MAX77714 I^{2}C-Compatible Serial Interface Implementation Guide

UG6564; Rev 0; 1/18

Abstract

This document serves as a guide to engineers designing with the MAX77714 serial interface. It details the various communication protocols implemented in the 2-wire serial interface block of the MAX77714, as well as some general information on the I^{2}C specification.

Table of Contents

Overview 4

Features 4

Write Protection 4

Writing to Registers Using the I2CWP Bit 4

Simplified Block Diagram 5

System Configuration 5

Hardware Implementation and Interface Power 6

Data Transfer 6

START and STOP Conditions 6

Acknowledge Bit 7

Slave Address 7

Clock Stretching 7

General Call Address 7

Device ID 7

Communication Speed 7

Communication Protocols 9

Writing to a Single Register 9

Writing Multiple Bytes to Sequential Registers 10

Reading from a Single Register 12

Reading from Sequential Registers 13

Engaging High-Speed Mode (HS-Mode) for Operation Up to 3.4MHz 14

Revision History 15

List of Figures

Figure 1. Simplified block diagram. 5

Figure 2. I^{2}C system configuration. 5

Figure 3. I^{2}C START and STOP conditions. 6

Figure 4. Writing to a single register. 9

Figure 5. Writing multiple bytes.. 11

Figure 6. Reading from a single register.. 12

Figure 7. Reading from multiple registers.. 13

Figure 8. Engaging high-speed mode.. 14

List of Tables

Table 1. MAX77714 Slave Addresses 7

Overview

The MAX77714 features a revision 4.0 I^{2}C-compatible, 2-wire serial interface consisting of a bidirectional serial data line (SDA) and a serial clock line (SCL). As shown in the Simplified Block Diagram, the I^{2}C SDA and SCL signals are internally decoded by the devices used to communicate with the top-level control logic as well as the bucks, LDOs, and GPIO. The MAX77714 is a slave-only device that relies on an external bus master to generate SCL. SCL clock rates from 0Hz to 3.4MHz are supported. I^{2}C is an open-drain bus and, therefore, SDA and SCL require pullups.

The MAX77714 I^{2}C communication controller implements 7-bit slave addressing. An I^{2}C bus master initiates communication with the slave by issuing a START condition followed by the slave address. The OTP address is factory programmable for one of two options. All slave addresses not mentioned in the slave address table (Table 1) are not acknowledged.

The devices use 8-bit registers with 8-bit register addressing. They support standard communication protocols: (1) writing to a single register, (2) writing to multiple sequential registers with an automatically incrementing data pointer, (3) reading from a single register, and (4) reading from multiple sequential registers with an automatically incrementing data pointer.

For additional information on the I^{2}C protocols, refer to the I^{2}C bus specification available from NXP (Philips) Semiconductors.

Features

- I^{2}C Revision 4.0-Compatible Serial Communications Channel
- 0Hz to 100kHz (Standard Mode)
- 0Hz to 400kHz (Fast Mode)
- 0Hz to 1MHz (Fast Mode Plus)
- 0Hz to 3.4MHz (High-Speed Mode)
- Does not utilize I^{2}C clock stretching

Write Protection

The MAX77714 incorporates write protection for all the registers in the PMIC.

- When the I2CWP bit is set to 1, writes to any register are ignored.
- However, the I^{2}C configuration registers I2C_CTRL1 and I2C_CTRL2 are not subject to write protection even if I2CWP = 1.
- The default value for this bit is 0. If the customer decides to use this feature, the microcontroller must be configured to set I2CWP = 1. Once this bit is set, I^{2}C can be used to update the registers by first clearing this bit (I2CWP = 0), followed by updating the I^{2}C register of interest, followed by setting the I2CWP = 1.

Writing to Registers Using the I2CWP Bit

To write to a register, the first write must be to the register containing the I2CWP bit, followed by the write to the appropriate data registers with a repeated START condition in between.

