MAX17841B Automotive SPI Communication Interface (ASCI)

General Description

The MAX17841B ASCI combines an SPI port with a universal asynchronous receiver transmitter (UART) specially designed to interface with Maxim battery management devices.

The UART can be configured to automatically perform Manchester encoding/decoding, message framing, parity, wake-up, and keep-alive signaling as required for Maxim's battery management UART protocol.

The UART has programmable baud rates of 0.5Mbps, 1Mbps, or 2Mbps and supports either single-ended or differential signaling. For host efficiency, the UART contains a 28-byte transmit buffer and a 62-byte receive buffer with host-configurable interrupt events.

Applications

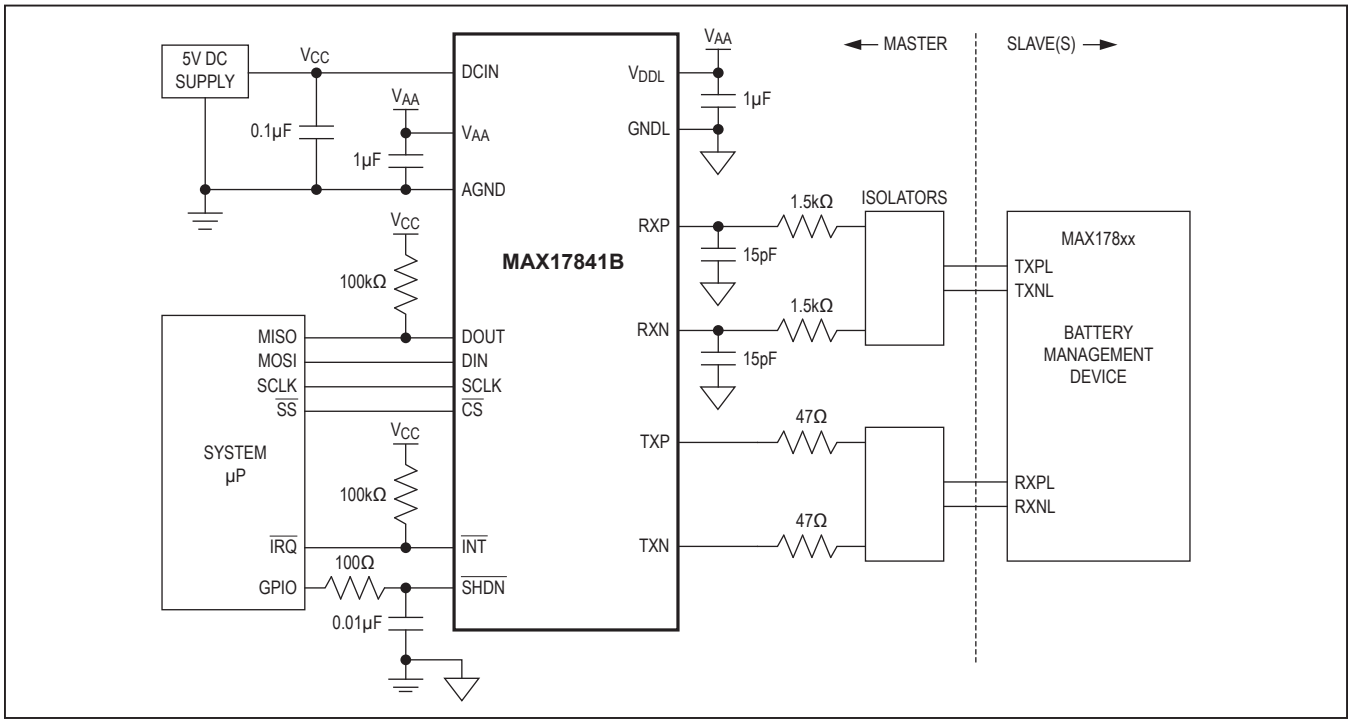
- Battery Management Systems (BMS)
- Electric and Hybrid Vehicles (EV/HEV)
- Energy Storage Systems (ESS)

Benefits and Features

- Supports Maxim's Battery Management UART Protocol
- SPI Interface Up to 4MHz
- UART Baud Rate Programmable Up to 2Mbps
- 3.3V or 5V Operation
- Ultra-Low Quiescent Current
- Transmit and Receive Buffers with Programmable Interrupts Allow for Queuing of UART Messages
- Manchester Encoder and Decoder Reduces Host Controller Burden
- Operating Temperature Range from -40°C to +105°C (AEC-Q100 Type 2)
- Supports ASIL Requirements

***Ordering Information* appears at end of data sheet.**

Simplified Operating Circuit



maxim integrated™

Absolute Maximum Ratings

DCIN to AGND .....-0.3V to +6V
V_{AA} to AGND.....-0.3V to +4V
V_{DDL} to GNDL.....-0.3V to +4V
AGND to GNDL.....-0.3V to +0.3V
TXP, TXN to GNDL..... -0.3V to (V_{DDL} + 0.3V)
DOUT to GNDL .....-0.3V to (V_{DCIN} + 0.3V)
CTG to AGND.....-0.3V to +8V
$\overline{SHDN}$ to AGND .....-0.3V to (V_{DCIN} + 0.3V)
$\overline{CS}$, DIN, SCLK, $\overline{INT}$ to GNDL.....-0.3V to +6V
RXP, RXN to GNDL.....-30V to +30V

Maximum Continuous Current into Any Pin 20mA
Maximum Average Power for ESD Diodes (Note 1) .... 14.4/√τW
Continuous Power Dissipation
On Multilayer Board (T_A = +70°C)
16 TSSOP (derate 11.1mW/ºC above +70ºC).....889mW
Operating Temperature Range..... -40°C to +105°C
Storage Temperature Range..... -55°C to +150°C
Junction Temperature (Continuous) .....+150°C
Soldering Lead Temperature for 10s.....+300°C

Note 1: Average power for time period τ where τ is the time constant (in µs) of the transient diode current during a hot-plug event. For, example, if τ is 330µs, the maximum average power is 0.793W. Peak current must never exceed 2A. Actual average power during hot-plug must be calculated from the diode current waveform for the application circuit and compared to the maximum rating.

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.

Electrical Characteristics

(V_{DCIN} = 5V, V_{AA} = V_{DDL} = 3.3V, T_A = T_{MIN} to T_{MAX}, unless otherwise noted, where T_{MIN} = -40°C and T_{MAX} = +105°C. Typical values are at T_A = +25°C. Operation is with the recommended application circuit.)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
POWER REQUIREMENTS						
Supply Voltage	V_{DCIN}	V_{DCIN} = V_{AA} = V_{DDL} = 3.3V nominal	3.1	3.3	3.5	V
		V_{DCIN} = 5V nominal; V_{AA} = V_{DDL}	4.5	5.0	5.5	
Supply Current	$I_{SHUTDOWN}$	$V_{\overline{SHDN}}$ = 0V, V_{DCIN} = V_{AA} = 3.3V		4	10	µA
		$V_{\overline{SHDN}}$ = 0V, V_{DCIN} = 5V		1	10	
	$I_{STANDBY}$	$\overline{SHDN}$ high, f_{SCLK} = 0, f_{UART} = 0	1.0	2.3	5.0	mA
	I_{ACTIVE}	Continuous SPI writes at 4MHz, 50pF TXP load, 50pF TXN load, f_{UART} = 2Mbps, Transmit Preambles mode	1.0	4	6	
REGULATOR						
Output Voltage	V_{AA}	0mA < I_{VAA} < 10mA, 4.5V < V_{DCIN} < 5.5V	3.13	3.30	3.46	V
Short-Circuit Current	I_{AASC}	V_{AA} = AGND	13.0	26.0	120.0	mA
POR Threshold	$V_{AARESET}$	V_{AA} falling	2.8	2.9	3.0	V
	$V_{AAVALID}$	V_{AA} rising	2.9	3.0	3.1	
POR Hysteresis	V_{AAHYS}			40.0	100	mV
LOGIC INPUTS ($\overline{SHDN}$, $\overline{CS}$, DIN, SCLK)						
Pulldown Resistance ($\overline{CS}$)	$R_{\overline{CS}}$	$V_{\overline{CS}}$ = 5V	5.5	12	28.8	MΩ
Pulldown Resistance ($\overline{SHDN}$)	$R_{\overline{SHDN}}$	$V_{\overline{SHDN}}$ = 5V	0.75	1.5	3.0	MΩ
Input Leakage Current (DIN, SCLK)	I_{LKG}	V_{DIN}, V_{SCLK} = 0V	-1.0		+1.0	µA
Input Low Threshold	V_{IL}				0.3 x V_{DCIN}	V
Input High Threshold	V_{IH}		0.7 x V_{DCIN}			V

Electrical Characteristics (continued)

(V_{DCIN} = 5V, V_{AA} = V_{DDL} = 3.3V, T_A = T_{MIN} to T_{MAX}, unless otherwise noted, where T_{MIN} = -40°C and T_{MAX} = +105°C. Typical values are at T_A = +25°C. Operation is with the recommended application circuit.)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
LOGIC OUTPUTS (DOUT, $\overline{INT}$)						
Output Leakage Current	I_{LKG}	V_{DOUT} = 0 and 5V, $V_{\overline{INT}}$ = 5V	-1.0		+1.0	µA
Output Low Voltage	V_{OL}	I_{OL} = -2mA			V_{GNDL} + 0.4	V
Output High Voltage (DOUT)	V_{OH}	I_{OH} = 2mA	V_{DCIN} - 0.4			V
POWER AND GROUND FAULT DETECTION						
Open Detection Voltage (V_{DDL})	$V_{VDDLALRT}$	V_{AA} = 3.3V	2.8	3.0		V
Open Detection Voltage (GNDL)	$V_{GNDLALRT}$	V_{AGND} = 0V		0.13	0.25	V
Open Detection Voltage (AGND)	$V_{AGNDALRT}$	V_{GNDL} = 0V		0.13	0.25	V
UART INPUTS (RXP, RXN)						
RXP Input Voltage	V_{RXP}		V_{GNDL} - 28		V_{GNDL} + 28	V
RXN Input Voltage	V_{RXN}		V_{GNDL} - 28		V_{GNDL} + 28	V
Differential Input High Threshold	V_{TH}	(Note 2)	$V_{DDL}/2$ - 400mV	$V_{DDL}/2$	$V_{DDL}/2$ + 400mV	V
Differential Input Zero-Crossing Threshold	V_{ZC}	(Note 2)	-400	0	+400	mV
Differential Input Low Threshold	V_{TL}	(Note 2)	$-V_{DDL}/2$ - 400mV	$-V_{DDL}/2$	$-V_{DDL}/2$ + 400mV	V
Differential Input Hysteresis	V_{HYST}	(Note 2)	25	75	150	mV
Common-Mode Voltage Bias	V_{CM}		$V_{DDL}/3$ - 0.1	$V_{DDL}/3$	$V_{DDL}/3$ + 0.1	V
Input Capacitance	C_{IN}			2		pF
Leakage Current	I_{LKG}		-30		+30	µA
Input Resistance to V_{CM}	R_{RXIN}			1.1		MΩ
UART OUTPUTS (TXP, TXN)						
Output Low Voltage	V_{OL}	I_{OL} = -20mA			V_{GNDL} + 0.4	V
Output High Voltage	V_{OH}	I_{OH} = 20mA	V_{DDL} - 0.4			V
SPI TIMING						
SCLK Frequency	f_{SCLK}				4	MHz
$\overline{CS}$ to SCLK Setup Time	t_{CSS}		250			ns
$\overline{CS}$ High Pulse Width	t_{CSWH}		200			ns
SCLK High Time	t_{CH}		100			ns
SCLK Low Time	t_{CL}		100			ns

Electrical Characteristics (continued)

(V_{DCIN} = 5V, V_{AA} = V_{DDL} = 3.3V, T_A = T_{MIN} to T_{MAX}, unless otherwise noted, where T_{MIN} = -40°C and T_{MAX} = +105°C. Typical values are at T_A = +25°C. Operation is with the recommended application circuit.)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
SCLK Fall to DOUT Valid	t_{DO}				30	ns
DIN to SCLK Setup Time	t_{DS}		10			ns
DIN to SCLK Hold Time	t_{DH}		30			ns
UART TIMING						
Bit Period Except for Second STOP Bit (Notes 3, 4)	t_{BIT}	f_{UART} = 2Mbps		8		$1/f_{OSC}$
		f_{UART} = 1Mbps		16		
		f_{UART} = 0.5Mbps		32		
Second STOP Bit Period (Notes 3, 4)	$t_{STOPBIT}$	f_{UART} = 2Mbps		9		$1/f_{OSC}$
		f_{UART} = 1Mbps		18		
		f_{UART} = 0.5Mbps		36		
Rx Idle to START Setup Time (Notes 3, 4)	t_{RXSTSU}	f_{UART} = 2Mbps	0		8	$1/f_{OSC}$
		f_{UART} = 1Mbps	0		16	
		f_{UART} = 0.5Mbps	0		32	
Tx Idle to START Setup Time (Notes 3, 4)	t_{TXSTSU}	f_{UART} = 2Mbps		8		$1/f_{OSC}$
		f_{UART} = 1Mbps		16		
		f_{UART} = 0.5Mbps		32		
STOP Hold Time to Idle (Notes 3, 4)	t_{SPHD}				4	$1/f_{OSC}$
Rx Minimum Idle Time (STOP Bit to START Bit) (Notes 3, 4)	$t_{RXIDLESPST}$	f_{UART} = 2Mbps	8			$1/f_{OSC}$
		f_{UART} = 1Mbps	16			
		f_{UART} = 0.5Mbps	32			
Tx Minimum Idle Time (Notes 3, 4)	$t_{TXIDLESPST}$		10			$1/f_{OSC}$
Rx Fall Time (Notes 3–5)	t_{FALL}	f_{UART} = 2Mbps			4	$1/f_{OSC}$
		f_{UART} = 1Mbps			8	
		f_{UART} = 0.5Mbps			16	
Rx Rise Time (Notes 3–5)	t_{RISE}	f_{UART} = 2Mbps			4	$1/f_{OSC}$
		f_{UART} = 1Mbps			8	
		f_{UART} = 0.5Mbps			16	
Startup Time ($\overline{SHDN}$ High to RXP Valid)	$t_{STARTUP}$			800	2000	µs
Oscillator Frequency	f_{OSC}		15.68	16.00	16.32	MHz
UART MESSAGE TIMING						
SPI Command to Tx Valid Delay (Note 6)	t_{TX}		4 x t_{BIT}		5 x t_{BIT}	

