AVR Microcontroller

Microprocessor Course

Chapter 16

I2C PROTOCOL and

DS1307 RTC INTERFACING

Day 1397

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.1 : I2C BUS PROTOCOL

The IIC (Inter-Integrated Circuit) is a bus interface connection incorporated into many devices such as sensors, RTC, and EEPROM. The IIC is also referred to as I2C (I^2C) or I square C in many technical literatures.

I2C bus

The I2C bus was originally started by Philips, but in recent years has become a widely used standard adapted by many semiconductor chip companies. I2C is ideal for attaching low-speed peripherals to a motherboard or embedded system or anywhere that a reliable communication over a short distance is required. I2C provides a connection-oriented communication with acknowledge. I2C devices use only 2 pins for data transfer, instead of the 8 or more pins used in traditional buses. They are called SCL (Serial Clock), which synchronize the data transfer between two chips, and SDA (Serial Data).

This reduction of communication pins reduces the package size and power consumption drastically, making them ideal for many applications in which space is a major concern. These two pins, SDA and SCI, make the I2C a 2-wire interface.

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING

SECTION 18.1 : I2C BUS PROTOCOL

These two pins, SDA and SCK, make the I2C a 2-wire interface. In many application notes, including AVR datasheets, I2C is referred to as Two-Wire Serial Interface (TWI). In this chapter we use I2C and TWI interchangeably.

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.1 : I2C BUS PROTOCOL

I2C line electrical characteristics

I2C devices use only 2 bidirectional open-drain pins for data communication. To implement I2C, only a 4.7 KΩ pull-up resistor for each of bus lines is needed.

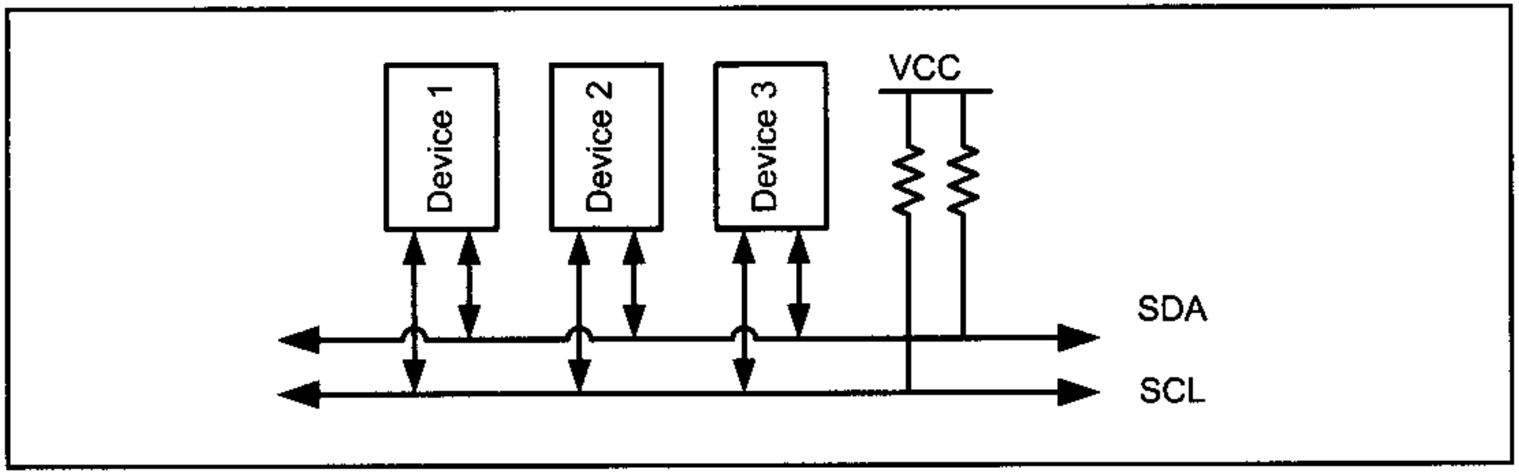


Figure 18-1. I2C Bus

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING

SECTION 18.1 : I2C BUS PROTOCOL

This implements a wired-AND, which is needed to implement I2C protocols. This means that if one or more devices pull the line to low (zero) level, the line state is zero and the level of line will be 1 only if none of devices pull the line to low level.

