR-compliant pulse shaping at the TX and RX interfaces. It also allows inversion of the TX and RX logic, separate from whether IrDA pulse shaping is enabled or not.

Bits 7, 6, and 2: No Function

Bit 5: TxInv

Set the TxInv bit high to invert the logic at the TX output. This functionality is separate from IrDA operation.

Bit 4: RxInv

Set the RxInv bit high to invert the logic at the RX input. This functionality is separate from IrDA operation.

Bit 3: MIR

Set the MIR and IrDAEn bits high to select IrDA 1.1 (MIR) with 1/4-period pulse widths.

Bit 1: SIR

Set the SIR and IrDAEn bits high to select IrDA 1.0 pulses (SIR) with 3/16th-period pulse widths.

Bit 0: IrDAEn

Set the IrDAEn bit high to program the MAX3108 to produce IrDA-compliant pulses at the TX output and expect IrDA-compliant pulses at the RX input. If IrDAEn is set low, normal (non-IrDA) pulses are generated by the transmitter and expected by the receiver. Use IrDAEn in conjunction with the SIR or MIR bits to select the pulse width.

Flow Level Register (FlowLvl)

ADDRESS:	0x0F							
MODE:	R/W							
BIT	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
NAME	Resume3	Resume2	Resume1	Resume0	Halt3	Halt2	Halt1	Halt0
RESET	0	0	0	0	0	0	0	0

The **FlowLvl** register is used for selecting the RxFIFO threshold levels used for auto software (XON/XOFF) and hardware ($\overline{RTS}$/$\overline{CTS}$) flow control.

Bits 7–4: Resumex

The Resumex bits set the receive FIFO threshold at which an XON character is automatically sent in auto software flow control mode or $\overline{RTS}$ is automatically asserted in AutoRTS mode. These flow control actions occur once the RxFIFO is emptied to below the value in Resumex. This signals the far-end station to resume transmission. The threshold level is calculated as 8 x Resumex. The resulting possible threshold-level range is 0 to 120 (decimal).

Bits 3–0: Haltx

The Haltx bits set the receive FIFO threshold level at which an XOFF character is automatically sent in auto software flow control mode or $\overline{RTS}$ is automatically deasserted in AutoRTS mode. These flow control actions occur once the RxFIFO is filled to above the value in Haltx. This signals the far-end station to halt transmission. The threshold level is calculated as 8 x Haltx. The resulting possible threshold-level range is 0 to 120 (decimal).

FIFO Interrupt Trigger Level Register (FIFOTrgLvl)

ADDRESS:	0x10							
MODE:	R/W							
BIT	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
NAME	RxTrig3	RxTrig2	RxTrig1	RxTrig0	TxTrig3	TxTrig2	TxTrig1	TxTrig0
RESET	1	1	1	1	1	1	1	1

Bits 7–4: RxTrigx

The RxTrigx bits allow definition of the receive FIFO threshold level at which the MAX3108 generates an interrupt in **ISR**[3]. This interrupt can be used to signal that either the receive FIFO is nearing overflow or a predefined number of FIFO locations are available for being read out in one block, depending on the state of the **MODE2**[2]: RxTrgInv bit.

The selectable threshold resolution is eight FIFO locations, so the actual FIFO trigger level is calculated as 8 x RxTrigx. The resulting possible trigger-level range is 0 to 120 (decimal).

Bits 3–0: TxTrigx

The TxTrigx bits allow definition of the transmit FIFO threshold level at which the MAX3108 generates an interrupt in **ISR**[4]. This interrupt can be used to manage data flow to the transmit FIFO. For example, if the trigger level is defined near the bottom of the TxFIFO, the host knows that a predefined number of FIFO locations are available for being written to in one block. Alternatively, if the trigger level is set near the top of the FIFO, the host is warned when the transmit FIFO is nearing overflow.

The selectable threshold resolution is eight FIFO locations, so the actual FIFO trigger level is calculated as 8 x TxTrigx. The resulting possible trigger-level range is 0 to 120 (decimal).

Transmit FIFO Level Register (TxFIFOLvl)

ADDRESS:	0x11							
MODE:	R							
BIT	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
NAME	TxFL7	TxFL6	TxFL5	TxFL4	TxFL3	TxFL2	TxFL1	TxFL0
RESET	0	0	0	0	0	0	0	0

Bits 7–0: TxFLx

The **TxFIFOLvl** register represents the current number of words in the transmit FIFO.

Receive FIFO Level Register (RxFIFOLvl)

ADDRESS:	0x12							
MODE:	R							
BIT	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
NAME	RxFL7	RxFL6	RxFL5	RxFL4	RxFL3	RxFL2	RxFL1	RxFL0
RESET	0	0	0	0	0	0	0	0

Bits 7–0: RxFLx

The **RxFIFOLvl** register represents the current number of words in the receive FIFO.

Flow Control Register (FlowCtrl)

ADDRESS:	0x13							
MODE:	R/W							
BIT	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
NAME	SwFlow3	SwFlow2	SwFlow1	SwFlow0	SwFlowEn	GPIAddr	AutoCTS	AutoRTS
RESET	0	0	0	0	0	0	0	0

The **FlowCtrl** register configures hardware (RTS/CTS) and software (XON/XOFF) flow control as well as special characters detection.

Bits 7–4: SwFlowx

The SwFlowx bits select the XON and XOFF characters used for auto software flow control and/or special character detection in combination with the characters programmed in the **XON1**, **XON2**, **XOFF1**, and/or **XOFF2** registers. See Table 3.

If auto software flow control is enabled (through **FlowCtrl**[3]: SwFlowEn) and special character detection is not enabled, SwFlowx allows selecting either single or dual XON/XOFF character flow control. When double character flow control is enabled, the transmitter sends out **XON1/XOFF1** first followed by **XON2/XOFF2** during receive flow control. For transmit flow control, the receiver only recognizes the received character sequence **XON1/XOFF1** followed by **XON2/XOFF2** as a valid control sequence to resume/halt transmission.

If only special character detection is enabled (through **MODE2**[4]: SpecialChr), while auto software flow control is disabled, then SwFlowx allows selecting either single or double character detection. Single character detection allows the detection of two characters: **XON1** or **XON2** and **XOFF1** or **XOFF2**. Double character detection does not distinguish between the sequence or the two received **XON1**/**XON2** or **XOFF1**/**XOFF2** characters. The two characters have to be received in succession, but it is insignificant which of the two is received first. The special characters are deposited in the receive FIFO. An **ISR**[1]: SpCharInt interrupt is generated when special characters are received.

Auto software flow control and special character detection can be enabled to operate simultaneously. If both are enabled, **XON1** and **XOFF1** define the auto flow control characters, while **XON2** and **XOFF2** constitute the special character detection characters.

Bit 3: SwFlowEn

Set the SwFlowEn bit high to enable auto software flow control. The characters used for automatic software flow control are selected by SwFlowx. If special character detection is enabled by setting the **MODE2**[4]: SpecialChr bit high in addition to automatic software flow control, **XON1** and **XOFF1** are used for flow control while **XON2** and **XOFF2** define the special characters.