Electrical Characteristics (continued)

(V_{DCIN} = 5V, V_{AA} = V_{DDL} = 3.3V, T_A = T_{MIN} to T_{MAX}, unless otherwise noted, where T_{MIN} = -40°C and T_{MAX} = +105°C. Typical values are at T_A = +25°C. Operation is with the recommended application circuit.)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
Tx Valid to Rx Valid Up Stack Delay (Note 7)	t_{RXUP}				n x t_{PROP}	
Tx Valid to Rx Valid Down Stack Delay (Note 7)	t_{RXDN}				n x t_{PROP}	
End of STOP Character to RX_Stop_INT Flag True (Note 8)	$t_{\overline{INT}}$				2 x t_{BIT}	
SPI START to UART Slave Device Register Write Delay (Notes 9, 10)	t_{REGWR}				8 / f_{SCLK} + 130 x t_{BIT} + n x t_{PROP}	

Note 2: Differential signal (V_{RXP} - V_{RXN}) where V_{RXP}, V_{RXN} do not exceed a common-mode voltage range of ±25V.

Note 3: All parameters measured based on differential signal.

Note 4: Guaranteed by design and not production tested.

Note 5: Fall time measured 90% to 10%, rise time measured 10% to 90%.

Note 6: Measured from falling edge of 8th SCLK cycle of the WR_NXT_LD_Q SPI command byte (B0h).

Note 7: t_{PROP} is the maximum propagation delay through a slave device in a given direction. Refer to the UART slave device data sheet for the actual delay. The number of UART slave devices is denoted by n.

Note 8: Measured from end of 12th bit of stop character.

Note 9: Parameter t_{REGWR} is the minimum amount of time needed to write a register in the nth slave device of the daisy-chain. It is measured from the start of the SPI transaction WR_NXT_LD_Q (B0h) that initiates transmission of a WRITEALL message to when the nth device receives a valid WRITEALL message. For example, for 4MHz SPI frequency, 2Mbps UART baud rate, n = 10 and t_{PROP} = 3 x t_{BIT}, t_{REGWR} = 2µs + 65µs + 15µs = 82µs.

Note 10: Computation of t_{REGWR} consists of three terms: 1) duration of the SPI transaction, 2) partial duration of the UART message, and 3) propagation delay of the UART message. The first term equals the number of bits in the SPI transaction (8) x the SPI bit time (1 / f_{SCLK}). The second term equals the time from the start of the WRITEALL message to the first STOP bit of the last PEC nibble. The last PEC nibble is the 11th character in the message. With each character lasting 12 UART bit times, there are 11 x 12 = 132 bit times from the start of the message to the end of the last PEC nibble. Since the write occurs just before the two STOP bits of the 11th character, the term is actually 130 x t_{BIT}. The third term is the propagation delay required for the WRITEALL message to get to the nth device.

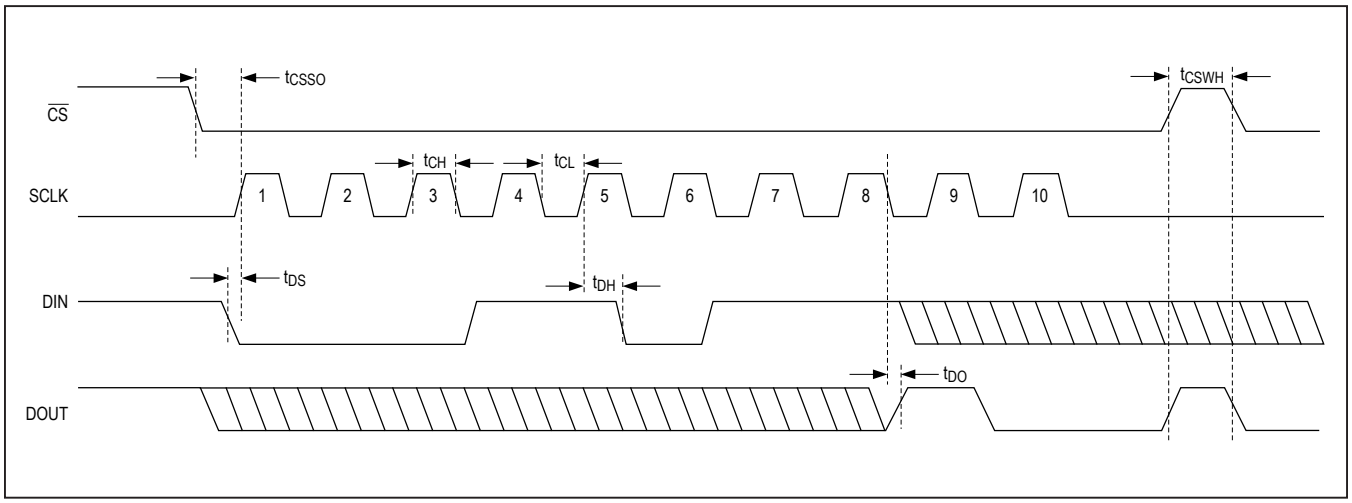


Figure 1. SPI Timing Diagram (Example of Reading Register 0x1B with Data 80h and Transaction Terminated Prematurely)

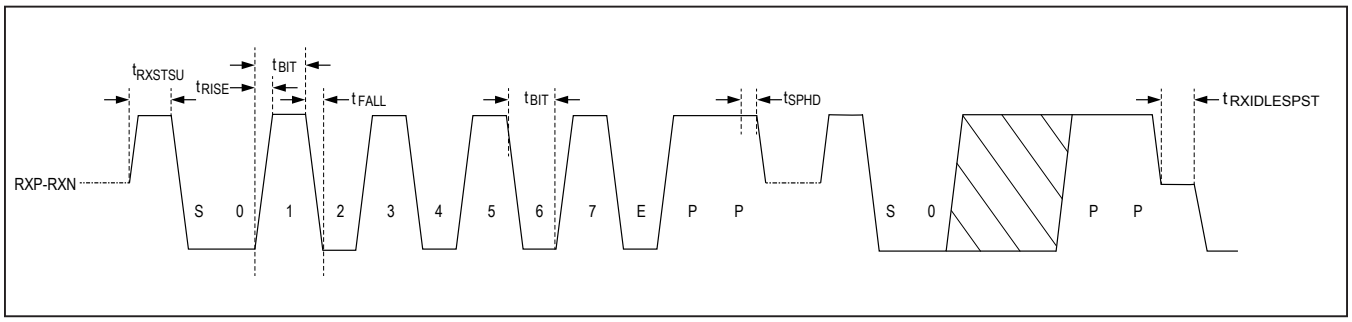


Figure 2. Receive UART Timing

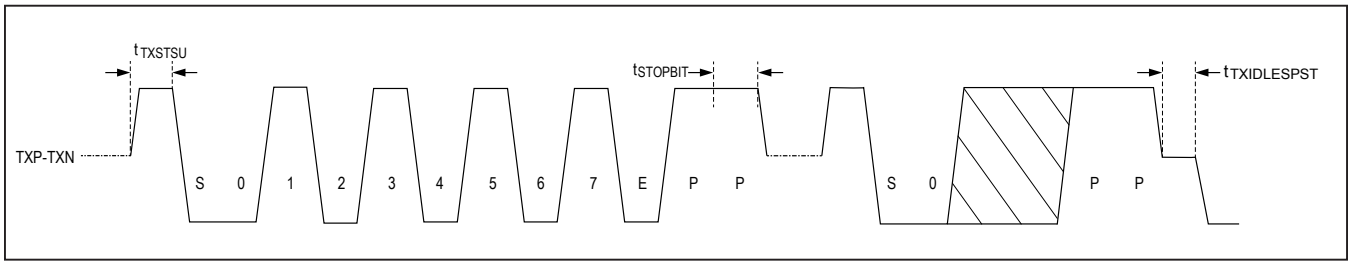


Figure 3. Transmit UART Timing

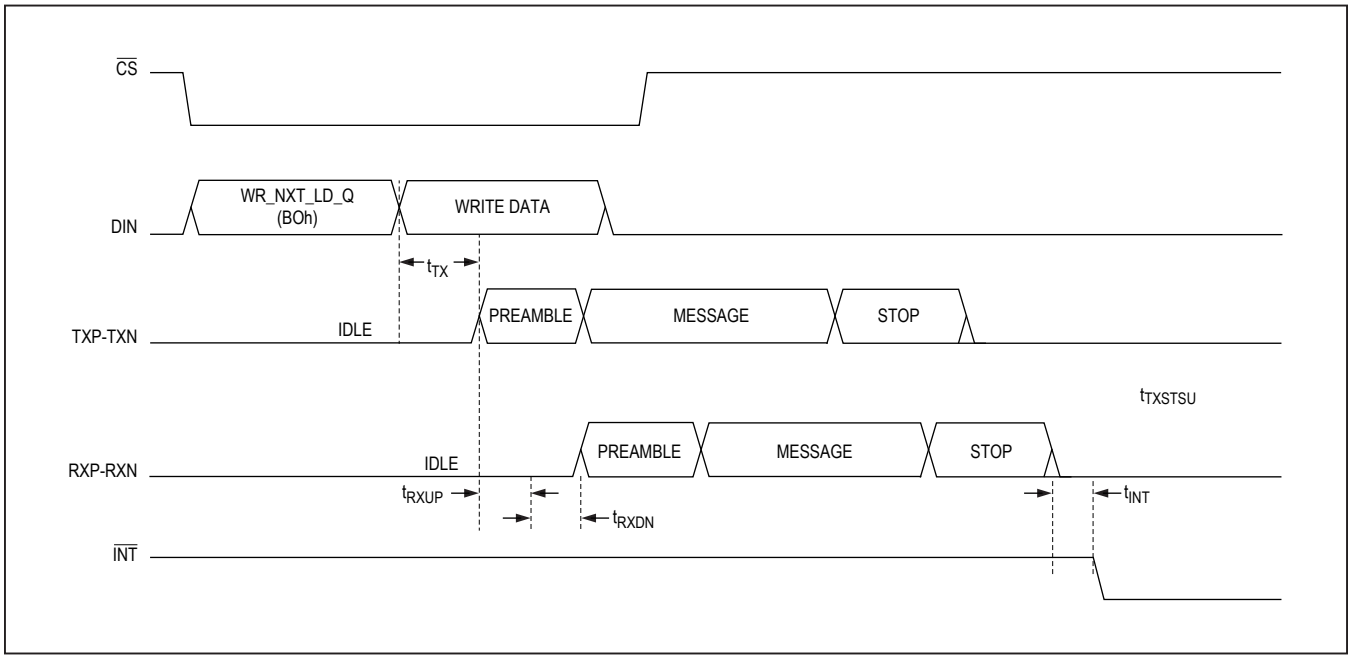
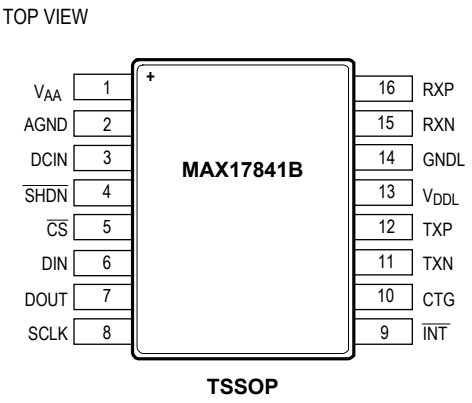


Figure 4. UART Message Timing

Pin Configuration



Pin Description

PIN	NAME	FUNCTION	DESCRIPTION
1	V_{AA}	Power	Power Output for LDO Regulator (5V Mode Only) and Supply for Oscillator. For 5V mode, connect to V_{DDL}. For 3.3V mode, connect this pin to 3.3V supply. Decouple per application circuit.
2	AGND	Ground	Analog Ground. Connect to the power supply ground.
3	DCIN	Power	Power Input for LDO Regulator and SPI Port. For 5V mode, connect to 5V supply. For 3.3V mode, connect this pin to 3.3V supply. Decouple per application circuit.
4	$\overline{SHDN}$	Input	Active-Low Shutdown Input. Connect to host GPIO. Assert to place device in shutdown mode. In this mode, the regulator is disabled and the device is reset. This pin has a 1.5MΩ internal pulldown. 5V tolerant.
5	$\overline{CS}$	Input	Active-Low SPI Chip-Select Input. Connect to the Slave_Select output of the SPI master. Assert to enable the SPI port. This pin has a 12MΩ internal resistor to ground. 5V tolerant.
6	DIN	Input	SPI Data Input. Connect to DOUT/MOSI output of SPI master. 5V tolerant.
7	DOUT	Output	SPI Data Output. Connect to DIN/MISO input of SPI master. This output is three-stated when $\overline{CS}$ is deasserted. When $\overline{CS}$ is asserted, this pin is driven between DCIN and AGND supplies.
8	SCLK	Input	SPI Clock Input. Connect to SCLK output of SPI master. 5V tolerant.
9	$\overline{INT}$	Output	Active-Low, Open-Drain Interrupt Output. Connect a pullup resistor to this pin per application requirements. This pin is asserted if any interrupt flag is set.
10	CTG	Ground	Reserved for factory use. Connect to AGND.
11	TXN	Output	UART Transmitter Negative Output. Connect to Rx port negative input circuit of UART slave device per application circuit. This pin is driven between the V_{DDL} and GNDL supplies.
12	TXP	Output	UART Transmitter Positive Output. Connect to Rx port positive input circuit of UART slave device per application circuit. This pin is driven between the V_{DDL} and GNDL supplies.
13	V_{DDL}	Power	3.3V Digital and UART Port Power. Connect to V_{AA}. Decouple per application circuit.
14	GNDL	Ground	Digital and UART Port Ground. Connect to AGND.