Simplified Block Diagram

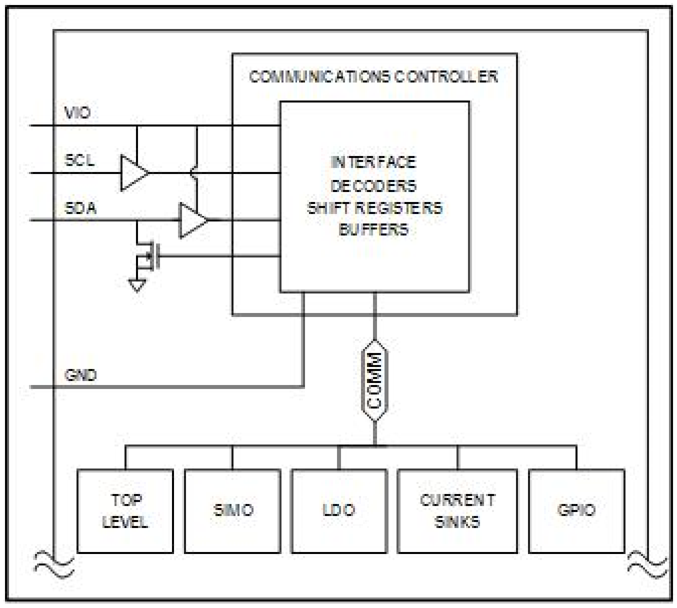

Figure 1. Simplified block diagram.

System Configuration

The I^{2}C bus is a multimaster bus. The maximum number of devices that can attach to the bus is only limited by bus capacitance.

A device on the I^{2}C bus that sends data to the bus is called a transmitter. A device that receives data from the bus is called a receiver. The device that initiates a data transfer and generates the SCL clock signals to control the data transfer is a master. Any device that is being addressed by the master is considered a slave. The MAX77714 I^{2}C-compatible interface operates as a slave on the I^{2}C bus with transmit and receive capabilities.

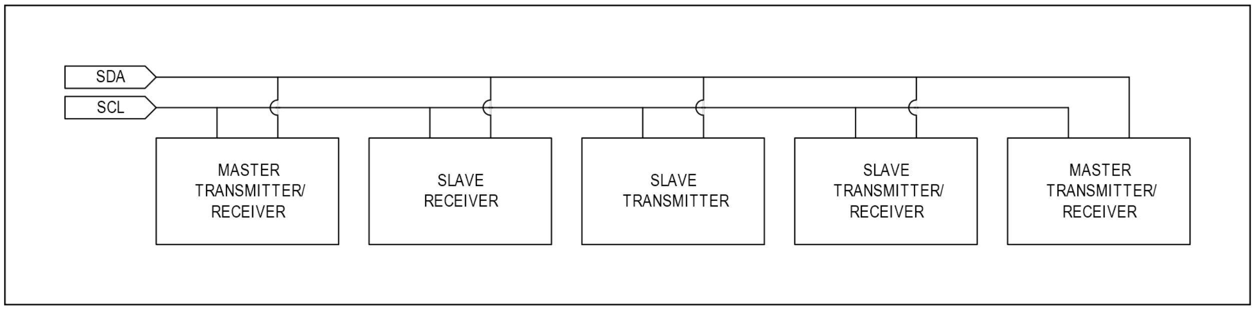
SDA
SCL
MASTER TRANSMITTER/ RECEIVER
SLAVE RECEIVER
SLAVE TRANSMITTER
SLAVE TRANSMITTER/ RECEIVER
MASTER TRANSMITTER/ RECEIVER

Figure 2. I^{2}C system configuration.

Hardware Implementation and Interface Power

The MAX77714 I^2C interface derives its power from V_{IO}. Typically, a power input such as V_{IO} would require a local 0.1µF ceramic bypass capacitor to ground. However, in highly integrated power distribution systems, a dedicated capacitor might not be necessary. If the impedance between V_{IO} and the next closest capacitor (≥ 0.1µF) is less than 100mΩ in series with 10nH, then a local capacitor is not needed. Otherwise, bypass V_{IO} to GND with a 0.1µF ceramic capacitor.

V_{IO} accepts voltages from 1.7V to 3.6V. Cycling V_{IO} does not reset the I^2C registers. When V_{IO} is invalid and V_{SYS} is less than $V_{SYSUVLO}$, SDA and SCL are high impedance. Note that I^2C is an open-drain bus and requires pullup resistors. Typical applications place these pullups near the host controller.