Bit 2: GPIAddr

Set the GPIAddr bit high to enable the four GPIO inputs to be used in conjunction with **XOFF2** for the definition of a special character. This can be used, for example, for defining the address of an RS-485 slave device through hardware. The GPIOx input logic levels define the four LSBs of the special character, while the four MSBs are defined by the **XOFF2**[7:4] bits. The contents of the **XOFF2**[3:0] bits are neglected while the GPIO inputs are used in special character definition. Reading the **XOFF2** register does not reflect the logic on GPIO in this mode.

Bit 1: AutoCTS

Set the AutoCTS bit high to enable AutoCTS flow control mode. In this mode, the transmitter stops and starts sending data at the TX interface depending on the logic state of the $\overline{CTS}$ input. See the *Auto Hardware Flow Control* section for more information about AutoCTS flow control mode. Logic changes at the $\overline{CTS}$ input result in an interrupt in **ISR**[7]: CTSInt. The transmitter must be turned off by setting the **MODE1**[1]: TxDisabl bit high before AutoCTS mode is enabled.

Bit 0: AutoRTS

Set the AutoRTS bit high to enable AutoRTS flow control mode. In this mode, the logic state of the $\overline{RTS}$ output is dependent on the receive FIFO fill level. The FIFO thresholds at which $\overline{RTS}$ changes state are set in **FlowLvl**. See the *Auto Hardware Flow Control* section for more information about AutoRTS flow control mode.

Table 3. SwFlow[3:0] Truth Table

RECEIVE FLOW CONTROL		TRANSMIT FLOW CONTROL/SPECIAL CHARACTER DETECTION		DESCRIPTION
SwFlow3	SwFlow2	SwFlow1	SwFlow0	
0	0	0	0	No flow control/no special character detection.
0	0	X	X	No receive flow control.
1	0	X	X	Transmitter generates XON1, XOFF1.
0	1	X	X	Transmitter generates XON2, XOFF2.
1	1	X	X	Transmitter generates XON1, XON2, XOFF1, and XOFF2.
X	X	0	0	No transmit flow control.
X	X	1	0	Receiver compares XON1 and XOFF1 and controls the transmitter accordingly. XON1 and XOFF1 special character detection.
X	X	0	1	Receiver compares XON2 and XOFF2 and controls the transmitter accordingly. XON2 and XOFF2 special character detection.
X	X	1	1	Receiver compares XON1, XON2, XOFF1, and XOFF2 and controls the transmitter accordingly. XON1, XON2, XOFF1, and XOFF2 special character detection.

X = Don't care.

XON1 Register

ADDRESS:	0x14							
MODE:	R/W							
BIT	7	6	5	4	3	2	1	0
NAME	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
RESET	0	0	0	0	0	0	0	0

The **XON1** and **XON2** register contents define the XON character used for automatic XON/XOFF flow control and/or the special characters used for special-character detection. See the **FlowCtrl** register description for more information.

Bits 7–0: Bitx

These bits define the **XON1** character if single character XON auto software flow control is enabled in **FlowCtrl**[7:4]. If double-character flow control is selected in **FlowCtrl**[7:4], these bits constitute the least significant byte of the 2-byte XON character. If special character detection is enabled in **MODE2**[4] and auto flow control is not enabled, these bits define a special character.

If both special character detection and auto software flow control are enabled, **XON1** defines the XON flow control character.

XON2 Register

ADDRESS:	0x15							
MODE:	R/W							
BIT	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
NAME	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
RESET	0	0	0	0	0	0	0	0

The **XON1** and **XON2** register contents define the XON character for automatic XON/XOFF flow control and/or the special characters used in special-character detection. See the **FlowCtrl** register description for more information.

Bits 7–0: Bitx

These bits define the **XON2** character if single character auto software flow control is enabled in **FlowCntrl**[7:4]. If double-character flow control is selected in **FlowCntrl**[7:4], these bits constitute the most significant byte of the 2-byte XON character. If special character detection is enabled in **MODE2**[4] and auto software flow control is not enabled, these bits define a special character.

If both special character detection and auto software flow control are enabled, **XON2** defines a special character.

XOFF1 Register

ADDRESS:	0x16							
MODE:	R/W							
BIT	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
NAME	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
RESET	0	0	0	0	0	0	0	0

The **XOFF1** and **XOFF2** register contents define the XOFF character for automatic XON/XOFF flow control and/or the special characters used in special character detection. See the **FlowCtrl** register description for more information.

Bits 7–0: Bitx

These bits define the **XOFF1** character if single character XOFF auto software flow control is enabled in **FlowCntrl**[7:4]. If double character flow control is selected in **FlowCntrl**[7:4], these bits constitute the least significant byte of the 2-byte XOFF character. If special character detection is enabled in **MODE2**[4] and auto software flow control is not enabled, these bits define a special character.

If both special character detection and auto software flow control are both enabled, **XOFF1** defines the XOFF flow control character.

XOFF2 Register

ADDRESS:	0x17							
MODE:	R/W							
BIT	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
NAME	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
RESET	0	0	0	0	0	0	0	0

The **XOFF1** and **XOFF2** register contents define the XOFF character for automatic XON/XOFF flow control and/or the special characters used in special character detection. See the **FlowCtrl** register description for more information.

Bits 7–0: Bitx

These bits define the **XOFF1** character if single character XOFF auto software flow control is enabled in **FlowCntrl**[7:4]. If double character flow control is selected in **FlowCntrl**[7:4], these bits constitute the least significant byte of the 2-byte XOFF character. If special character detection is enabled in **MODE2**[4] and auto software flow control is not enabled, these bits define a special character.

If both special character detection and auto software flow control are both enabled, **XOFF2** defines a special character.

GPIO Configuration Register (GPIOConfg)

ADDRESS:	0x18							
MODE:	R/W							
BIT	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
NAME	GP3OD	GP2OD	GP1OD	GP0OD	GP3Out	GP2Out	GP1Out	GP0Out
RESET	0	0	0	0	0	0	0	0

The four GPIOs can be configured as inputs or outputs and can be operated in push-pull or open-drain mode. The reference clock needs to be active for the GPIOs to work.

Bits 7–4: GPxOD

Set the GPxOD bits high to configure the associated GPIOs as open-drain outputs. Set the GPxOD bits low to configure the associated GPIOs as push-pull outputs.

When configured as inputs in GPxOut, the GPIOs are high-impedance inputs with weak pulldown resistors, regardless of the state of GPxOD.

Bits 3–0: GPxOut

The GPxOut bits configure the associated GPIOs to be either inputs or outputs. Set the GPxOut bits high to configure the associated GPIOs as outputs. Set the GPxOut bits low to configure the associated GPIOs as inputs.

GPIO Data Register (GPIOData)

ADDRESS:	0x19							
MODE:	R/W							
BIT	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
NAME	GPI3Dat	GPI2Dat	GPI1Dat	GPI0Dat	GPO3Dat	GPO2Dat	GPO1Dat	GPO0Dat
RESET	0	0	0	0	0	0	0	0

Bits 7–4: GPIxDat

The GPIxDat bits reflect the input logic on the associated GPIOs. These bits are inly updated while the UART clock is active.