Pin Description (continued)

PIN	NAME	FUNCTION	DESCRIPTION
15	RXN	Output	UART Receiver Negative Input. Connect to Tx port negative output of UART slave device per application circuit.
16	RXP	Output	UART Receiver Positive Input. Connect to Tx port positive output of UART slave device per application circuit.

Detailed Description

The MAX17841B allows any host controller with an SPI port to communicate with one or more battery management slave devices that use Maxim's battery management UART protocol.

Table 1. Internal Power Distribution

BLOCK	SUPPLY
Oscillator	V_{AA}
SPI Port and LDO Regulator	DCIN
Digital, UART, and Control	V_{DDL}

Together with the host controller, the ASCI is the master for communications with the slave devices. Figure 5 shows the functional block diagram. Table 1 shows how power is distributed inside the device.

Serial Peripheral Interface (SPI)

The SPI port is a synchronous data link that the host uses to read and write the ASCI registers and the UART communication buffers.

SPI Transactions

An SPI transaction is initiated when the host drives the $\overline{CS}$ pin low. The host always transmits data most-significant bit (MSB) first to the ASCI. After the first byte, it can termi-

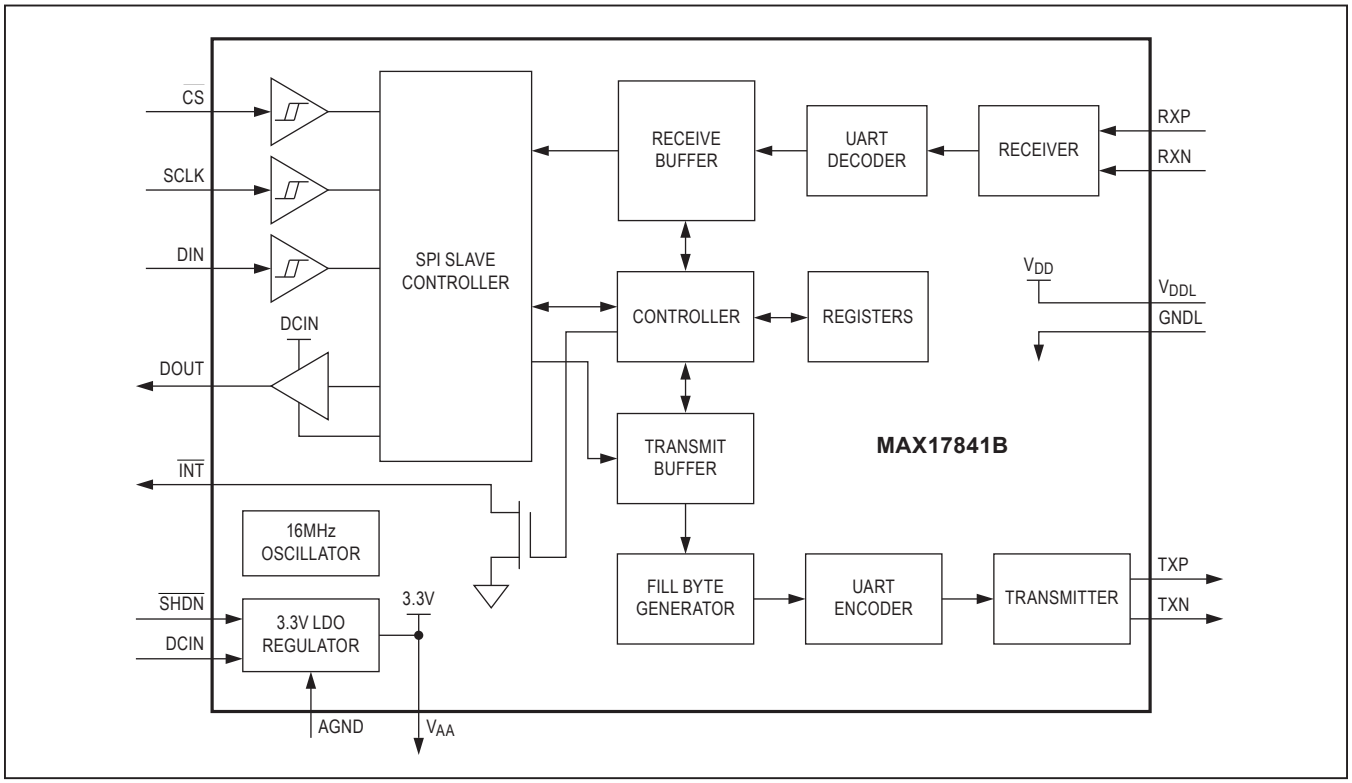


Figure 5. Functional Diagram

nate the transaction (single-byte transaction), continue to clock data out (write transaction), or start clocking data in (read transaction). However, it does not send and receive data at the same time (half-duplex operation).

Register Transactions

For register transactions, the host first sends a single-byte register address. Register addresses are either read-only (odd addresses) or write-only (even addresses). For a read transaction, the second byte is the read data sent by the ASCI to the host. For a write transaction, the second byte is the write data sent by the host to the ASCI. Multiple data bytes are allowed as long as $\overline{CS}$ remains active-low—the ASCI automatically selects the next read-only register address (for reads) or the next write-only address (for writes). The SPI transaction is terminated when the host drives $\overline{CS}$ high.

Buffer Transactions

Buffer transactions can consist of only a command byte, a command byte followed by one or more read bytes, or a command byte followed by one or more write bytes. All allowed transactions are specified in Table 9.

SPI Timing

The ASCI is only compatible with SPI mode 0 (CPOL = 0/CPHA = 0). In this mode, data is always driven on the falling edge of SCLK and is always sampled on the rising edge of SCLK.

For reads, the ASCI starts driving DOUT on the first falling edge of SCLK immediately after the ASCI samples the least-significant bit (LSB) of the command/address byte. DIN is a “don’t care” while reading. Reads attempted beyond the address space return zero.

For writes, registers are written on the falling edge of SCLK, after the last bit is sampled. However, if $\overline{CS}$ goes high before the last bit’s falling edge of SCLK, that register is not written.

Table 2. SPI Communication Summary

PARAMETER	VALUE
Communication Mode	Half-duplex
Maximum Clock Frequency	4MHz
Bit Order	Most-significant bit first
Clock Polarity (CPOL)	0 (leading clock edge is rising edge)
Clock Phase (CPHA)	0 (data sampled on leading clock edge)

UART Interface

Slave devices that use Maxim’s battery management UART protocol can be connected in daisy-chain fashion to manage a multiple battery-cell stack. In a BMS, or Battery Management System, the BMS controller is the host for all slave devices and initiates all communication. The data flow always starts from the host, goes up the daisy-chain and back down to the host as represented in Figure 6.

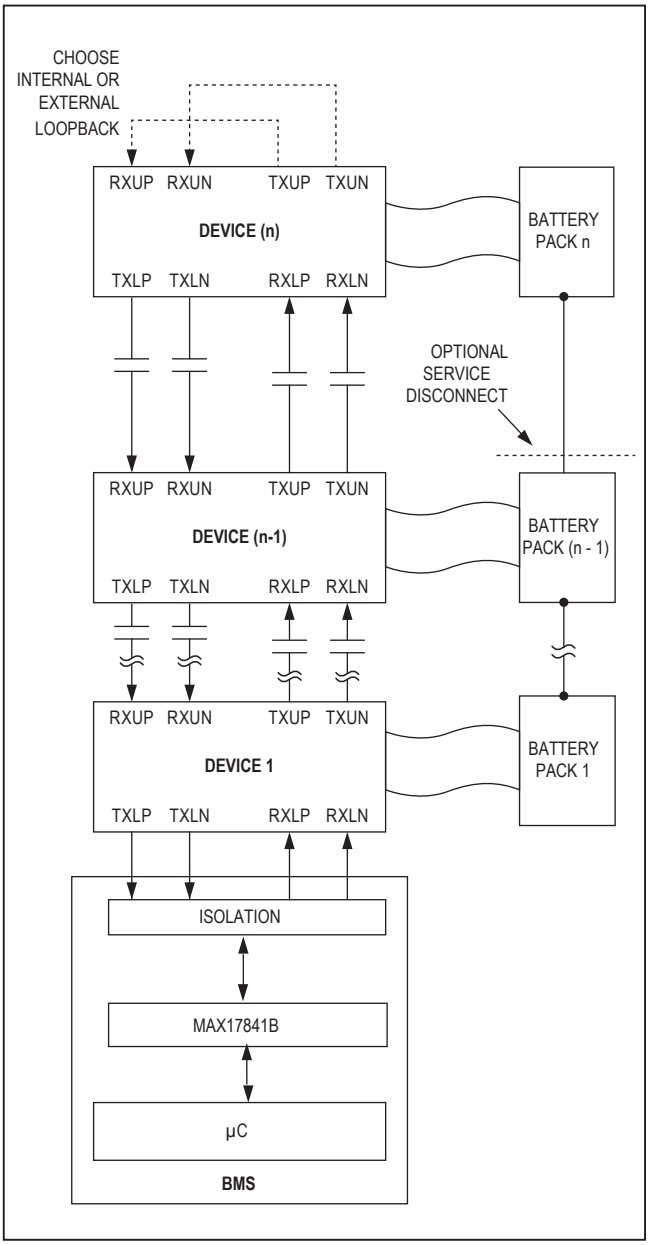


Figure 6. System Data Flow

Battery Management UART Protocol

The ASCI uses a UART protocol specifically designed for Maxim battery management devices. This protocol uses the following features to maximize the integrity of the communications:

- All transmitted data are Manchester-encoded where each data bit is transmitted twice with the second bit inverted (G.E. Thomas convention).
- Every transmitted character contains 12 bits that include a START bit, a parity bit, and two STOP bits.
- Each message contains a CRC-8 packet error-checking (PEC) byte
- Each message is framed by a preamble character and stop character.
- Each received message contains a data-check byte for verifying the integrity of the transmission.

The protocol is also designed to minimize power consumption by allowing slave devices to shut down if the data link is idle for a specified period of time. To prevent the unintentional shutdown of slave devices, the host should enable the ASCI's Keep-Alive mode to periodically transmit stop characters. The time period between stop characters is configurable by the host.

UART Messages

A message is defined as a sequence of UART characters. The message starts with a preamble character, followed by data characters, and ending with a stop character.

Each character consists of the following 12 bits:

- One START bit
- Eight data bits (LSB first)
- One parity bit (even)
- Two STOP bits

Each data byte is transmitted and received as two separate characters, one 12-bit character for each 4-bit data nibble. Each Manchester-encoded nibble actually requires eight data bits: four true bits and four inverted bits.

In its default configuration, when the ASCI transmits a message, it automatically performs the following functions:

- Frames the message with the required preamble character at the beginning of the message.
- Manchester encodes each data nibble and transmits each encoded nibble with the required START, parity, and STOP bits.
- Transmits the message at the configured baud rate of 0.5Mbps, 1Mbps, or 2Mbps.
- Frames the message with the required stop character at the end of the message.