Data Transfer

One data bit is transferred during 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 are control signals. See the *I^2C START and STOP Conditions* section. Each transmit sequence is framed by a START (S) condition and a STOP (P) condition. Each data packet is nine bits long: eight bits of data followed by the acknowledge bit. Data is transferred with the MSB first.

START and STOP Conditions

When the serial interface is inactive, SDA and SCL idle high. A master device initiates communication by issuing a START condition. A START condition is a high-to low transition on SDA with SCL high. A STOP condition is a low-to-high transition on SDA, while SCL is high.

A START condition from the master signals the beginning of a transmission to the MAX77714. The master terminates transmission by issuing a not-acknowledge followed by a STOP condition (see the *Acknowledge Bit* section for information on not-acknowledge). The STOP condition frees the bus. To issue a series of commands to the slave, the master can issue repeated START (Sr) commands instead of a STOP command to maintain control of the bus. In general, a repeated START command is functionally equivalent to a regular START command.

When a STOP condition or incorrect address is detected, the MAX77714 internally disconnects SCL from the serial interface until the next START condition, minimizing digital noise and feedthrough.

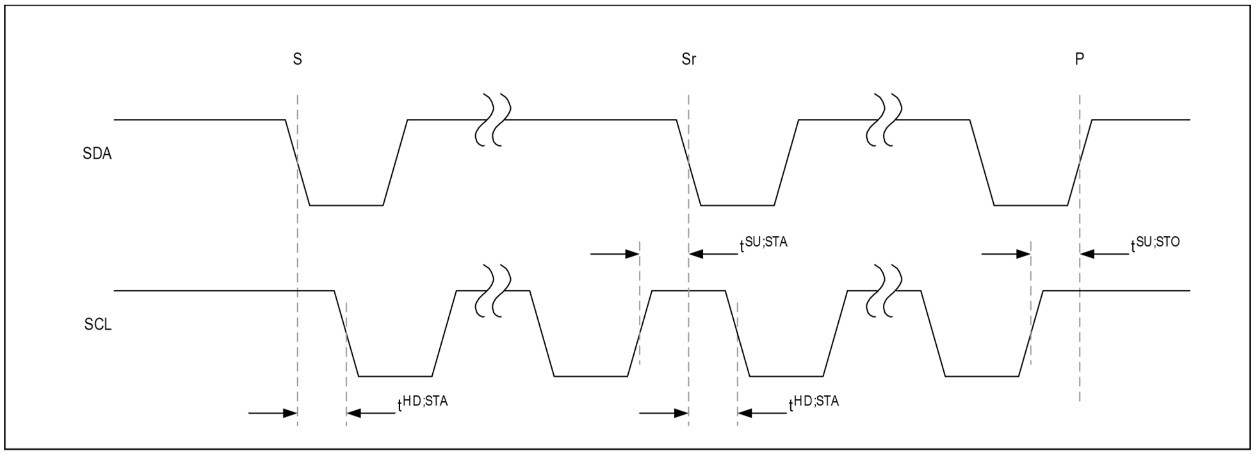

Figure 3. I^2C START and STOP conditions.

Acknowledge Bit

Both the I^{2}C bus master and the MAX77714 (slave) generate acknowledge bits when receiving data. The acknowledge bit is the last bit of each 9-bit data packet. To generate an acknowledge (A), the receiving device must pull SDA low before the rising edge of the acknowledge-related clock pulse (ninth pulse) and keep it low during the high period of the clock pulse. To generate a not-acknowledge (NA), the receiving device allows SDA to be pulled high before the rising edge of the acknowledge-related clock pulse and leaves it high during the high period of the clock pulse.

Monitoring the acknowledge bits allows for detection of unsuccessful data transfers. An unsuccessful data transfer occurs if a receiving device is busy or if a system fault has occurred. In the event of an unsuccessful data transfer, the bus master should reattempt communication at a later time.

The MAX77714 issues an ACK for all register addresses in the possible address space even if the particular register does not exist.

Slave Address

Refer to the *I^{2}C Serial Interface* and the *Ordering Information* section of the MAX77714 IC data sheet for more information.