Bits 3–0: GPOxDat

The GPOxDat bits allow programming of the logic state of the GPIOs when configured as outputs in **GPIOConfg**[3:0]. For open-drain operation, pullup resistors are needed on the GPIOs.

PLL Configuration Register (PLLConfig)

ADDRESS:	0x1A							
MODE:	R/W							
BIT	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
NAME	PLLFactor1	PLLFactor0	PreDiv5	PreDiv4	PreDiv3	PreDiv2	PreDiv1	PreDiv0
RESET	0	0	0	0	0	0	0	1

Bits 7–6: PLLFactorx

The PLLFactorx bits allow programming the PLL multiplication factor. The input and output frequencies of the PLL must be limited to the ranges shown in Table 4. Enable the PLL in **CLKSource**[2].

Bits 5–0: PreDivx

The PreDivx bits allow programming of the divisor in the PLL's predivider. The divisor must be chosen such that the output frequency of the predivider, which is the PLL's input frequency, is limited to the ranges shown in Table 4. The PLL input frequency is calculated as:

$$f_{PLLIN} = f_{CLK}/PreDiv$$

where f_{CLK} is the input frequency of the crystal oscillator or external clock source (Figure 14), and PreDiv is an integer in the range of 1 to 63.

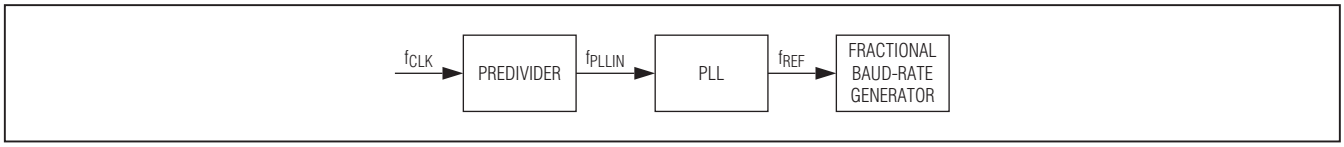


Figure 14. PLL Signal Path

Table 4. PLLFactorx Selection Guide

PLLFactor1	PLLFactor0	MULTIPLICATION FACTOR	f_{PLLIN}		f_{REF}	
			MIN	MAX	MIN	MAX
0	0	6	500kHz	800kHz	3MHz	4.8MHz
0	1	48	850kHz	1.2MHz	40.8MHz	56MHz
1	0	96	425kHz	1MHz	40.8MHz	96MHz
1	1	144	390kHz	667kHz	56MHz	96MHz

Baud-Rate Generator Configuration Register (BRGConfig)

ADDRESS:	0x1B							
MODE:	R/W							
BIT	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
NAME	—	—	4xMode	2xMode	FRACT3	FRACT2	FRACT1	FRACT0
RESET	0	0	0	0	0	0	0	0

Bits 7 and 6: No Function

Bit 5: 4xMode

Set the 4xMode bit high to quadruple the regular (16x sampling) baud rate. Set the 2xMode bit low when 4xMode is enabled. See the *2x and 4x Rate Modes* section for more information.

Bit 4: 2xMode

Set the 2xMode bit high to double the regular (16x sampling) baud rate. Set the 4xMode bit low when 2xMode is enabled. See the *2x and 4x Rate Modes* section for more information.

Bits 3–0: FRACTx

The FRACTx bits are the fractional portion of the baud-rate generator divisor. Set FRACTx to 0000b if not used. See the *Fractional Baud-Rate Generator* section for calculations of how to set this value to select the baud rate.

Baud-Rate Generator LSB Divisor Register (DIVLSB)

ADDRESS:	0x1C							
MODE:	R/W							
BIT	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
NAME	Div7	Div6	Div5	Div4	Div3	Div2	Div1	Div0
RESET	0	0	0	0	0	0	0	1

DIVLSB and **DIVMSB** define the baud-rate generator integer divisor. The minimum value for **DIVLSB** is 1. See the *Fractional Baud-Rate Generator* section for more information.

Bits 7–0: Divx

The Divx bits are the eight LSBs of the integer divisor portion (DIV) of the baud-rate generator.

Baud-Rate Generator MSB Divisor Register (DIVMSB)

ADDRESS:	0x1D							
MODE:	R/W							
BIT	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
NAME	Div15	Div14	Div13	Div12	Div11	Div10	Div9	Div8
RESET	0	0	0	0	0	0	0	0

DIVLSB and **DIVMSB** define the baud-rate generator integer divisor. The minimum value for **DIVLSB** is 1. See the *Fractional Baud-Rate Generator* section for more information.

Bits 7–0: Divx

The Divx bits are the eight MSBs of the integer divisor portion (DIV) of the baud-rate generator.

Clock Source Register (CLKSource)

ADDRESS:	0x1E							
MODE:	R/W							
BIT	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
NAME	CLKtoRTS	—	—	—	PLLBypass	PLLEn	CrystalEn	—
RESET	0	0	0	1	1	0	0	0

Bit 7: CLKtoRTS

Set the CLKtoRTS bit high to route the baud-rate generator (16x baud rate) output clock to $\overline{RTS}$. The $\overline{RTS}$ clock frequency is a factor of 16x, 8x, or 4x of the baud rate in 1x, 2x, and 4x rate modes, respectively.

Bits 6, 5, 4, and 0: No Function

Bit 3: PLLBypass

Set the PLLBypass bit high to bypass the internal PLL and predivider.

Bit 2: PLLEn

Set the PLLEn bit high to enable the internal PLL. Set PLLEn low to disable the internal PLL.

Bit 1: CrystalEn

Set the CrystalEn bit high to enable the crystal oscillator. When using an external clock source at XIN, set CrystalEn low.

Serial Controller Interface

The MAX3108 can be controlled through I2C or SPI as defined by the logic on SPI/$\overline{I2C}$. See the *Bump Configuration* for further details.

SPI Interface

The SPI supports both single-cycle and burst read/write access. The SPI master must generate clock and data signals in SPI MODE0 (i.e., with clock polarity CPOL = 0 and clock phase CPHA = 0).

SPI Single-Cycle Access

Figure 15 shows a single-cycle read, and Figure 16 shows a single-cycle write.

SPI Burst Access

Burst access allows writing and reading multiple data bytes in one block by defining only the initial register address in the SPI command byte. Multiple characters can be loaded into the TxFIFO by using the THR (0x00) as the initial burst write address. Similarly, multiple characters can be read out of the RxFIFO by using the RHR (0x00) as the SPI's burst read address. If the SPI burst address is different from 0x00, the MAX3108 automatically increments the register address after each SPI data byte. Efficient programming of multiple consecutive registers is thus possible. The chip-select input, $\overline{CS}$/A0, must be held low during the whole cycle. The SCLK/SCL clock continues clocking throughout the burst access cycle. The burst cycle ends when the SPI master pulls $\overline{CS}$/A0 high.

For example, writing 128 bytes into the TxFIFO can be achieved by a burst write access using the following sequence:

1) Pull $\overline{CS}$/A0 low.
2) Send SPI write command to address 0x00.
3) Send 128 bytes.
4) Release $\overline{CS}$/A0.

This takes a total of (1 + 128) x 8 clock cycles.