These automatic functions can be disabled by enabling the following special transmit modes:

- Transmit No Preamble mode (eliminates preamble characters)
- Transmit No Stop mode (eliminates stop characters)
- Transmit Raw Data mode (transmits data with no Manchester encoding)
- Receive Raw Data mode (receives data as not Manchester encoded)

Preamble Character

The preamble is a framing character that the UART generates to signal the beginning of a message. It is transmitted as an unencoded 15h, but is still a DC-balanced character. If any bit(s) other than the STOP bits deviate from the unique preamble sequence, the character is not interpreted as a valid preamble, but rather as a data character.

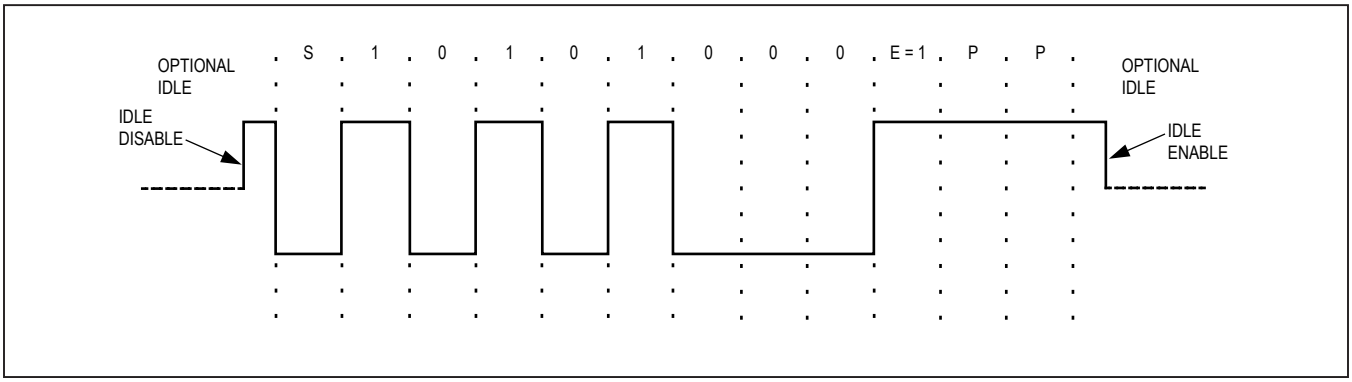


Figure 7. UART Timing for a Preamble

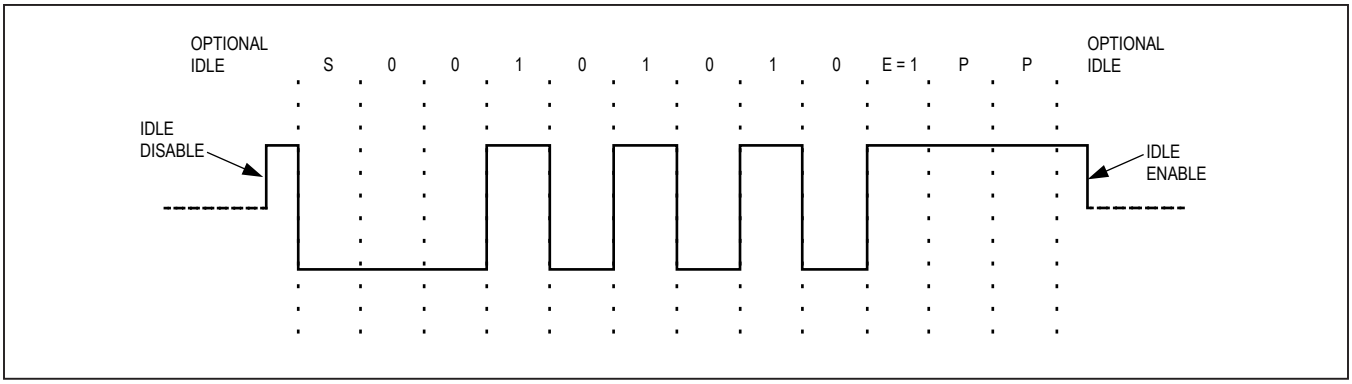


Figure 8. UART Timing for a Stop Character

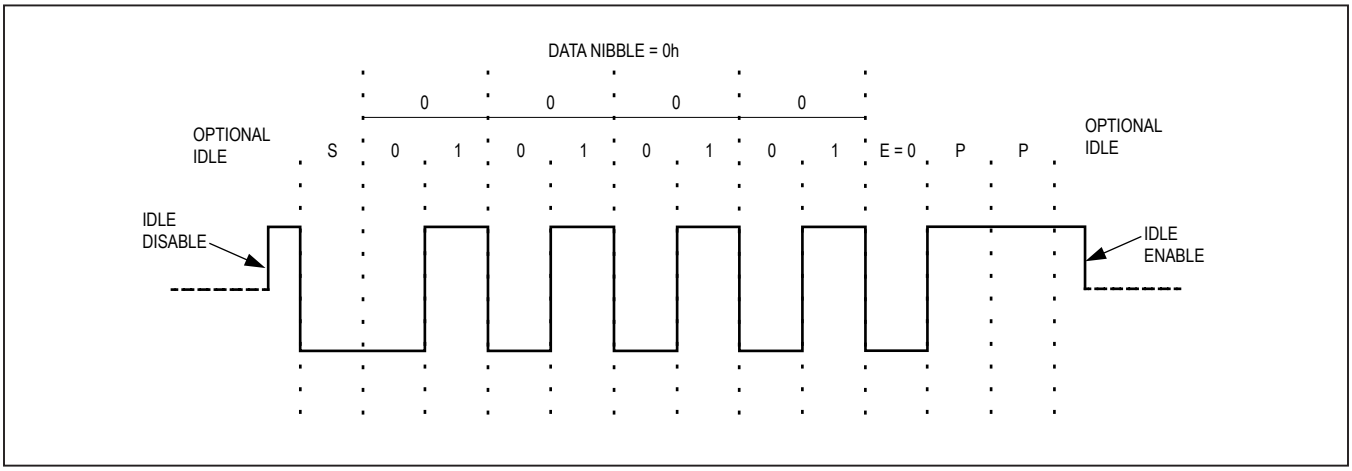


Figure 9. UART Timing for a Manchester-Encoded Data Nibble 0h

Stop Character

The stop character is a framing character that the UART generates to signal the end of a message. It is transmitted as an unencoded 54h, but it is still a DC-balanced character.

Manchester Encoding

Each data byte is transmitted as two separate nibbles (four bits) that are Manchester-encoded. For each data bit, the first bit represents the information and the second bit is its complement. The parity is even so its value should always result in an even number of high bits. Since the data is Manchester-encoded and there are two STOP bits, the parity bit for data characters (but not framing characters) should always be zero.

Data Types

Maxim's battery management UART protocol supports several different data types as described in Table 3. The ASCI does not interpret the significance of any of these data types. It is up to the host to both compose the data being transmitted and interpret the data being received. For example, the host must compute the proper PEC value for each transmitted message and must verify the PEC value on each received message.

Assigning Slave Device Addresses

The battery management UART protocol requires that the host assign a unique and contiguous address between 0 and 31 to each UART slave device so that the host can address each slave device individually as desired. The host performs this assignment by specifying a seed address in the HELLOALL command sequence. As the command propagates up the daisy-chain, each slave device assigns its own address. The HELLOALL sequence returns a value from which the host can determine the number of devices in the daisy-chain as well as the device addresses.

Table 3. Message Data Types

DATA TYPE	DESCRIPTION
Command	Defines the type of message, either a write command or a read command.
Address	Register address to be read or written.
Data	Register data being read or written.
PEC	CRC-8 packet error-checking byte; sent and returned with every message.
Data-Check	Error status provided by the slave devices; returned only on reads.
Alive-Counter	Used to verify the number of devices responding to a transmitted message. This byte is optional but is recommended for error-checking purposes.
Fill	Bytes with values C2h or D3h transmitted as a part of read commands so that the total number of bytes sent equals the number of bytes received. However, these bytes are not returned to receiver with their original values; instead each slave device replaces the fill bytes with the register data being requested by the host.

Table 4. Common Commands

COMMAND BYTE	VALUE	DESCRIPTION
HELLOALL	57h	Assigns a unique device address to each device in the daisy-chain.
WRITEALL	02h	Writes a specific register in all devices.
READALL	03h	Reads a specific register from all devices.

UART Operation

The UART is the subsystem that transmits messages to the UART slave devices and receives them back. The host uses SPI buffer transactions to store unencoded outgoing messages in the transmit buffer and also to read decoded incoming messages out of the receive buffer as shown in Figure 10. Table 5 shows the size and organization of the UART buffers.

UART Interrupts

There are 12 different UART events that can cause an interrupt (refer to the *Register Table* for details). For each event, there is a status bit, an enable bit, and a flag bit. The status bit is the real-time status of the event and can only be set or cleared by the UART. The enable bit determines whether or not the event causes an interrupt. Interrupt flags (except the POR_Flag) are edge-triggered in that they are set only when the interrupt enable bit is true and the corresponding status bit transitions from a logic-zero state to logic-one state. Interrupt flags can only be cleared by the host.

If the flag enable is set when its corresponding status bit is true, the flag is not set until the status bit transitions from a logic-zero state to a logic-one state. If the flag is cleared when the corresponding status bit is true, the flag does not set again until the status bit transitions from a logic-zero state to a logic-one state.

When any flag is true, the UART asserts the $\overline{INT}$ pin. All flags must be cleared for the $\overline{INT}$ pin to be deasserted. The only exception is the POR_Flag, which has no effect on $\overline{INT}$.

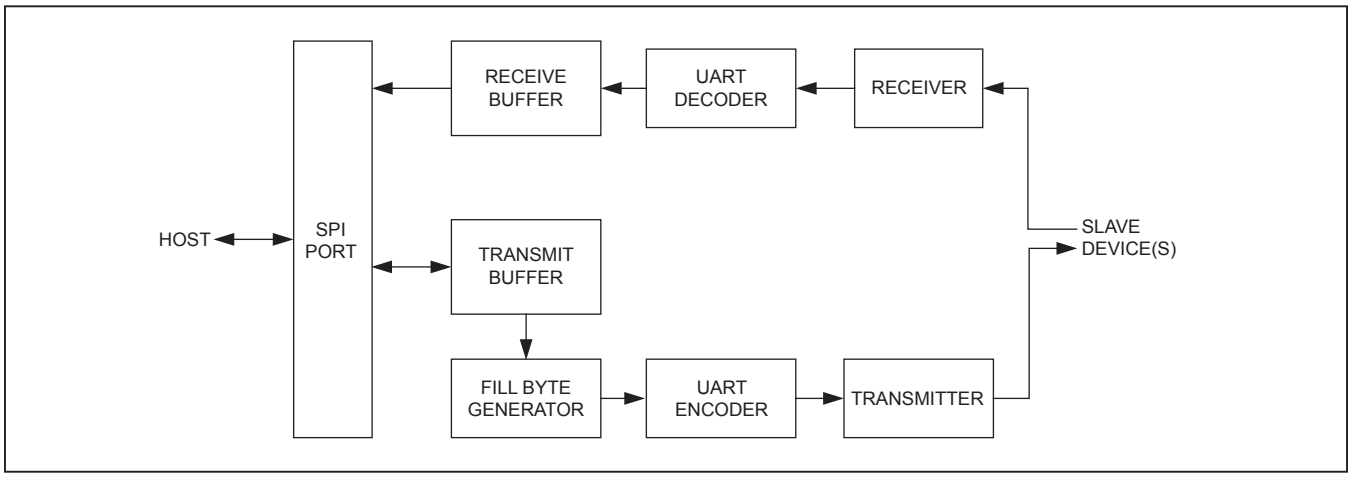


Figure 10. UART Data Flow

Table 5. UART Buffers

PARAMETER	TRANSMIT BUFFER	RECEIVE BUFFER
Organization	4 x 7 bytes	1 x 62 bytes
Size	28 bytes	62 bytes
Message Capacity	4 messages	Variable
Host Access	Read and Write	Read Only

Table 6. UART Operational Modes

MODE	DESCRIPTION
Shutdown	Asserting $\overline{SHDN}$ resets all ASCI registers and buffer data to their default state, stops sending and receiving UART communication, and disables the 3.3V regulator.
Transmit Preambles	Transmits preambles continuously (no idle state). Used to wake up the UART slave devices and initialize the UART baud rate of each slave device. This mode takes precedence over all transmit modes except Transmit Pause mode.
Keep-Alive	Periodically sends a stop character to prevent UART slave devices from shutting down during periods of no communication (idle state). The idle time in between the periodic stop characters is programmable from zero to 10.24ms through the Keep-Alive [3:0] configuration. The default setting is infinite (mode disabled). The Transmit Pause, Transmit Preambles, and the Transmit Queue modes take precedence over this mode.
Transmit Queue (default mode)	Starts transmission of the message loaded in the transmit queue if 1) there is sufficient space in the receive buffer for the message (RX_Full_Status is false) or 2) the limitations on message length are removed (TX_Unlimited is set). Default is enabled.
Transmit Unlimited	In this mode, the transmit queue automatically limits the message length to 255 bytes instead of the default 62-byte limit, and the message transmission is permitted even if the message length is greater than the available write space in the receive buffer.
Transmit Pause	Places the transmitter into idle state once the UART has finished transmitting the current byte, however, the TX_Busy_Status and TX_Idle_Status bits remain unchanged. Transmission resumes when this bit is cleared. This mode takes precedence over all other transmit modes.
Transmit Odd Parity	Transmits characters with odd parity. Since the battery management UART protocol uses even parity, this mode can be used to test the system's ability to detect parity errors. Even parity is default.
Transmit No Stop	Transmits messages without a stop character. By sending subsequent messages with the No Preamble bit, a framed message of indefinite length can be constructed. The TX_Unlimited bit must be set for messages greater than 62 bytes.
Transmit No Preamble	Transmits messages without a preamble. By first sending a message in which the TX_No_Stop bit is set, and then sending messages with this bit set, a framed message of indefinite length can be constructed. However, if the preceding message was terminated with a stop character (end of frame), then the data sent in this mode is unframed (without preamble) and is not stored in the receive buffer.
Transmit Raw Data	Disables Manchester encoding of transmitted data. In this mode, each data byte is transmitted as one character (instead of two characters).
Receive Raw Data	Disables Manchester decoding of the received data. In this mode, there is one data byte stored for every character received (instead of every two received).