Table 1. MAX77714 Slave Addresses

OTP_I2CADDR[1:0]	RTC SLAVE ADDRESS WRITE	RTC SLAVE ADDRESS READ	PMIC/GPIO SLAVE ADDRESS WRITE	PMIC/GPIO SLAVE ADDRESS READ
0b00	0x90, 0b1001_0000	0x91, 0b1001_0001	0x38, 0b0011_1000	0x39, 0b0011_1001
0b01	0x94, 0b1001_0100	0x95, 0b1001_0101	0x3C, 0b0011_1100	0x3D, 0b0011_1101
0b10	0xD0, 0b1101_0000	0xD1, 0b1101_0001	0x78, 0b0111_1000	0x79, 0b0111_1001
0b11	0xD4, 0b1101_0100	0xD5, 0b1101_0101	0x7C, 0b0111_1100	0x7D, 0b0111_1101

Clock Stretching

In general, the clock signal generation for the I^{2}C bus is the responsibility of the master device. The I^{2}C specification allows slow slave devices to alter the clock signal by holding down the clock line. The process in which a slave device holds down the clock line is typically called clock stretching. The MAX77714 does not use any form of clock stretching to hold down the clock line.

General Call Address

The MAX77714 does not implement the I^{2}C specification's general call address. If the MAX77714 sees the general call address (0b0000_0000), it does not issue an acknowledge.

Device ID

The MAX77714 does not support the I^{2}C Device ID feature.

Communication Speed

The MAX77714 is compatible with all four communication speed ranges as defined by the Revision 4.0 I^{2}C specification:

- 0Hz to 100kHz (Standard Mode)
- 0Hz to 400kHz (Fast Mode)
- 0Hz to 1MHz (Fast Mode Plus)
- 0Hz to 3.4MHz (High-Speed Mode)

Operating in standard mode, fast mode, and fast mode plus does not require any special protocols. The main consideration when changing the bus speed through this range is the combination of the bus capacitance and pullup resistors. Higher time constants created by the bus capacitance and pullup resistance (C x R) slow the bus operation. Therefore, when increasing bus speeds, the pullup resistance must be decreased to maintain a reasonable time constant. Refer to the *Pullup Resistor Sizing* section of the I^2C bus specification and user manual (available for free online) for detailed guidance on the pullup resistor selection. In general, for bus capacitances of 200pF, a 100kHz bus needs 5.6kΩ pullup resistors, a 400kHz bus needs approximately 1.5kΩ pullup resistors, and a 1MHz bus needs 680Ω pullup resistors. Note that when the open-drain bus is low, the pullup resistor is dissipating power, and lower value pullup resistors dissipate more power (V^2/R).

Operating in high-speed mode requires some special considerations. For a full list of considerations, refer to the I^2C bus specification and user manual. The major considerations with respect to the MAX77714 include the following:

- The I^2C bus master should use current source pullups to shorten the signal rise time.
- The I^2C slave must use a different set of input filters on its SDA and SCL lines to accommodate for the higher bus speed.
- The communication protocols need to utilize the high-speed master code.

At power-up and after each STOP condition, the MAX77714 input filters are set for standard mode, fast mode, and fast mode plus (i.e., 0Hz to 1MHz). To switch the input filters for high-speed mode, use the high-speed master code protocols that are described in the *Communication Protocols* section.

Communication Protocols

The MAX77714 supports both writing and reading from its registers.

Writing to a Single Register

Figure 4 shows the protocol for the I^2C master device to write one byte of data to the MAX77714. This protocol is the same as the SMBus specification's write-byte protocol.

The write-byte protocol is as follows:

1. The master sends a START command (S).
2. The master sends the 7-bit slave address followed by a write bit ($R/\overline{W}$ = 0).
3. The addressed slave asserts an acknowledge (A) by pulling SDA low.
4. The master sends an 8-bit register pointer.
5. The slave acknowledges the register pointer.
6. The master sends a data byte.
7. The slave updates with the new data
8. The slave acknowledges or not-acknowledges the data byte. The next rising edge on SDA loads the data byte into its target register and the data becomes active.
9. The master sends a STOP condition (P) or a repeated START condition (Sr). Issuing a P ensures that the bus input filters are set for 1MHz or slower operation. Issuing an Sr leaves the bus input filters in their current state.