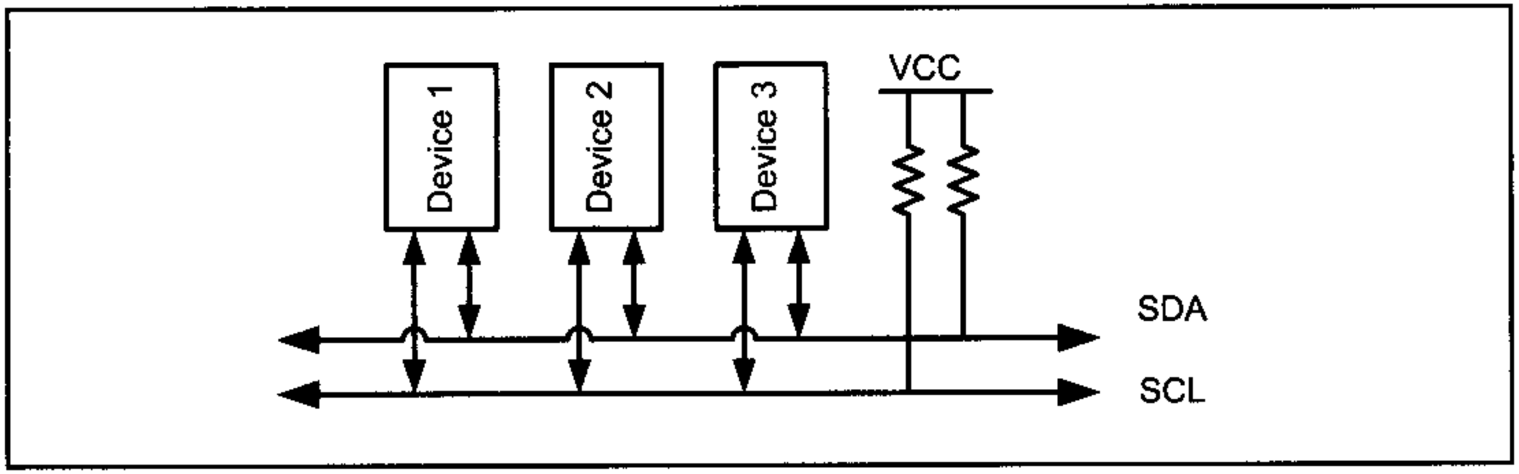


Figure 18-1. I2C Bus

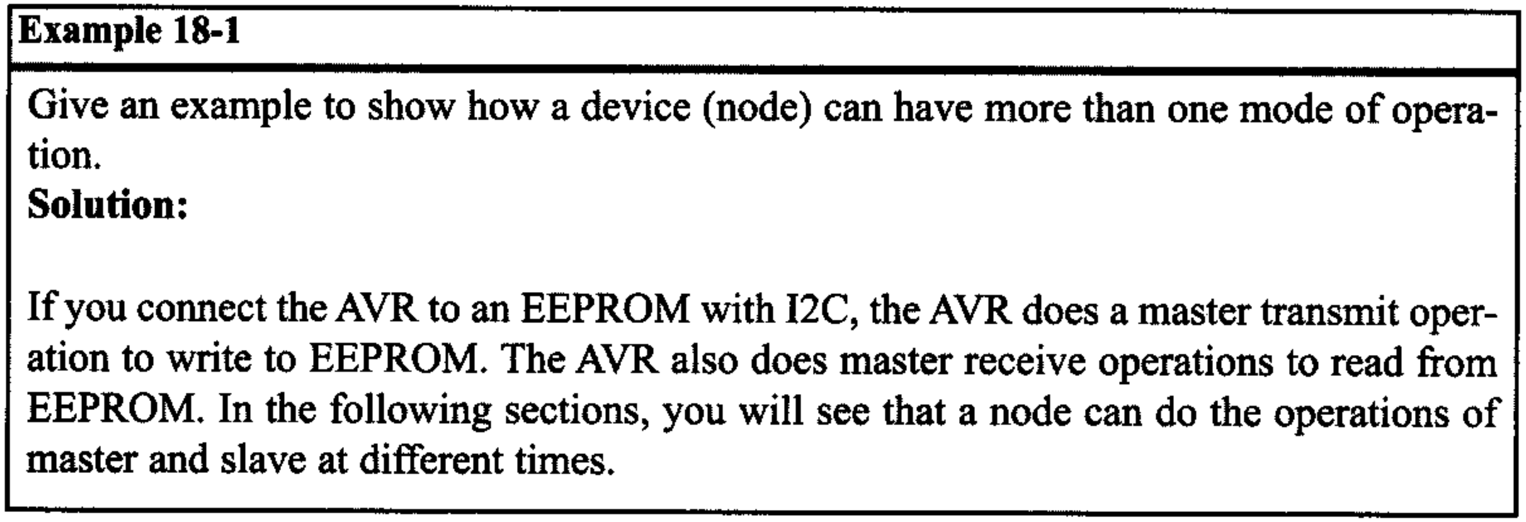
SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.1 : I2C BUS PROTOCOL