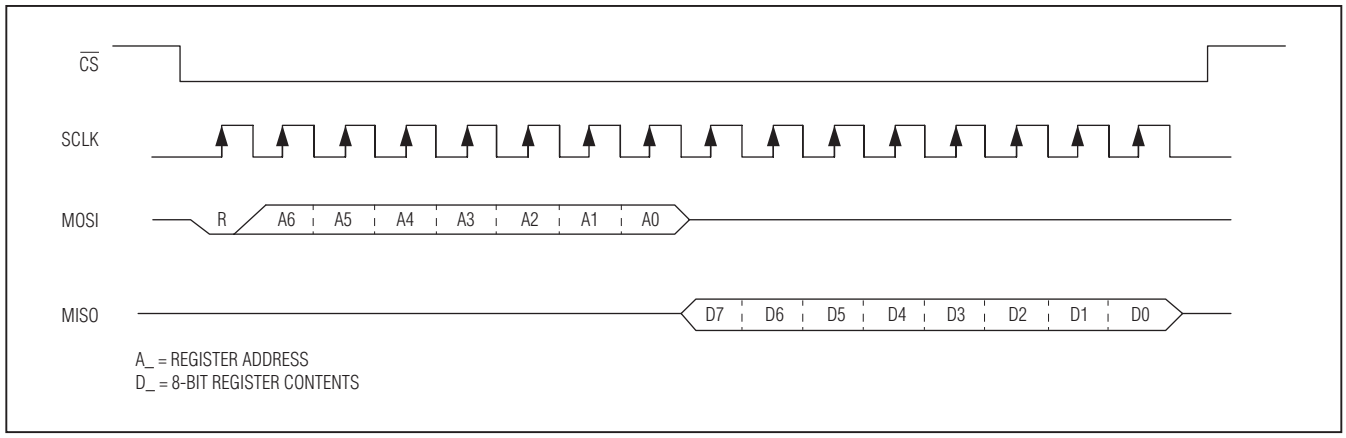


Figure 15. Single-Cycle Read

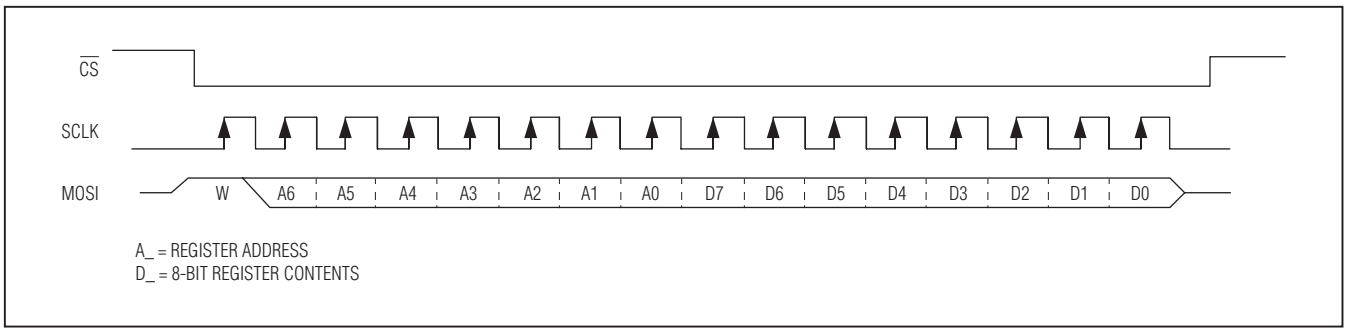


Figure 16. Single-Cycle Write

I²C Interface

The MAX3108 contains an I²C-compatible interface for data communication with a host processor (SCL and SDA). The interface supports a clock frequency of up to 1MHz. SCL and SDA require pullup resistors that are connected to a positive supply.

Table 5. I²C Address Map

MOSI/A1	$\overline{CS}$/A0	I²C WRITE ADDRESS	I²C READ ADDRESS
DGND	DGND	0xD8	0xD9
DGND	V_L	0xC2	0xC3
DGND	SCL	0xC4	0xC5
DGND	SDA	0xC6	0xC7
V_L	DGND	0xC8	0xC9
V_L	V_L	0xCA	0xCB
V_L	SCL	0xCC	0xCD
V_L	SDA	0xCE	0xCF
SCL	DGND	0xD0	0xD1
SCL	V_L	0xD2	0xD3
SCL	SCL	0xD4	0xD5
SCL	SDA	0xD6	0xD7
SDA	DGND	0xC0	0xC1
SDA	V_L	0xDA	0xDB
SDA	SCL	0xDC	0xDD
SDA	SDA	0xDE	0xDF

START, STOP, and Repeated START Conditions

When writing to the MAX3108 using I²C, the master sends a START condition (S) followed by the MAX3108 I²C address. After the address, the master sends the register address of the register that is to be programmed. The master then ends communication by issuing a STOP condition (P) to relinquish control of the bus, or a repeated START condition (Sr) to communicate to another I²C slave. See Figure 17.

Slave Address

The MAX3108 includes a configurable 7-bit I²C slave address, allowing up to 16 MAX3108 devices to share the same I²C bus. The address is defined by connecting the MOSI/A1 and $\overline{CS}$/A0 inputs to DGND, V_L, SCL, or SDA (Table 5). Set the R/$\overline{W}$ bit high to configure the MAX3108 to read mode. Set the R/$\overline{W}$ bit low to configure the MAX3108 to write mode. The address is the first byte of information sent to the MAX3108 after the START condition.

Bit Transfer

One data bit is transferred on the rising edge of each SCL clock cycle. The data on SDA must remain stable during the high period of the SCL clock pulse. Changes in SDA while SCL is high and stable are considered control signals (see the *START, STOP, and Repeated START Conditions* section). Both SDA and SCL remain high when the bus is not active.