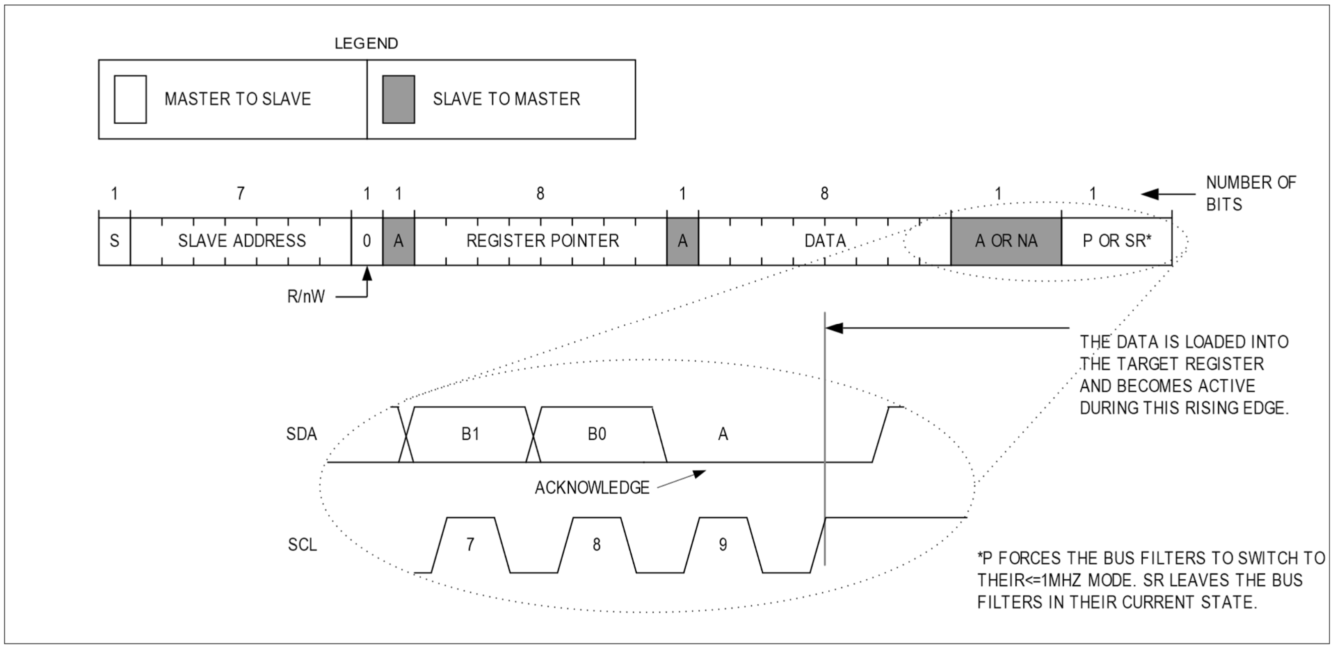
LEGEND
MASTER TO SLAVE
SLAVE TO MASTER
1
7
1
1
8
1
8
1
1
NUMBER OF BITS
S
SLAVE ADDRESS
0
A
REGISTER POINTER
A
DATA
A OR NA
P OR SR*
R/nW
THE DATA IS LOADED INTO THE TARGET REGISTER AND BECOMES ACTIVE DURING THIS RISING EDGE.
SDA
B1
B0
A
ACKNOWLEDGE
SCL
7
8
9
*P FORCES THE BUS FILTERS TO SWITCH TO THEIR<=1MHZ MODE. SR LEAVES THE BUS FILTERS IN THEIR CURRENT STATE.

Figure 4. Writing to a single register.

Writing Multiple Bytes to Sequential Registers

Figure 5 shows the protocol for writing to sequential registers. This protocol is similar to the write-byte protocol above, except the master continues to write after it receives the first byte of data. When the master is done writing it issues a STOP or repeated START.