Transmit Buffer

The transmit buffer memory map is shown below. It consists of four fixed-length queues, which the host uses to store outgoing messages. At any time, one of the queues is designated as the load queue (the queue being loaded) and one of the queues is designated as the transmit queue (the queue being unloaded). The load queue is selected by the two-bit register LD_Q and the transmit queue is selected by the two-bit register TX_Q. Each queue consists of seven bytes.

Transmit Buffer Queues

In each queue, location 0 is reserved for the message length and the remaining six locations are for specific message data. The default state of each queue is as shown in Table 7.

Clearing the Transmit Buffer

During UART initialization, it is recommended that the host reset the transmit buffer by issuing the CLR_TX_BUF SPI transaction (20h). This resets the transmit buffer as follows:

- TX_Q [1:0] = 00b
- LD_Q [1:0] = 00b
- Data in transmit buffer (28 bytes) is reset to default state per Table 7.

Message Length

Before composing any message, the host should compute the message's length (in bytes, not characters) based on both the type of command (read or write) and the device count. The message length should include any required fill bytes (but not preamble and stop characters). The host writes the message length into location 0 of the load queue, but if the specified message length is greater than 62d, only 62d (3Eh) is actually written. If the TX_Unlimited = 1, then the maximum message length written is increased to 255d (FFh), but the host must service the receive buffer accordingly to avoid any possible overflow.

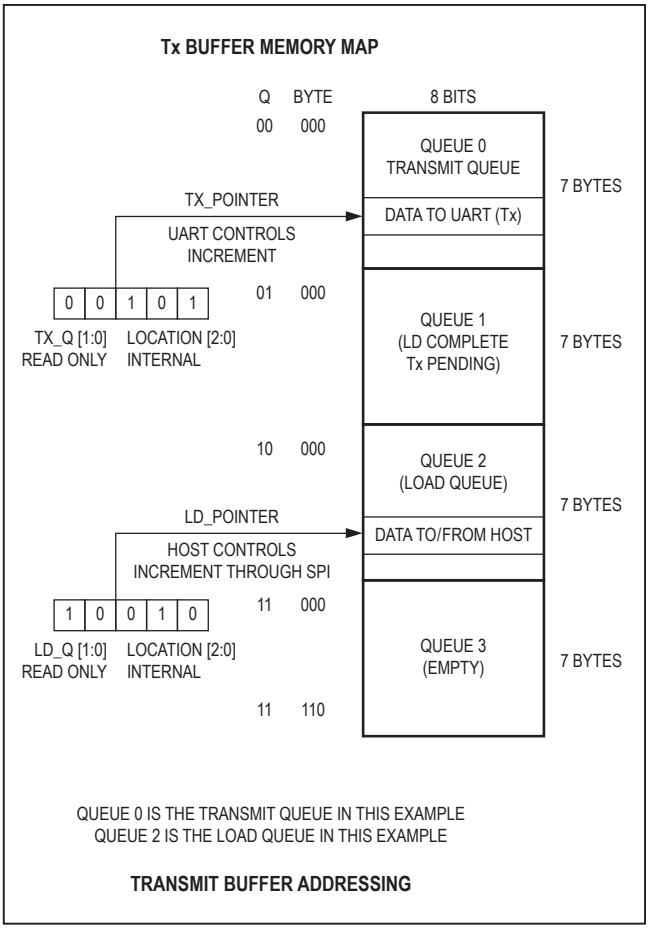


Figure 11. Transmit Buffer Memory Map

If the specified message length is greater than 6 bytes, the UART automatically appends alternating fill bytes (D3h, C2h) as required by the battery management UART protocol during the latter portion of the message transmission.

Table 7. Queue Memory Map

LOCATION	DESCRIPTION	DEFAULT VALUE	MAXIMUM DEFAULT PERMITTED	
			TX_UNLIMITED = 0	TX_UNLIMITED = 1
0	Message length	00h	3Eh	FFh
1	Data bytes and/or fill bytes	D3h	FFh	FFh
2		C2h		
3		D3h		
4		C2h		
5		D3h		
6		C2h		

Writing the Load Queue

A message, not including fill bytes, consists of three (3) to six (6) bytes. The HELLOALL sequence, for example, is three bytes: 57h, 00h, 00h (first address set to zero). Since no fill bytes are required, the total message length is 3 bytes. Therefore, the host should write the load queue with the following data in Table 8.

The host can write the load queue starting at any location within the queue by using appropriate SPI commands listed in SPI transaction Table 9. However, if the host attempts to write beyond location 6 of the queue, the additional data is ignored.

The UART never attempts to transmit the queue selected by LD_Q because the host may be in the process of loading it or, even if it has finished loading, may need to verify (read) the contents of the load queue by using the RD_LD_Q transaction (C1h). The host can then select the next queue in sequence for loading by performing the WR_NXT_LD_Q transaction (B0h), which increments the LD_Q value. It is only when this increment occurs that the UART starts transmitting the data in the previously loaded queue. For both LD_Q[1:0] and TX_Q[1:0], values of 3h increment to 0h.

Table 8. Example of Queue Loaded with Message HELLOALL

LOCATION	VALUE	DESCRIPTION
0	03h	Message length
1	57h	Command byte
2	00h	Address byte
3	00h	Data byte
4	C2h	Not written
5	D3h	Not written
6	C2h	Not written

Filling the Transmit Buffer

The host can load all available queues until LD_Q = TX_Q - 1. In this state, the transmit buffer is full (TX_Full_Status true). In this condition, the host cannot start loading the transmit queue because the UART may still be unloading/transmitting data. If the transmit buffer is full and the host attempts to perform a WR_NXT_LD_Q transaction and thus attempts to load the transmit queue, the increment does not occur and an overflow condition is indicated (TX_Overflow_Status true). The only time the host can write the transmit queue is when the transmit buffer is empty (TX_Q = LD_Q), which is the default state. This state can also occur when the UART finishes sending the last loaded message and thus creates an empty transmit buffer.

Message Transmission

Whenever LD_Q = TX_Q, the transmit buffer is considered empty (TX_Empty_Status is true) because either the host has not yet finished loading the selected queue or the host has written but not yet verified the queue. However, once the host is finished servicing the queue, it performs a WR_NXT_LD_Q transaction to select the next queue. Once this occurs, the transmit buffer is no longer considered empty because LD_Q ≠ TX_Q.

The UART unloads/transmits the transmit queue if the following conditions are met:

- The UART is in Transmit_Queue mode (TX_Queue bit is set)
- The transmit buffer has at least one loaded queue (TX_Empty_Status is false)
- There is sufficient space in the receive buffer for the message (RX_Space_ ≥ Message Length)

Note: The limitation on available space in the receive buffer can be removed by setting the TX_Unlimited bit.

Once the transmit conditions are met, the UART automatically starts unloading the transmit queue until the entire message, including any required fill bytes, has been transmitted. After the transmission is complete, the contents of the transmit queue are reset to their default values and the queue is once again available to the host for loading.

Receive Buffer

The receive buffer is a 62-byte circular buffer that the host can read with the SPI, but can only be loaded by the UART as it receives data. It utilizes three pointers as shown in the receive buffer memory map (Figure 12).

- RX_RD_POINTER: Read pointer or buffer location to be read by host (default 00h, read-only)
- RX_WR_POINTER: Write pointer or buffer location to be written by UART (default 01h, read-only)
- RX_NXT_MSG_POINTER: Buffer location that is start of next unread message (default 00h, read-only)

In the default state, where the read pointer is one less than the write pointer, the receive buffer is considered empty (RX_Empty_Status is true). Any receive buffer data read in this condition will be zero.

Clearing the Receive Buffer

During UART initialization, it is recommended that the host clear the receive buffer by issuing the CLR_RX_BUF SPI transaction (E0h). This resets the receive buffer as follows:

- RX_RD_Pointer: 00h
- RX_WR_Pointer: 01h
- RX_NXT_MSG_POINTER: 00h
- Data in receive buffer (62 bytes) is cleared to 00h

If the receive buffer is cleared during Transmit Preambles mode, the state of the buffer cannot be guaranteed. Therefore, after disabling the Transmit Preambles mode, the host should wait until all transmitted preambles have been received before clearing the buffer. Since the first keep-alive stop character received after the last preamble results in a null message, the host can simply wait until the buffer is no longer empty (RX_Empty_Status = 0) before clearing the buffer.

The RX Clear Buffer command acts an an asynchronous reset not only for the RX Buffer, but also for the UART receiver logic. When the UART receiver logic is reset, it must resynchronize to the incoming UART signal before bytes can be properly processed. This is accomplished by receiving either a preamble byte or an idle state lasting at least one UART byte period. The preamble used to resynchronize the data stream should not be the same preamble that is at the beginning of the next transmitted message. The application should make sure that one of these conditions is met following an RX buffer clear prior to sending the next message from the MAX17841B.

Receiving Messages

UART messages are framed with the preamble and stop characters. If the UART receiver decodes a valid preamble, it prepares to receive a message but it does not store the preamble in the receive buffer. Once data is received, the buffer is no longer empty (RX_Empty_Status = 0) and the UART sequentially stores decoded data bytes in the receive buffer until either a stop character or another preamble is received. When the stop character is received at the end of a message, the UART stores it in the receive buffer as a null byte (00h) and sets the RX_Stop_Status bit. The RX_Stop_Status bit is subsequently cleared when all unread messages have been read (buffer empty) or the next preamble is detected. The host can set the RX_Stop_INT_Enable bit and monitor the interrupt line to determine when to service the receive buffer.

When the host services the receive buffer, three bits in the RX_Byte register indicate specific information about the byte being read (the byte addressed by RX_RD_Pointer), which is useful for error checking:

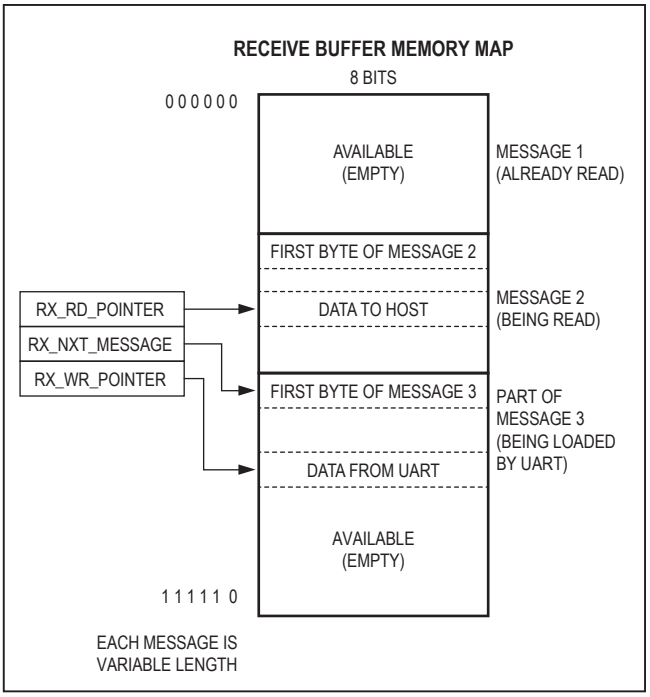


Figure 12. Receive Buffer Memory Map

- First_Byte bit: Indicates that the byte is the first data byte in a message (the corresponding character was preceded by preamble character).
- Byte_Error bit: Indicates that the byte may contain an error (the corresponding character contained a Manchester and/or parity error). This bit drives the RX_Error interrupt.
- Last_Byte bit: Indicates that the byte is the last byte in a message (the corresponding character was a stop character and was stored as a null byte).

Message Exceptions

If a message is not framed with a valid preamble, then the UART ignores the data and does not store it.

If a message is not framed with a stop character, then the preamble of the next message serves to delineate between the two messages. However, the first message has no stop character stored.

If the UART receives a preamble followed by a stop character it stores a null message in the receive buffer consisting of a single null byte (00h). This occurs when a keep-alive stop character is received after Transmit Preambles mode is disabled. In this use case, the receive

buffer is not empty. The host should dispense of the null message by either clearing the receive buffer or by simply reading the null message and discarding it.