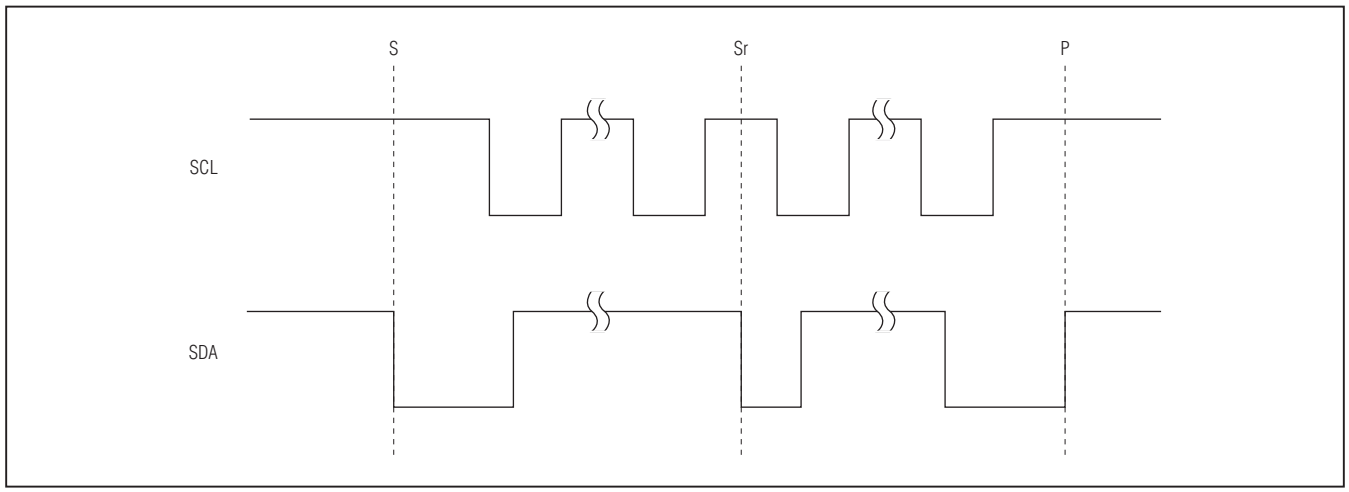


Figure 17. I²C START, STOP, and Repeated START Conditions

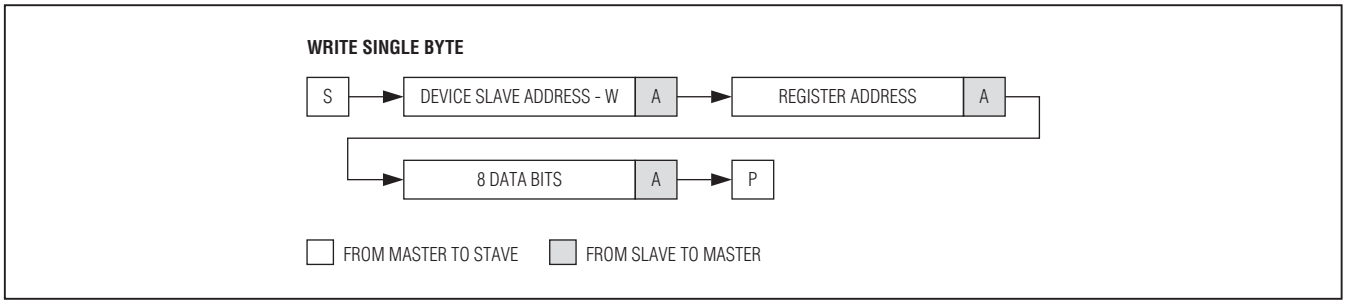


Figure 18. Write Byte Sequence

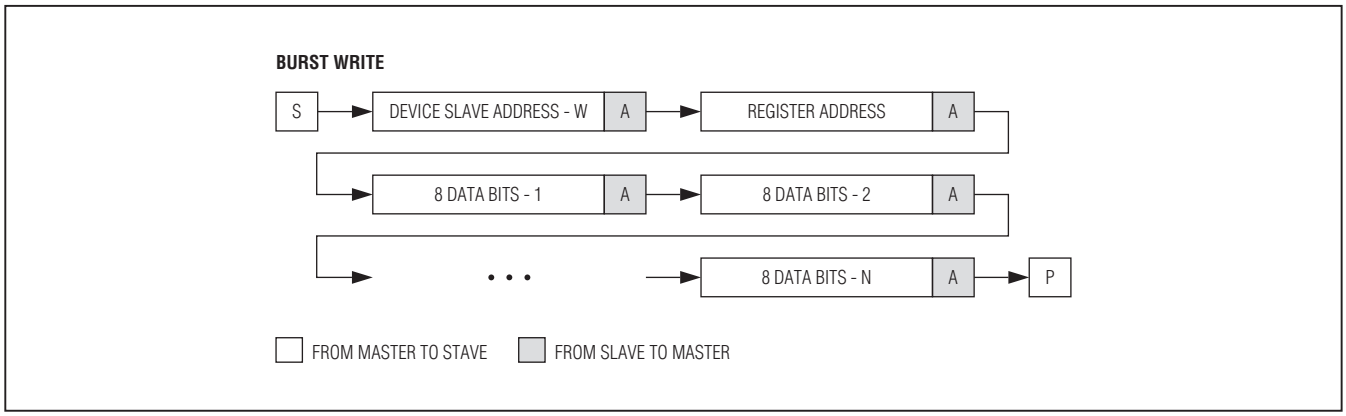


Figure 19. Burst Write Sequence

Single-Byte Write

In this operation, the master sends an address and two data bytes to the slave device (Figure 18). The following procedure describes the single-byte write operation:

1) The master sends a START condition.
2) The master sends the 7-bit slave address plus a write bit (low).
3) The addressed slave asserts an ACK on the data line.
4) The master sends the 8-bit register address.
5) The slave asserts an ACK on the data line only if the address is valid (NACK if not).
6) The master sends 8 data bits.
7) The slave asserts an ACK on the data line.
8) The master generates a STOP condition.

Burst Write

In this operation, the master sends an address and multiple data bytes to the slave device (Figure 19). The slave device automatically increments the register address after each data byte is sent, unless the register being accessed is 0x00, in which case the register address remains the same. The following procedure describes the burst write operation:

1) The master sends a START condition.
2) The master sends the 7-bit slave address plus a write bit (low).
3) The addressed slave asserts an ACK on the data line.
4) The master sends the 8-bit register address.
5) The slave asserts an ACK on the data line only if the address is valid (NACK if not).
6) The master sends 8 data bits.
7) The slave asserts an ACK on the data line.
8) Repeat 6 and 7 N-1 times.
9) The master generates a STOP condition.

Single-Byte Read

In this operation, the master sends an address plus two data bytes and receives one data byte from the slave device (Figure 20). The following procedure describes the single-byte read operation:

1) The master sends a START condition.
2) The master sends the 7-bit slave address plus a write bit (low).
3) The addressed slave asserts an ACK on the data line.
4) The master sends the 8-bit register address.
5) The active slave asserts an ACK on the data line only if the address is valid (NACK if not).
6) The master sends a repeated START condition.
7) The master sends the 7-bit slave address plus a read bit (high).
8) The addressed slave asserts an ACK on the data line.
9) The slave sends 8 data bits.
10) The master asserts a NACK on the data line.
11) The master generates a STOP condition.

Burst Read

In this operation, the master sends an address plus two data bytes and receives multiple data bytes from the slave device (Figure 21). The slave device automatically increments the register address after each data byte is sent, unless the register being accessed is 0x00, in which case the register address remains the same. The following procedure describes the burst byte read operation:

1) The master sends a START condition.
2) The master sends the 7-bit slave address plus a write bit (low).
3) The addressed slave asserts an ACK on the data line.
4) The master sends the 8-bit register address.
5) The slave asserts an ACK on the data line only if the address is valid (NACK if not).