The writing to sequential registers protocol is as follows:

1. The master sends a START command (S).
2. The master sends the 7-bit slave address followed by a write bit ($R/\overline{W} = 0$).
3. The addressed slave asserts an acknowledge (A) by pulling SDA low.
4. The master sends an 8-bit register pointer.
5. The slave acknowledges the register pointer.
6. The master sends a data byte.
7. The slave acknowledges the data byte. The next rising edge on SDA load the data byte into its target register and the data becomes active.
8. Steps 6 to 7 are repeated as many times as the master requires.
9. During the last acknowledge-related clock pulse, the master can issue an acknowledge or a not-acknowledge.
10. The master sends a STOP condition (P) or a repeated START condition (Sr). Issuing a P ensures that the bus input filters are set for 1MHz or slower operation. Issuing an Sr leaves the bus input filters in their current state.

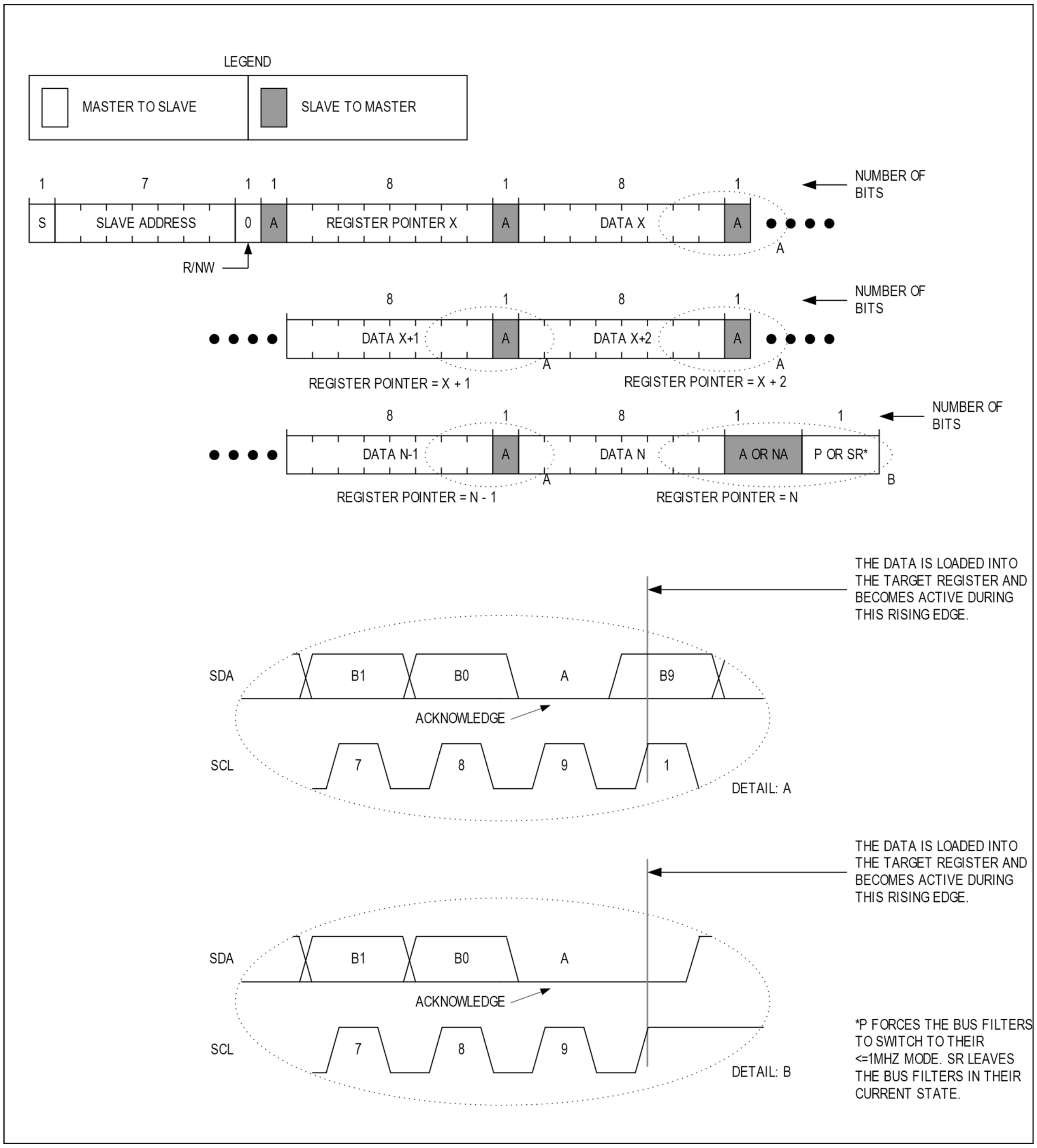

Figure 5. Writing multiple bytes.