A receive buffer overflow occurs when the UART receives data but there is no more space to store it. This could potentially occur if TX_Unlimited was set or if there was sufficient latency in the daisy-chain. The UART cannot overtake the read pointer and overwrite the data being read so the last address that can be written is the one just behind the read pointer. If more data is received after the last address is written, the UART simply overwrites the last address and then sets the RX_Overflow_Status bit. The RX_Overflow_Status bit is cleared when the receive buffer is read, thereby creating more write space. To detect any overflow, the status must be checked before servicing the receive buffer. After servicing the receive buffer, the status

Table 9. SPI Transactions

REGISTER TRANSACTIONS		
ADDRESS	**NAME**	**DESCRIPTION**
0x01 to 0x1B and 0x95 to 0x9B	See the *Register Table*	Reads or writes the specified ASCI register
BUFFER TRANSACTIONS		
COMMAND	**START LOCATION**	**DESCRIPTION**
0x20	—	CLR_TX_BUF Command: Resets the transmit buffer to its default state and clears TX_Q and LD_Q.
0x91	RX_RD_ Pointer	RD_MSG Command: Reads the receive buffer starting at the address RX_RD_ Pointer. Automatically increments the read pointer after the byte is read but does not increment the read pointer into the next message.
0x93	RX_NXT_MSG_ Pointer	RD_NXT_MSG Command: Reads the receive buffer starting at the address RX_ NXT_MSG_Pointer (oldest unread message). Automatically increments the read pointer after the byte is read but does not increment the read pointer into the next message.
0xB0	LD_Q Location 0	WR_NXT_LD_Q Command: Increments LD_Q, then writes the transmit buffer load queue. The increment occurs whether the host loads the data or not. The command byte defines the first location to be written (locations 0 to 6). For example, 0xB0 starts writing at location 0 and continues through location 6. Writes beyond location 6 have no effect.
0xB2	LD_Q Location 1	
0xB4	LD_Q Location 2	
0xB6	LD_Q Location 3	
0xB8	LD_Q Location 4	
0xBA	LD_Q Location 5	
0xBC	LD_Q Location 6	
0xC0	LD_Q Location 0	WR_LD_Q Command: Writes the transmit buffer load queue. The command byte defines the first byte written (locations 0 to 6). For example, 0xC0 starts writing at location 0 and continues through location 6. Writes beyond location 6 have no effect.
0xC2	LD_Q Location 1	
0xC4	LD_Q Location 2	
0xC6	LD_Q Location 3	
0xC8	LD_Q Location 4	
0xCA	LD_Q Location 5	
0xCC	LD_Q Location 6	
0xC1	LD_Q Location 0	RD _LD_Q Command: Reads transmit buffer load queue. The command byte defines the first byte read (locations 0 to 6). For example, 0xC1 starts reading at location 0 and continues through location 6. Reading beyond location 6 reads zeros.
0xC3	LD_Q Location 1	
0xC5	LD_Q Location 2	
0xC7	LD_Q Location 3	
0xC9	LD_Q Location 4	
0xCB	LD_Q Location 5	
0xCD	LD_Q Location 6	
0xE0	—	CLR_RX_BUF Command: Resets the receive buffer and the receive buffer pointers to their default state.

should be checked again for data errors (e.g., parity errors) prior to initiating transmission of the next message.

If multiple messages are received without being read, then an overflow can occur and the UART sets the RX_Overflow_Status bit. This occurs when the write pointer has incremented until it is one less than the read pointer, at which point the UART no longer increments it. In this case, the last data byte is overwritten.

Reading Messages

The host can use two different SPI transactions to read the receive buffer:

- RD_RX_BUF (91h): Starts reading at the current read pointer location
- RD_NXT_MSG (93h): Starts reading at the start of the next unread message

During any read transaction, the host may continue reading data until the end of the message, after which the data read will be 00h. The host cannot continue reading into the next message, if there is one.

During any read transaction, the UART increments the read pointer after the data byte is read so that if the SPI transaction is prematurely terminated in the middle of the byte, then the same location is resent on the next RD_MSG SPI transaction. This allows the host to stop a read and restart it without losing data. Each byte in the buffer is cleared after it is read and is eventually available to the UART for storing incoming data.

Applications Information

Transaction Sequence for UART Initialization

In the example shown in Table 10, the host initializes communication with two UART slave devices. $\overline{SHDN}$ must be deasserted first. Transactions to poll RX_STATUS register are repeated until the poll is successful or times out.

It is recommended that all writes to configuration registers be verified by reading back the register data. Transmit buffer data can be verified by reading the buffer contents or by reading the transmitted data in the receive buffer.

Transaction Sequence for UART Write and Read

In the example shown in Table 11, the host communicates with two UART slave devices to:

- Write the value B2B1h to the device register address 0x12 for all slave devices using a WRITEALL command sequence
- Read back the value B2B1h from the device register address 0x12 for all slave devices using READALL command sequence.

This example assumes that the slave devices have been configured with the alive counter enabled. To execute these two command sequences, the host performs the SPI transactions listed in Table 11.

Table 10. UART Daisy-Chain Initialization Sequence

DIN	DOUT	DESCRIPTION
TRANSACTION 1		Enable Keep-Alive mode (prior to the UART slave wake-up to prevent shutdown)
10h	xxh	Write Configuration 3 register
05h	xxh	Set keep-alive period to 160µs
TRANSACTION 2		Enable Rx Interrupt flags for RX_Error and RX_Overflow
04h	xxh	Write RX_Interrupt_Enable register
88h	xxh	Set the RX_Error_INT_Enable and RX_Overflow_INT_Enable bits
TRANSACTION 3		Clear receive buffer
E0h	xxh	Clear receive buffer
TRANSACTION 4		Wake-up UART slave devices (transmit preambles)
0Eh	xxh	Write Configuration 2 register
30h	xxh	Enable Transmit Preambles mode
TRANSACTION 5		Wait for all UART slave devices to wake up (poll RX_Busy_Status bit)
01h	xxh	Read RX_Status register (RX_Busy_Status and RX_Empty_Status should be true)
xxh	21h	If RX_Status = 21h, continue. Otherwise, repeat transaction until true or timeout.
TRANSACTION 6		End of UART slave device wake-up period
0Eh	xxh	Write Configuration 2 register
10h	xxh	Disable Transmit Preambles mode

Table 10. UART Daisy-Chain Initialization Sequence (continued)

DIN	DOUT	DESCRIPTION
TRANSACTION 7		Wait for null message to be received (poll RX_Empty_Status bit)
01h	xxh	Read RX_Status register
TRANSACTION 8		Clear transmit buffer
20h	xxh	Clear transmit buffer
TRANSACTION 9		Clear receive buffer
E0h	xxh	Clear receive buffer
TRANSACTION 10		Load the HELLOALL command sequence into the load queue
C0h	xxh	WR_LD_Q SPI command byte (write the load queue)
03h	xxh	Message length
57h	xxh	HELLOALL command byte
00h	xxh	Register address (0x00)
00h	xxh	Initialization address of HELLOALL
TRANSACTION 11		Verify contents of the load queue
C1h	xxh	RD_LD_Q SPI command byte
xxh	03h	OK
xxh	57h	OK
xxh	00h	OK
xxh	00h	OK
TRANSACTION 12		Transmit HELLOALL sequence
B0h	xxh	WR_NXT_LD_Q SPI command byte (write the next load queue)
TRANSACTION 13		Poll RX_Stop_Status bit
01h	xxh	Read RX_Status register
xxh	12h	If RX_Status[1] is true, continue. If false, then repeat transaction until true.
TRANSACTION 14		Service receive buffer. Read the HELLOALL message that propagated through the daisy-chain and was returned back to the ASCI. The host should verify the device count.
93h	xxh	RD_NXT_MSG SPI transaction
xxh	57h	Sent command byte (HELLOALL)
xxh	00h	Sent address = 00h
xxh	02h	Returned address = 02h
TRANSACTION 15		Check for receive buffer errors
09h	xxh	Read RX_Interrupt_Flags register
xxh	00h	If no errors, continue. Otherwise, clear and go to error routine.

Table 11. Transaction Sequence for UART Write and Read

SPI DIN	SPI DOUT	DESCRIPTION
TRANSACTION 1		Load the WRITEALL command sequence into the load queue
C0h	xxh	WR_LD_Q SPI command byte
06h	xxh	Message length = 6
02h	xxh	WRITEALL command byte
12h	xxh	Register address of the device
B1h	xxh	LS byte of register data to be written
B2h	xxh	MS byte of register data to be written
C4h	xxh	PEC byte for 02h, 12h, B1h, B2h
00h	xxh	Alive-counter byte (seed value = 0)
TRANSACTION 2		Start transmitting the WRITEALL sequence from the transmit queue
B0h	xxh	WR_NXT_LD_Q SPI command byte
TRANSACTION 3		Check if a message has been received into the receive buffer
01h	xxh	Read RX_Status register
xxh	12h	If RX_Status[1] is true, continue. Otherwise, repeat until true or timeout.
TRANSACTION 4		Read receive buffer to verify the sent WRITEALL message
93h	xxh	RD_NXT_MSG SPI
xxh	02h	Sent command byte (WRITEALL)
xxh	12h	Sent address
xxh	B1h	Sent LS byte
xxh	B2h	Sent MS byte
xxh	C4h	Sent PEC
xxh	02h	Alive-counter byte (= sent seed + 2, if alive counter enabled)
TRANSACTION 5		Check for receive buffer errors
09h	xxh	Read RX_Interrupt_Flags register
xxh	00h	If no errors, continue. Otherwise, clear and go to error routine.
TRANSACTION 6		Load the READALL command sequence into the load queue
C0h	xxh	WR_NXT_LD_Q SPI command byte
09h	xxh	Message length (5 + 2 x n = 9)
03h	xxh	READALL command byte
12h	xxh	Register address
00h	xxh	Data-check byte (seed value = 00h)
CBh	xxh	PEC byte for bytes 03h, 12h, 00h
00h	xxh	Alive-counter byte (seed value = 00h)
TRANSACTION 7		Start transmitting the READALL sequence
B0h	xxh	WR_NXT_LD_Q SPI command byte
TRANSACTION 8		Check if a message has been received into the receive buffer
01h	xxh	Read the RX_Status register
xxh	12h	If RX_Status[1] is true, continue. Otherwise, repeat until true or timeout.

Table 11. Transaction Sequence for UART Write and Read (continued)

SPI DIN	SPI DOUT	DESCRIPTION
TRANSACTION 9		Read the receive buffer and verify that the device register data is what was written during the WRITEALL sequence
93h	xxh	RD_NXT_MSG SPI command byte
xxh	03h	Sent command byte (READALL)
xxh	12h	Sent register address
xxh	B1h	LS byte of device 1
xxh	B2h	MS byte of device 1
xxh	B1h	LS byte of device 0
xxh	B2h	MS byte of device 0
xxh	00h	Data-check byte (= 00h if all status bits have been cleared)
xxh	67h	PEC (for the previous 7 bytes)
xxh	02h	Alive-counter byte (= sent seed + 2, if alive counter is enabled)
TRANSACTION 10		Check for receive buffer errors
09h	xxh	Read RX_Interrupt_Flags register
xxh	00h	If no errors, continue. Otherwise, clear and go to error routine.

Error Checking

It is highly recommended that the host utilize the various error checking features available in both the ASCI and the battery management UART protocol to ensure the integrity of the data being received. The host should implement the following verifications:

- Verification of write data received (matching values, number of bytes)
- Verification of read data received (allowed values, ranges, number of bytes)
- Verification of the received PEC, data-check, and alive-counter bytes
- Verification of ASCI FMEA register
- Verification of the ASCI status bits (RX_Error_Status, RX_Overflow_Status, TX_Overflow_Status, POR_Flag)

Corrupted Preamble Character

If the preamble for a message is corrupted, none of the message is entered into the receive buffer. To detect this failure mode, the host should always verify that any message that it transmitted was also received into the receive buffer. In the case where the host is polling a register (identical messages) then the host can uniquely identify each message sent by sending up to 256 different seed values for the alive-counter byte. The host should increment the seed value every time a message is sent by the host so that its propagation through the daisy chain can be verified in the received data. The alive-counter byte is sent after the PEC byte and therefore the PEC is not affected.

Corrupted Message Content

Manchester, parity, and PEC errors are indications that the data in the message may have been corrupted. For each UART message received, the host should perform the appropriate computations on any error-checking bytes that may be available in the received message:

- Data-Check byte: Error status provided by the slave device(s); sent and returned on reads of slave device data as described in the slave device data sheet.
- PEC byte: CRC-8 packet error-checking byte provided by the slave device(s); sent and returned with every message as described in the slave device data sheet.
- Alive-Counter byte: Used to verify the number of devices responding to a transmitted message; can be sent and received with every message as described in the slave device data sheet.

The host should also set the ASCI's RX_Error_INT_Enable bit. If, in the course of reading the receive buffer, the ASCI sets the RX_Error_INT_Flag, it means that the UART had detected a Manchester and/or parity error in at least one of the received characters in the message. Because Manchester and parity errors can be introduced anywhere in the UART data stream, the errors denoted by the RX_Error_INT_Flag are not necessarily reflected in the error-checking bytes returned by the slave device(s). Therefore, the host should check and clear this flag after (but not before) reading each message in the receive buffer.