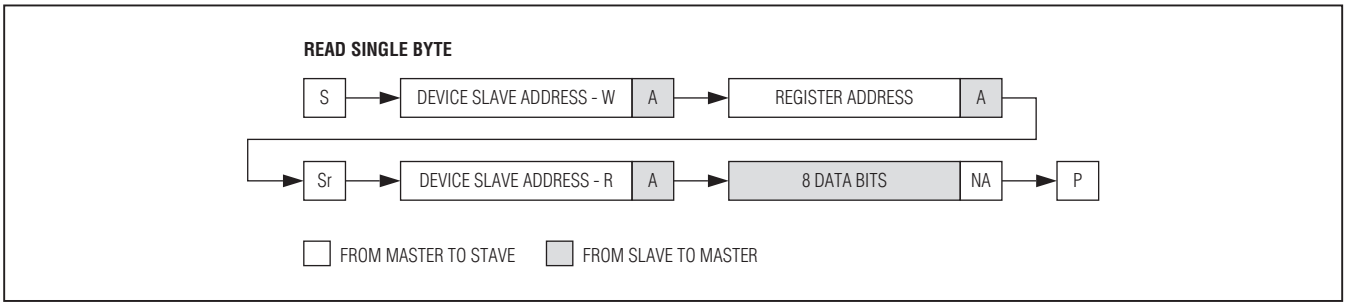


Figure 20. Read Byte Sequence

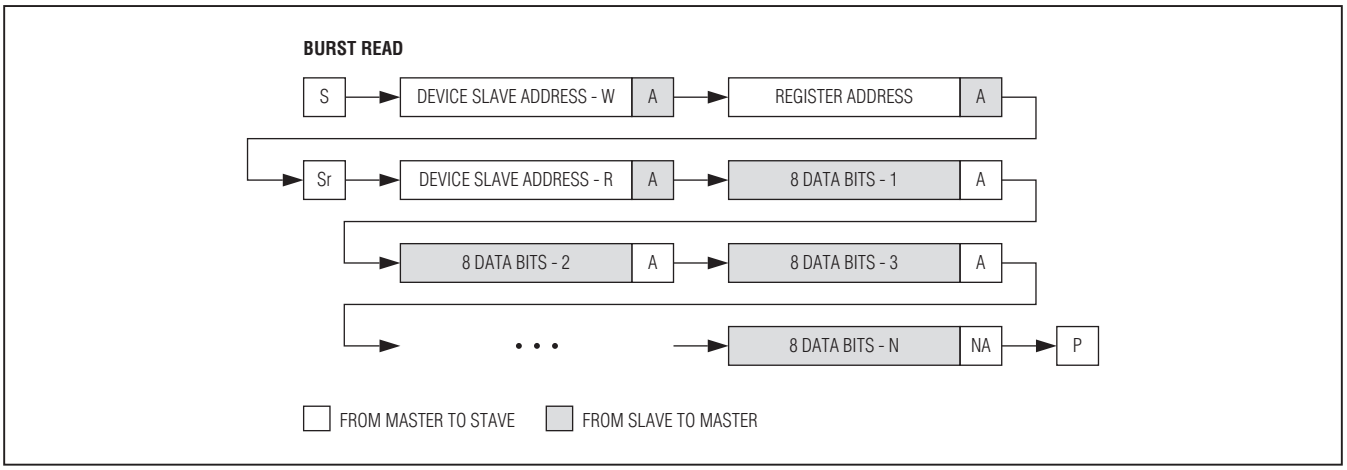


Figure 21. Burst Read Sequence

6) The master sends a repeated START condition.
7) The master sends the 7-bit slave address plus a read bit (high).
8) The slave asserts an ACK on the data line.
9) The slave sends 8 data bits.
10) The master asserts an ACK on the data line.
11) Repeat 9 and 10 N-2 times.
12) The slave sends the last 8 data bits.
13) The master asserts a NACK on the data line.
14) The master generates a STOP condition.

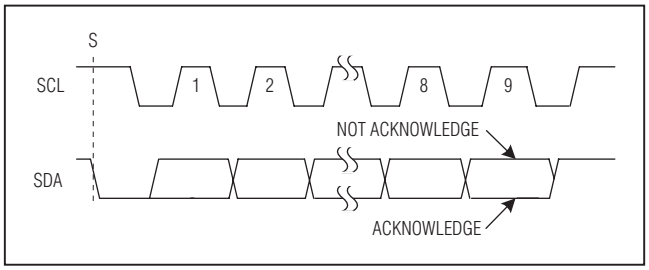


Figure 22. Acknowledge

Acknowledge Bits

Data transfers are acknowledged with an acknowledge bit (ACK) or a not-acknowledge bit (NACK). Both the master and the MAX3108 generate ACK bits. To generate an ACK, pull SDA low before the rising edge of the ninth clock pulse and hold it low during the high period of the ninth clock pulse (Figure 22). To generate a NACK, leave SDA high before the rising edge of the ninth clock pulse and leave it high for the duration of the ninth clock pulse. Monitoring for NACK bits allows for detection of unsuccessful data transfers.

Applications Information

Startup and Initialization

The MAX3108 can be initialized following power-up, a hardware reset, or a software reset as shown in Figure 23. To verify that the MAX3108 is ready for operation after a power-up or reset in an interrupt-driven operation, wait for the IRQ output to deassert.

In polled mode, repeatedly read a known register until the expected contents are returned.