Reading from a Single Register

Figure 6 shows the protocol for the I^2C master device to read one byte of data from the MAX77714. This protocol is the same as the SMBus specification's read-byte protocol.

The read byte protocol is as follows:

1. The master sends a START command (S).
2. The master sends the 7-bit slave address followed by a write bit ($R/\overline{W} = 0$).
3. The addressed slave asserts an acknowledge (A) by pulling SDA low.
4. The master sends an 8-bit register pointer.
5. The slave acknowledges the register pointer.
6. The master sends a repeated START command (Sr).
7. The master sends the 7-bit slave address followed by a read bit ($R/\overline{W} = 1$).
8. The addressed slave asserts an acknowledge by pulling SDA low.
9. The addressed slave places 8 bits of data on the bus from the location specified by the register pointer.
10. The master issues a not-acknowledge (nA).
11. The master sends a STOP condition (P) or a repeated START condition (Sr). Issuing a P ensures that the bus input filters are set for 1MHz or slower operation. Issuing an Sr leaves the bus input filters in their current state.

Note that when the MAX77714 receives a STOP, it does not modify the register pointer.

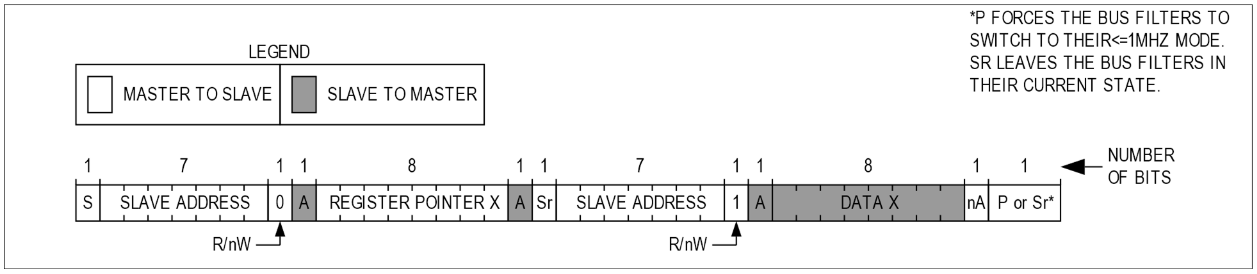

Figure 6. Reading from a single register.

Reading from Sequential Registers

Figure 7 shows the protocol for reading from sequential registers. This protocol is similar to the read-byte protocol except the master issues an acknowledge to signal the slave that it wants more data. When the master has all the data it requires, it issues a not-acknowledge (nA) and a STOP (P) to end the transmission.

The continuous read from sequential registers protocol is as follows:

1. The master sends a START command (S).
2. The master sends the 7-bit slave address followed by a write bit ($R/\overline{W} = 0$).
3. The addressed slave asserts an acknowledge (A) by pulling SDA low.
4. The master sends an 8-bit register pointer.
5. The slave acknowledges the register pointer.
6. The master sends a repeated START command (Sr).
7. The master sends the 7-bit slave address followed by a read bit ($R/\overline{W} = 1$). When reading the RTC timekeeping registers, secondary buffers are loaded with the timekeeping register data during this operation.
8. The addressed slave asserts an acknowledge by pulling SDA low.
9. The addressed slave places 8 bits of data on the bus from the location specified by the register pointer.
10. The master issues an acknowledge (A) signaling the slave that it wishes to receive more data.
11. Steps 9 to 10 are repeated as many times as the master requires. Following the last byte of data, the master must issue a not-acknowledge (nA) to signal that it wishes to stop receiving data.
12. The master sends a STOP condition (P) or a repeated START condition (Sr). Issuing a STOP (P) ensures that the bus input filters are set for 1MHz or slower operation. Issuing an Sr leaves the bus input filters in their current state.