Corrupted or Missing Stop Character

If a stop character is corrupted or missing, there is no data loss because the message is still framed either by a subsequent valid stop character (that is automatically sent in Keep-Alive mode) or by the preamble of the next UART message. A corrupted stop character can be interpreted as a data character and would be stored as such in error. In this case, if a valid stop character is eventually received before the next preamble, the message length is one byte too long. The host should check for this condition by computing the received message length and comparing it to the expected message length.

Before reading the next message, the host should also check the RX_Byte register to verify that the last character received was a valid stop character, in which case all of the following are true:

1) The last byte in the message is a null byte (00h).
2) The Last_Byte bit is set.
3) The Byte_Error bit is cleared.

If a stop character was not received then the Last_Byte bit is not set.

Unintended Preamble

The presence of an unintended preamble in the middle of a message creates an unintended message in the receive buffer. If the unintended preamble is the result of a corrupted data character within a message, then the message is prematurely terminated and a second, unexpected message is created. This event can be detected by comparing the number of received bytes in the message to the expected number.

Unintended Stop Character

The presence of an unintended stop character prematurely terminates the message. This is detected by comparing the number of received bytes in the message to the expected number.

UART Physical Layer

Single-Ended Mode

By default, UART ports are configured for differential communication. For single-ended operation, the host can set the Single_Ended_Mode configuration bit. This mode enables the UART to receive a single-ended signal by shifting the input threshold negative so that zero differential voltage is a logic one. The RXP input is connected to ground and the RXN input receives the inverted signal, just as it does for differential mode. In this mode, the Tx port operates the same as in differential mode.

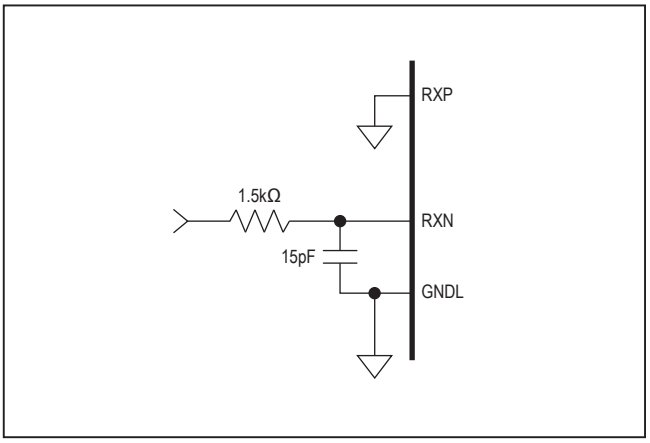


Figure 13. Single-Ended Mode

UART Transformer Coupling

The UART signals can be transformer-coupled because of the DC-balanced signaling. Placing an isolation transformer between the UART's transmitter and the slave device's receiver provides common-mode isolation for the case where the slave device is operating at a different voltage level.

When no data is being transmitted (idle state), the transmitter drives both outputs to a logic low level to prevent any current flow through the transformer winding.

A common-mode noise filter can be implemented by capacitively coupling the center tap of the transformer on the transmitter side to the UART ground. Any common-mode noise that passes through the transformer is effectively shunted to ground.

UART Supplemental ESD Protection

The UART transmitter and receiver, with supplemental protection diodes as shown in the application circuits, can be used for enhanced ESD protection. The diodes should be placed as close as possible to the external connector.

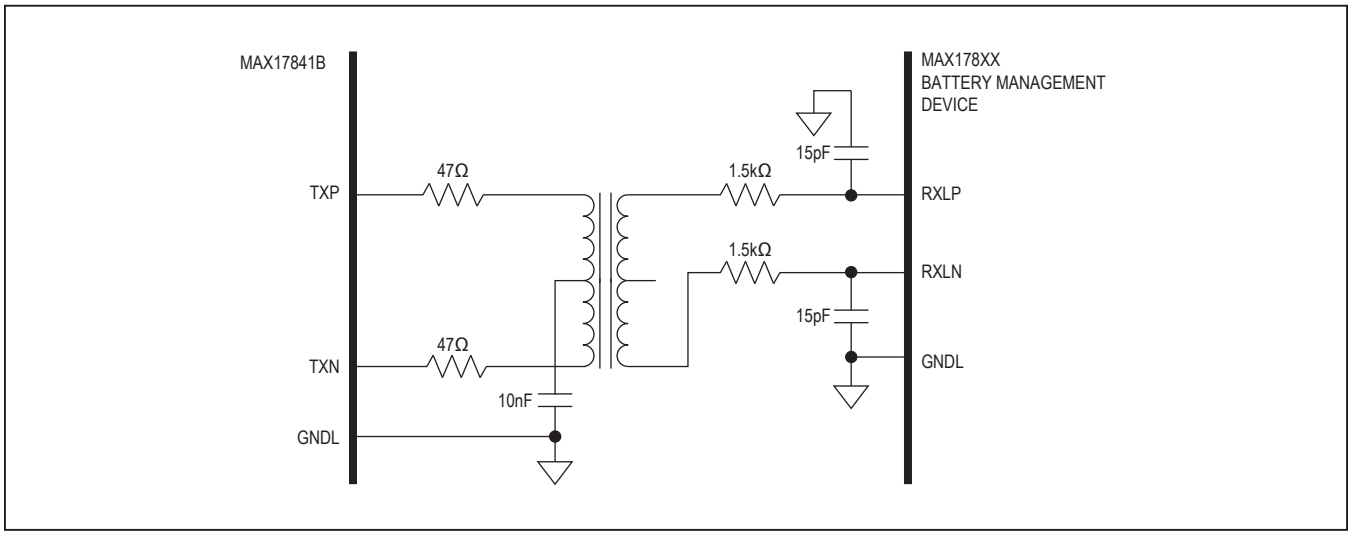


Figure 14. Transformer Coupling of UART Signals

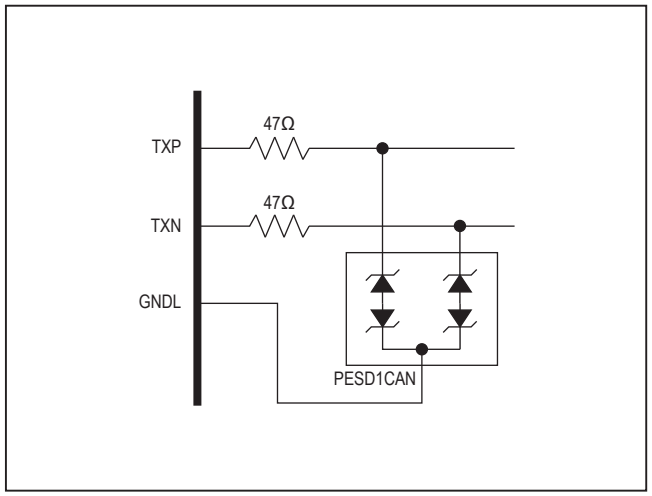


Figure 15. Supplemental ESD Protection for UART Transmitter

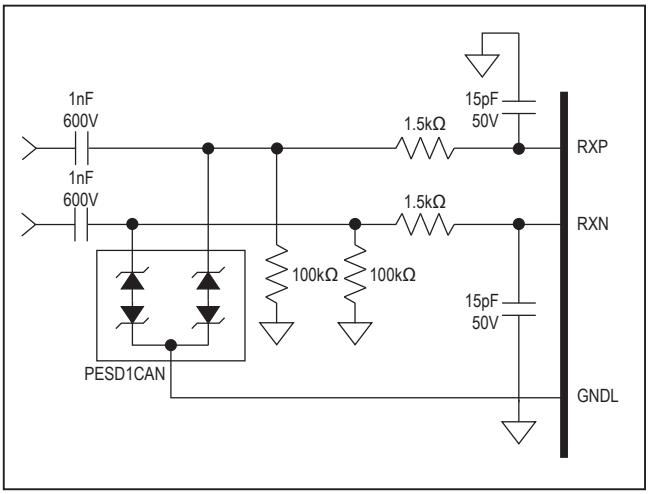


Figure 16. Supplemental ESD Protection for UART Receiver (Shown with Capacitive Coupling)

Internal ESD Protection

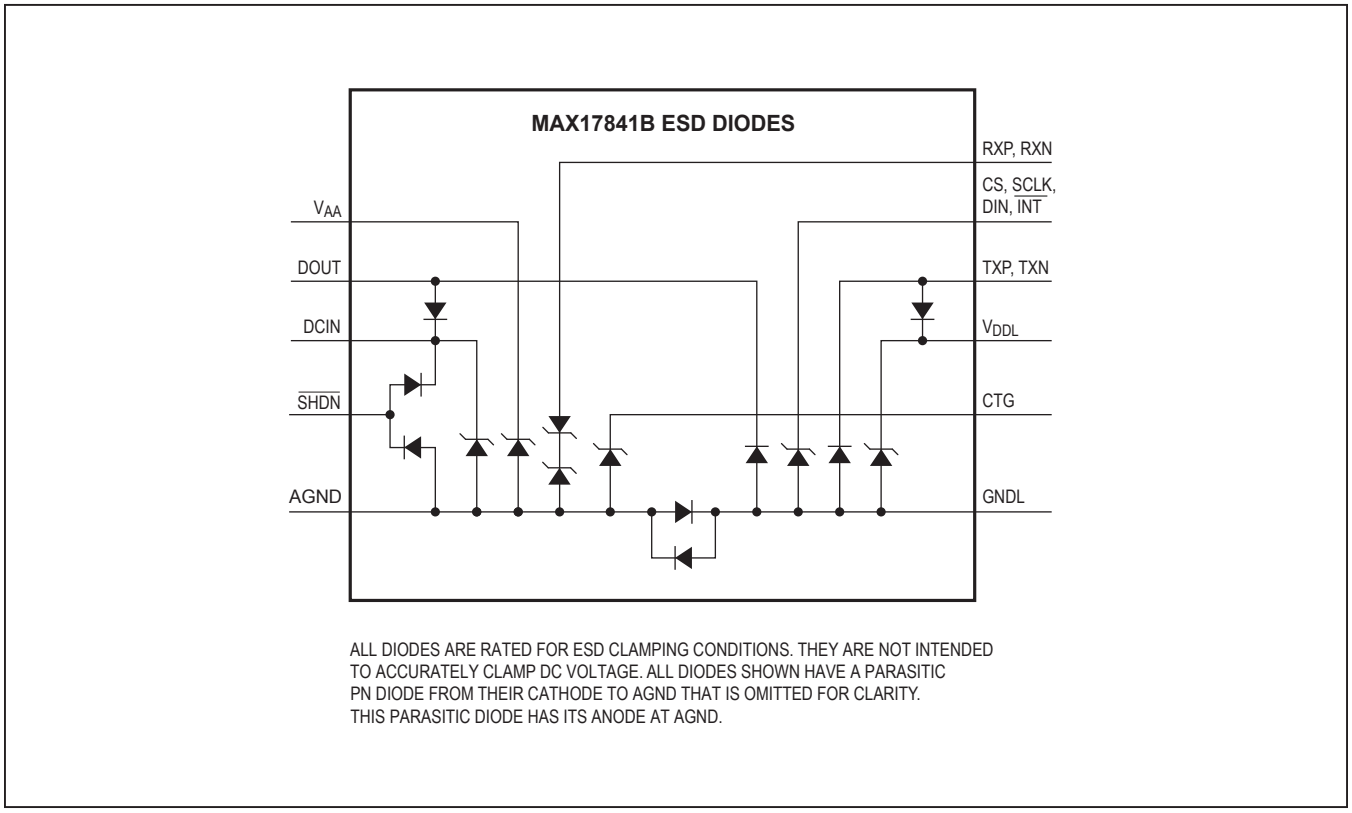


Figure 17. ESD Diode Diagram

Register Table

READ ADDRESS	WRITE ADDRESS	DEFAULT VALUE	NAME
0x01	NA	11h	RX_Status
0x03	NA	13h	TX_Status
0x05	0x04	00h	RX_Interrupt_Enable
0x07	0x06	00h	TX_Interrupt_Enable
0x09	0x08	00h	RX_Interrupt_Flags
0x0B	0x0A	80h	TX_Interrupt_Flags
0x0D	0x0C	60h	Configuration_1
0x0F	0x0E	10h	Configuration_2
0x11	0x10	0Fh	Configuration_3
0x13	NA	00h	FMEA
0x15	NA	84h	Model
0x17	NA	12h	Version
0x19	NA	01h	RX_Byte
0x1B	NA	3Eh	RX_Space
0x95	NA	00h	TX_Queue_Selects
0x97	NA	00h	RX_Read_Pointer
0x99	NA	01h	RX_Write_Pointer
0x9B	NA	00h	RX_Next_Message

RX_STATUS REGISTER

ADDRESS	BITS	DEFAULT	NAME	DESCRIPTION
0x01 (Read) (Note 1)	7	0	RX_Error_Status	The data byte at location RX_RD_Pointer may contain an error (the corresponding character contained a Manchester and/or parity error). This bit is set when the byte is read, not when the byte is received or written.
	6	0	Reserved	Always reads logic zero.
	5	0	RX_Busy_Status	The UART is busy receiving data.
	4	1	RX_Idle_Status	The UART is not receiving data.
	3	0	RX_Overflow_Status	The data byte at location RX_WR_POINTER in the receive buffer was overwritten because the receive buffer was full. Cleared when the receive buffer is not full (when the buffer is read).
	2	0	RX_Full_Status	The number of empty bytes in the receive buffer is less than the length of the message in the transmit queue.
	1	0	RX_STOP_Status	The UART has finished receiving a properly framed message (stop character) and it is ready to be read. The UART clears this bit after all unread messages have been read or if the UART detects a new preamble character. The UART does not set this bit if the buffer is empty and it receives a stop character.
	0	1	RX_Empty_Status	The receive buffer is cleared and contains no unread data (RX_RD_Pointer = RX_WR_Pointer - 1).