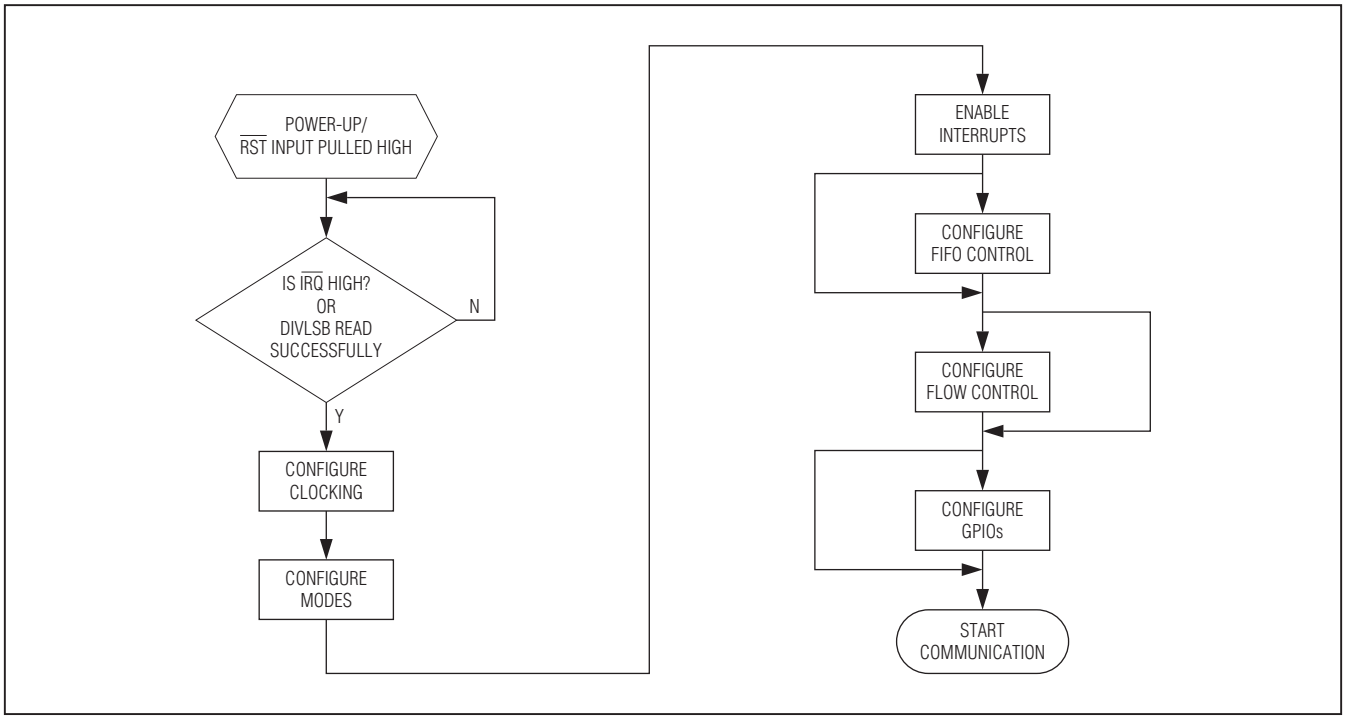


Figure 23. Startup and Initialization Flowchart

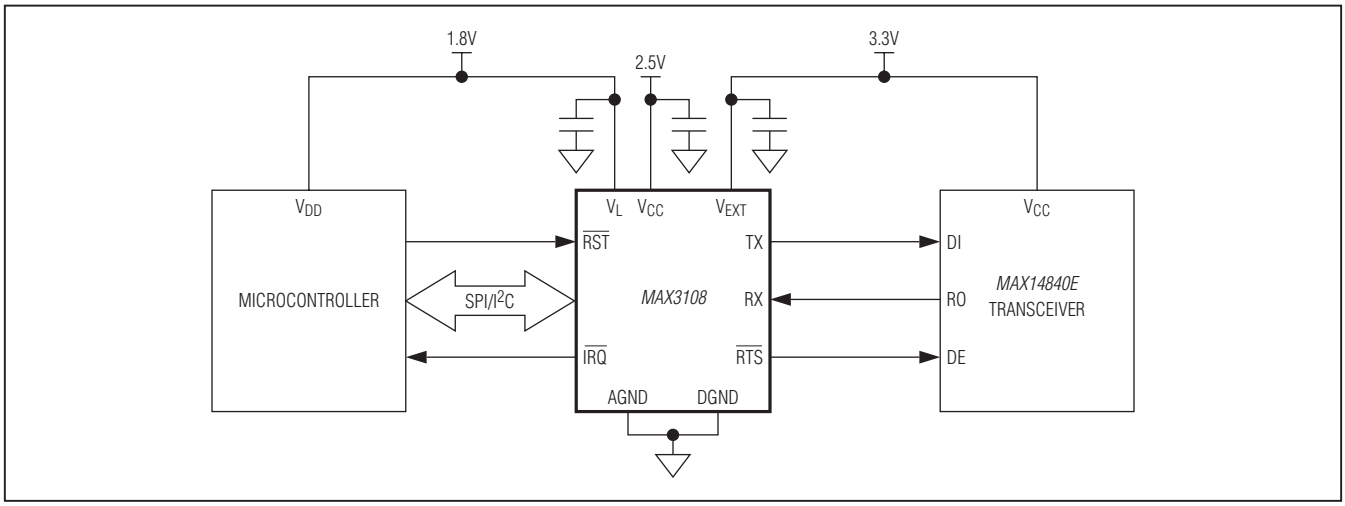


Figure 24. Logic-Level Translation

Low-Power Operation

To reduce the power consumption during normal operation, the following techniques can be adopted:

- Do not use the internal PLL. This saves the most power of the options listed here. Disable and bypass the PLL. With the PLL enabled, the current to the V_{CC} supply is in the range of a few mA (depending on clock frequency and multiplication factor), while it drops to below 1mA if disabled.
- Use an external clock source. The lowest power clocking mode is when an external clock signal is used. This drops the power consumption to about half that of an external crystal.
- Keep the internal clock rates as low as possible.
- Use a low voltage on the V_{CC} supply.
- Use an external 1.8V supply. This saves the power dissipated by the internal 1.8V linear regulator for the 1.8V core supply. Connect an external 1.8V supply to V_{18} and disable the internal regulator by connecting LDOEN to DGND.

Interrupts and Polling

Monitor the MAX3108 by polling the **ISR** register or by monitoring the $\overline{IRQ}$ output. In polled mode, the $\overline{IRQ}$ physical interrupt output is not used and the host controller polls the **ISR** register at frequent intervals to establish the state of the MAX3108.

Alternatively, the physical $\overline{IRQ}$ interrupt can be used to interrupt the host controller after specified events, making polling unnecessary. The $\overline{IRQ}$ output is an open-drain output that requires a pullup resistor to V_L.

Logic-Level Translation

The MAX3108 can be directly connected to transceivers and controllers that have different supply voltages. The V_L input defines the logic voltage levels of the controller interface, while the V_{EXT} voltage defines the logic of the transceiver interface. This ensures flexibility when selecting a controller and transceiver. Figure 24 shows an example of a configuration where the controller, transceiver, and the MAX3108 are powered by three different supplies.

Power-Supply Sequencing

The power supplies of the MAX3108 can be turned on in any order. Each supply can be present over the entire specified range regardless of the presence or level of the others. Ensure the presence of the interface supplies V_L and V_{EXT} before sending input signals to the controller and transceiver interfaces.

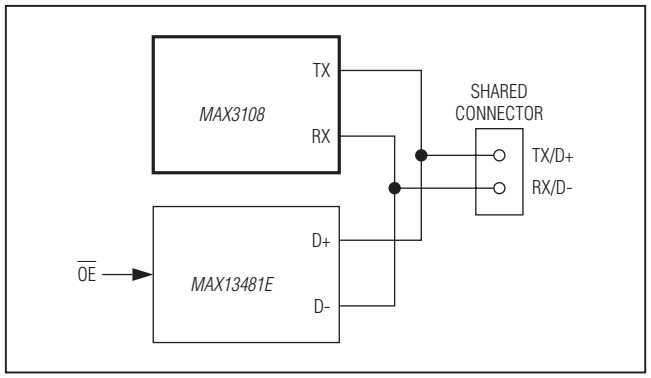


Figure 25. Connector Sharing with a USB Transceiver

Connector Sharing

The TX and $\overline{RTS}$ outputs can be programmed to be high impedance. This feature is used in cases where the MAX3108 shares a common connector with other communications devices. Set the output of the MAX3108 to high impedance when the other communication devices are active. Set the **MODE1**[2]: TXHiZ bit high to set TX to a high-impedance state. Set the **MODE1**[3]: RTSHiZ bit high to set $\overline{RTS}$ to a high-impedance state. Figure 25 shows an example of connector sharing with a USB transceiver.

RS-232 5x3 Application

The four GPIOs can be used to implement the other flow control signals defined in ITU V.24. Figure 26 shows how the GPIOs create the DSR, DTR, DCD, and RI signals found on some RS-232/V.28 interfaces.

Set the **FlowCtrl**[1:0] bits high to enable automatic hardware $\overline{RTS}$/$\overline{CTS}$ flow control.