Note that when the MAX77714 receives a STOP, it does not modify the register pointer.

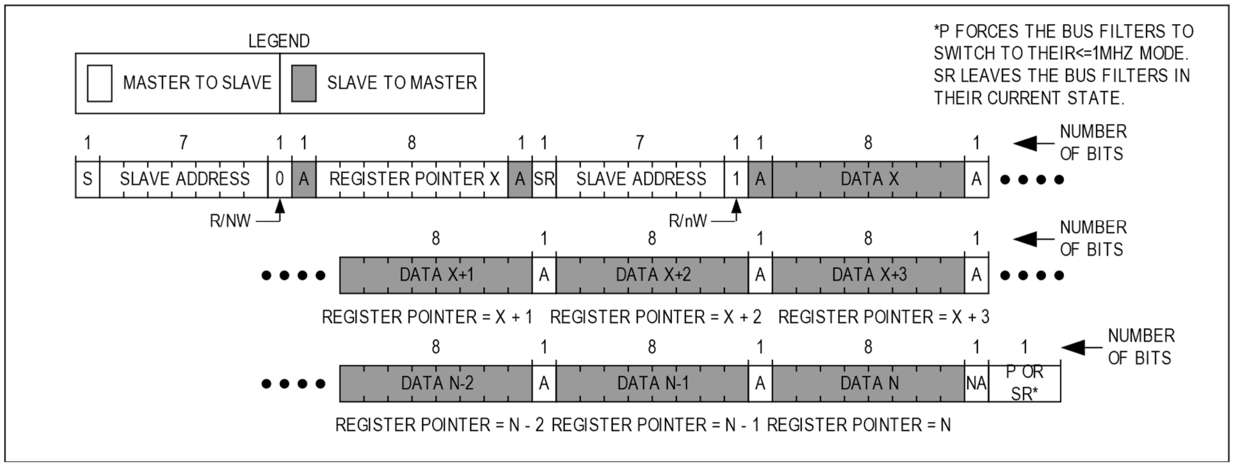

Figure 7. Reading from multiple registers.

Engaging High-Speed Mode (HS-Mode) for Operation Up to 3.4MHz

Figure 8 shows the protocol for engaging HS-mode operation. HS-mode operation allows for a bus operating speed up to 3.4MHz.

Engaging the HS-mode protocol is as follows:

1. Begin the protocol while operating at a bus speed of 1MHz or lower.
2. The master sends a START command (S).
3. The master sends the 8-bit master code of 0b0000 1XXX where 0bXXX are don't care bits.
4. The addressed slave issues a not-acknowledge (nA).
5. The master can now increase its bus speed up to 3.4MHz and issue any read/write operation.

The master can continue to issue high-speed read/write operations until a stop (P) is issued. To continue operations in HS-mode, use repeated START (Sr).

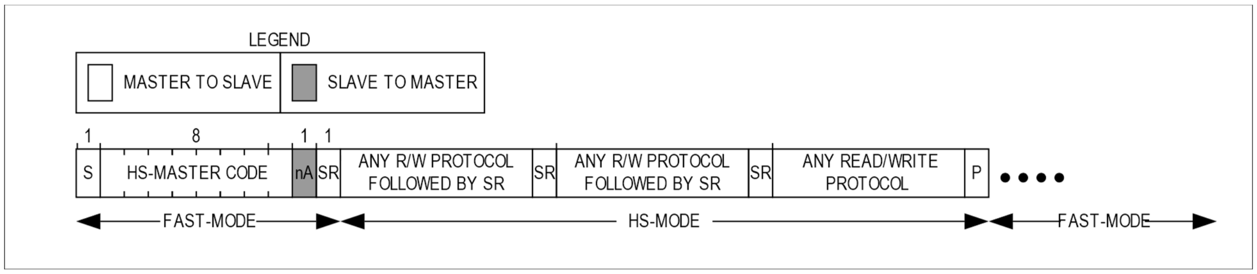

Figure 8. Engaging high-speed mode.

Revision History

REV NUMBER	REV DATE	DESCRIPTION	PAGES CHANGED
0	1/18	Initial release	—