Register Table (continued)

TX_STATUS REGISTER				
ADDRESS	**BITS**	**DEFAULT**	**NAME**	**DESCRIPTION**
0x03 (Read) (Note 1)	7	0	Reserved	Always reads logic zero.
	6	0	Reserved	Always reads logic zero.
	5	0	TX_Busy_Status	The UART is busy transmitting data.
	4	1	TX_Idle_Status	The UART is not transmitting data.
	3	0	TX_Overflow_Status	LD_Q could not be incremented because the next queue contained untransmitted data. Any writes in this state overwrite the load queue.
	2	0	TX_Full_Status	All queues in the transmit buffer are full except the load queue (LD_Q = TX_Q - 1).
	1	1	TX_Available_Status	One or more queues in the transmit buffer are available for loading (TX_Full_Status is false).
	0	1	TX_Empty_Status	All the queues in the Transmit Buffer are cleared and available for loading (LD_Q = TX_Q).
RX_INTERRUPT_ENABLE REGISTER				
0x04 (Write) 0x05 (Read)	7	0	RX_Error_INT_Enable	Interrupt enable for RX_Error_Status
	6	0	Reserved	Always reads logic zero.
	5	0	RX_Busy_INT_Enable	Interrupt enable for RX_Busy_Status
	4	0	RX_Idle_INT_Enable	Interrupt enable for RX_Idle_Status
	3	0	RX_Overflow_INT_Enable	Interrupt enable for RX_Overflow_Status
	2	0	RX_Full_INT_Enable	Interrupt enable for RX_Full_Status
	1	0	RX_Stop_INT_Enable	Interrupt enable for RX_Stop_Status
	0	0	RX_Empty_INT_Enable	Interrupt enable for RX_Empty_Status
TX_INTERRUPT_ENABLE REGISTER				
0x06 (Write) 0x07 (Read)	7:6	00	Reserved	Always reads logic zero.
	5	0	TX_Busy_INT_Enable	Interrupt enable for TX_Busy_Status
	4	0	TX_Idle_INT_Enable	Interrupt enable for TX_Idle_Status
	3	0	TX_Overflow_INT_Enable	Interrupt enable for TX_Overflow_Status
	2	0	TX_Full_INT_Enable	Interrupt enable for TX_Full_Status
	1	0	TX_Available_INT_Enable	Interrupt enable for TX_Not_Full_Status
	0	0	TX_Empty_INT_Enable	Interrupt enable for TX_Empty_Status

Register Table (continued)

RX_INTERRUPT_FLAG REGISTER				
ADDRESS	**BITS**	**DEFAULT**	**NAME**	**DESCRIPTION**
0x08 (Write) 0x09 (Read) (Notes 2, 3)	7	0	RX_Error_INT_Flag	Interrupt flag for RX_Error_Status
	6	0	Reserved	Always reads logic zero.
	5	0	RX_Bus_ INT_Flag	Interrupt flag for RX_Busy_Status
	4	0	RX_Idle_INT_Flag	Interrupt flag for RX_Idle_Status
	3	0	RX_Overflow_INT_Flag	Interrupt flag for RX_Overflow_Status
	2	0	RX_Full_INT_Flag	Interrupt flag for RX_Full_Status
	1	0	RX_Stop_INT_ Flag	Interrupt flag for RX_Stop_Status
	0	0	RX_Empty_INT_Flag	Interrupt flag for RX_Empty_Status
TX_INTERRUPT_FLAG REGISTER				
0x0A (Write) 0x0B (Read) (Notes 2, 3)	7	1	POR_Flag	Set by power-on-reset event and cleared only by writing to logic zero. Has no effect on state of the $\overline{INT}$ pin.
	6	0	Reserved	Always reads logic zero.
	5	0	TX_Busy_INT_Flag	Interrupt flag for TX_Busy_Status
	4	0	TX_Idle_INT_Flag	Interrupt flag for TX_Idle_Status
	3	0	TX_Overflow_INT_Flag	Interrupt flag for TX_Overflow_Status
	2	0	TX_Full_INT_Flag	Interrupt flag for TX_Full_Status
	1	0	TX_Available_INT_Flag	Interrupt flag for TX_Available_Status
	0	0	TX_Empty_INT_Flag	Interrupt flag for TX_Empty_Status
CONFIGURATION_1 REGISTER				
0x0C (Write) 0x0D (Read)	7	0	Single_Ended_Mode	Enables the UART to receive a single-ended signal by shifting the input threshold negative (zero differential voltage is a logic one). In this mode, the RXP input should be connected to ground and the RXN input should receive the inverted signal, same as for differential mode. In this mode, the Tx port operates the same as in differential mode. Default is differential mode.
	6:5	11	Baud_Rate [1:0]	Configures the UART baud rate as follows: 00 = 500kbps 01 = 500kbps 10 = 1Mbps 11 = 2Mbps (default)
	4:0	0	Device_Count [4:0]	Not used by the ASCI. Can be used by the host to store the device count or for general-purpose use.

Register Table (continued)

CONFIGURATION_2 REGISTER				
ADDRESS	**BITS**	**DEFAULT**	**NAME**	**DESCRIPTION**
0x0E (Write) 0x0F (Read)	7	0	RX_Raw_Data	Receive Raw Data Mode: Disables Manchester decoding of the received data. In this mode, there is one data byte stored for every one character received (instead of every two received).
	6	0	TX_Raw_Data	Transmit Raw Data Mode: Disables Manchester encoding of transmitted data. In this mode, each data byte is transmitted as one character (instead of two characters).
	5	0	TX_Preambles	Transmit Preambles Mode: Transmits preambles continuously. This mode takes precedence over all transmit modes except Transmit Pause mode.
	4	1	TX_Queue	Transmit Queue Mode: Enables transmission of the message loaded in the Transmit Queue IF 1) there is sufficient space in the Receive Buffer for the message (RX_Full_Status is false) OR 2) the limitations on message length are removed (TX_Unlimited is set).
	3	0	TX_Odd_Parity	Transmit Odd Parity Mode: Transmits characters with odd parity. Since the UART protocol uses even parity, this mode can be used to test the system's ability to detect parity errors. Even parity is default.
	2	0	TX_Pause	Transmit Pause Mode: Places the transmitter into idle state once the UART has finished transmitting the current byte, however, the TX_Busy_Status and TX_Idle_Status bits remain unchanged. Transmission resumes when this bit is cleared. Note: This mode takes precedence over all other transmit modes (Transmit Preambles, Transmit Queue, and Keep-Alive modes).
	1	0	TX_No_Stop	Transmit No Stop Mode: Transmits messages without a stop character. By sending subsequent messages with the TX_No_Preamble bit set, a framed message of indefinite length can be constructed. The TX_Unlimited bit must be set for messages greater than 62 bytes.
	0	0	TX_No_Preamble	Transmit No Preamble Mode: Transmits messages without a preamble. By first sending a message in which the TX_No_Stop bit is set and then sending messages with this bit set, a framed message of indefinite length can be constructed. However, if the preceding message is terminated with a stop character (end of frame), then the data sent in this mode is unframed (no preamble) and is not stored in the receive buffer.

Register Table (continued)

CONFIGURATION_3 REGISTER				
ADDRESS	**BITS**	**DEFAULT**	**NAME**	**DESCRIPTION**
0x10 (Write) 0x11 (Read	7	0	Reserved	Always reads logic zero.
	6	0	Reserved	Always reads logic zero.
	5	0	TX_Unlimited	In this mode, the transmit queue automatically limits the message length to 255 bytes instead of the default 62-byte limit, and the message transmission is permitted even if the message length is greater than the available write space in the receive buffer.
	4	0	DOUT_Enable	SPI Output Enable: Asserts DOUT pin. Default is three-stated.
	3:0	1111	Keep_Alive [3:0]	Keep-Alive Mode: Periodically sends a stop character to prevent slave devices from shutting down during periods of no communication (idle state). The idle time in between the periodic stop characters is based on the 4-bit value below. The default setting is infinite (mode disabled). Note: the Transmit Pause, Transmit Preambles, and the Transmit Queue modes take precedence over this mode. 0000 = 0µs 0001 = 10µs 0010 = 20µs 0011 = 40µs 0100 = 80µs 0101 = 160µs 0110 = 320µs 0111 = 640µs 1000 = 1.28ms 1001 = 2.56ms 1010 = 5.12ms 1011 = 10.24ms 1111 = Infinite delay/disabled (default)
FMEA REGISTER				
0x13 (Read)	7:3	00000	Reserved	Always reads logic 0.
	2	0	AGND_Alert	Indicates V_{AGND} - V_{GNDL} > 0.2V
	1	0	VDDL_Alert	Indicates V_{AA} - V_{DDL} > 0.3V
	0	0	GNDL_Alert	Indicates V_{GNDL} - V_{AGND} > 0.2V
MODEL REGISTER				
0x15 (Read)	7:0	10000100	Model [11:4]	First two digits of the Model Number (84h)
VERSION REGISTER				
0x17 (Read)	7:4	0001	Model [3:0]	Last digit of the Model Number (1h)
	3:0	0010	Version [3:0]	Mask revision (2)

Register Table (continued)

RX_BYTE REGISTER				
ADDRESS	**BITS**	**DEFAULT**	**NAME**	**DESCRIPTION**
0x19 (Read)	7:3	00000	Reserved	Always reads logic zero.
	2	0	First_Byte	The byte at location RX_RD_Pointer is the first data byte in a message (the corresponding character was preceded by preamble character).
	1	0	Byte_Error	The byte at location RX_RD_Pointer may contain an error (the corresponding character contained a Manchester and/or parity error). This bit drives the RX_Error interrupt.
	0	0	Last_Byte	The byte at location RX_RD_Pointer is the last byte in a message (the corresponding character was a stop character and was stored as a null byte).
RX_SPACE REGISTER				
0x1B (Read)	7:0	00111110	RX_Space [7:0]	Number of available bytes in the receive buffer. Default is 62 bytes (3Eh).
TX_QUEUE_SELECTS REGISTER				
0x95 (Read)	7:4	0000	Reserved	Always reads logic zero.
	3:2	00	TX_Q [1:0]	Transmit Queue Select: Addresses one of four queues in the transmit buffer that the UART has selected for message transmission (sending).
	1:0	00	LD_Q [1:0]	Load Queue Select: Addresses one of four queues in the transmit buffer that the host has selected for message loading (writing).
RX_READ_POINTER REGISTER				
0x97 (Read)	7:0	00h	RX_RD_Pointer [7:0]	Receive Buffer Read Pointer: The location in the receive buffer that the host is to read. The UART automatically increments this pointer.
RX_WRITE_POINTER REGISTER				
0x99 (Read)	7:0	01h	RX_WR_Pointer [7:0]	Receive Buffer Write Pointer: The location in the receive buffer that is written by the UART as it receives data.
RX_NEXT_MESSAGE REGISTER				
0x9B (Read)	7:0	00h	RX_NXT_MSG_ Pointer [7:0]	Receive Buffer Next Message Pointer: The start of the next unread message in the receive buffer. The RX_RD_Pointer is loaded with this value by the RD_NXT_MSG SPI transaction.

Note 1: A status bit is set when its corresponding condition is true and is cleared when the condition is false.

Note 2: An interrupt flag (except the POR_Flag) is set only when its interrupt enable bit is true and its corresponding status bit goes from a logic zero state to logic one state. The flag can only be cleared by writing it to a logic zero. If the status bit is true when the flag enable is set or when the flag is cleared, the flag remains cleared until the status bit transitions from a logic zero state to a logic one state.

Note 3: If any interrupt flag (except the POR_Flag) is set, then the $\overline{INT}$ pin is asserted (active low).

Ordering Information

PART	TEMP RANGE	PIN-PACKAGE
MAX17841BGUE+	-40°C to +105°C	16 TSSOP
MAX17841BGUE/V+	-40°C to +105°C	16 TSSOP

+Denotes a lead(Pb)-free/RoHS-compliant package.
/V denotes an automotive qualified part.

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.
16 TSSOP	U16+1	**21-0066**	**90-0117**

Revision History

REVISION NUMBER	REVISION DATE	DESCRIPTION	PAGES CHANGED
0	9/13	Initial release	—
1	2/14	Added MAX17842B to data sheet	1–33
2	1/15	Deleted MAX17842B from data sheet	1–33