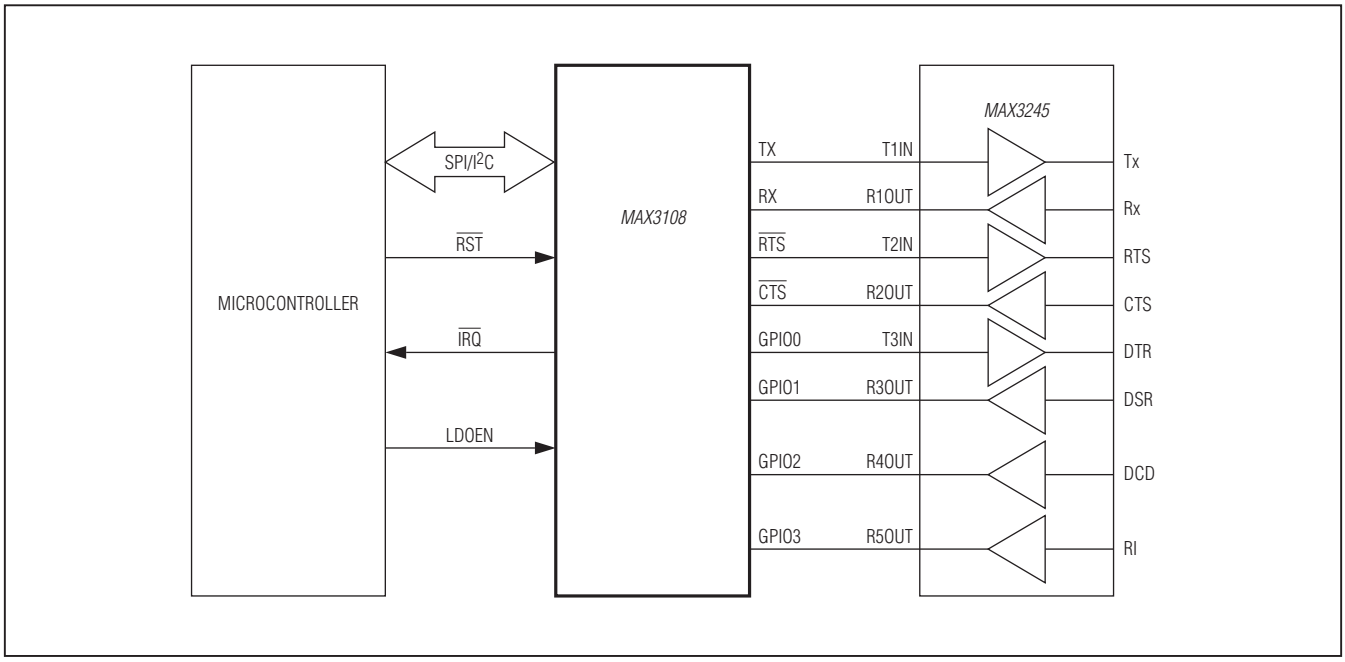


Figure 26. RS-232 Application

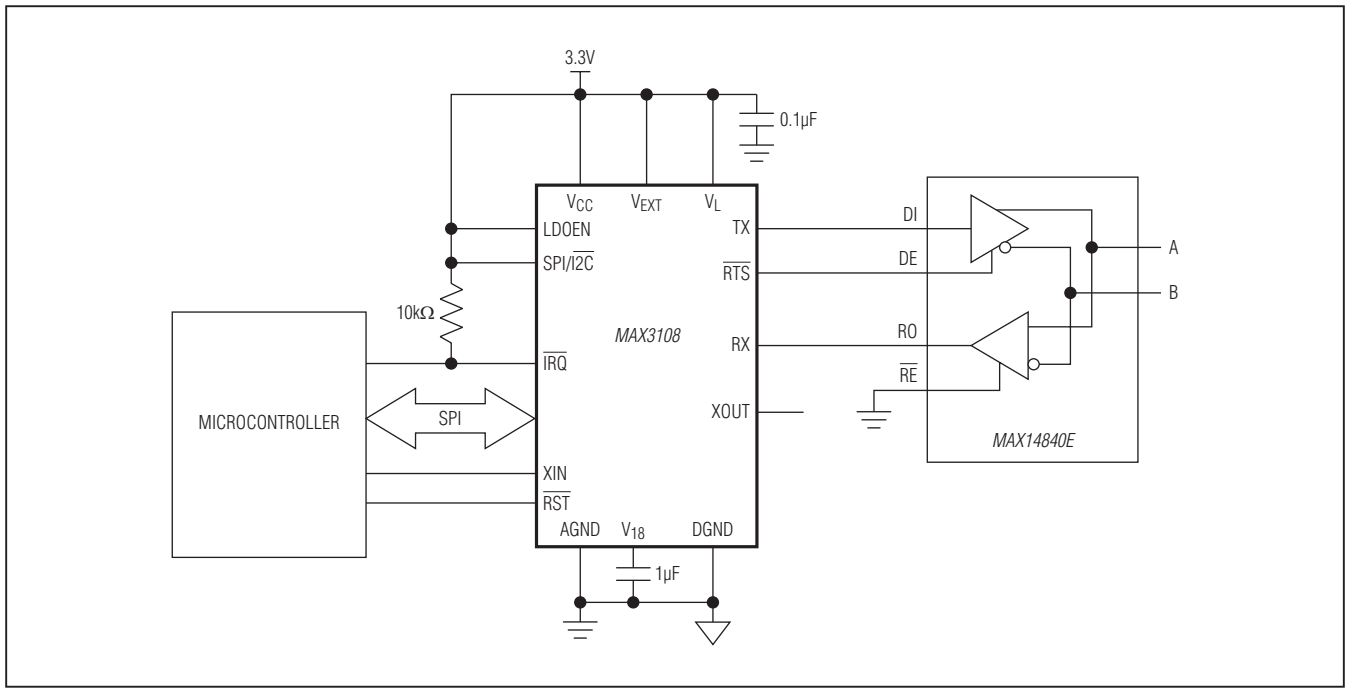


Figure 27. RS-485 Half-Duplex Application

Typical Application Circuit

Figure 27 shows the MAX3108 being used in a half-duplex RS-485 application. The microcontroller, the RS-485 transceiver, and the MAX3108 are powered by a single 3.3V supply. SPI is used as the controller's communication interface. The microcontroller provides an external clock source to clock the UART.

The MAX14840E receiver is always enabled, so echoing occurs. Enable auto echo suppression in the MAX3108 by setting the **MODE2**[7]: EchoSuprs bit high.

Set the **MODE1**[4]: TranscvCtrl bit high to enable auto transceiver direction control in order to automatically control the DE input of the transceiver.

Chip Information

PROCESS: BiCMOS

Package Information

For the latest package outline information and land patterns (footprints), go to **www.maximintegrated.com/packages**. Note that a "+", "#", or "-" in the package code indicates RoHS status only. Package drawings may show a different suffix character, but the drawing pertains to the package regardless of RoHS status.

PACKAGE TYPE	PACKAGE CODE	OUTLINE NO.	LAND PATTERN NO.
25 WLP	W252B2+1	21-0180	Refer to Application Note 1891

Revision History

REVISION NUMBER	REVISION DATE	DESCRIPTION	PAGES CHANGED
0	12/10	Initial release	—
1	2/11	Updated the I_L and I_{SHDN} max numbers in the *DC Electrical Characteristics* table	6
2	1/13	Updated *DC Electrical Characteristics* table, *Bump Description*, updated Figure 27	8, 13, 55