I2C nodes

In the AVR up to 120 different devices can share an I2C bus. Each of these devices is called a node. In I2C terminology, each node can operate as either master or slave. Master is a device that generates the clock for the system; it also initiates and terminates a transmission. Slave is the node that receives the clock and is addressed by the master.

In I2C, both master and slave can receive or transmit data, so there are four modes of operation. They are master transmitter, master receiver, slave transmitter, and slave receiver. Notice that each node can have more than one mode of operation at different times, but it has only one mode of operation at a given time.

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.1 : I2C BUS PROTOCOL



Example 18-1

Give an example to show how a device (node) can have more than one mode of operation.

Solution:

If you connect the AVR to an EEPROM with I2C, the AVR does a master transmit operation to write to EEPROM. The AVR also does master receive operations to read from EEPROM. In the following sections, you will see that a node can do the operations of master and slave at different times.

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.1 : I2C BUS PROTOCOL

Bit format

I2C is a synchronous serial protocol; each data bit transferred on the SDA line is synchronized by a high-to-low pulse of clock on the SCL line. According to I2C protocols the data line cannot change when the clock line is high; it can change only when the clock line is low. The STOP and START conditions are the only exceptions to this rule.

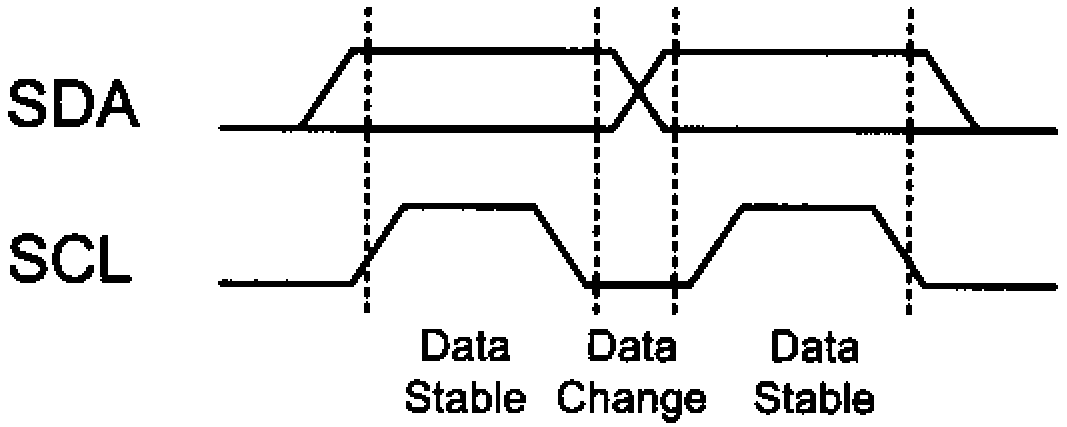


Figure 18-2. I2C Bit Format

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.1 : I2C BUS PROTOCOL

START and STOP conditions

I2C is a connection-oriented communication protocol. This means that each transmission is initiated by a START condition and is terminated by a STOP condition. Remember that the START and STOP conditions are generated by the master. STOP and START conditions must be distinguished from bits of address or data. That is why they do not obey the bit format rule that we mentioned before.

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.1 : I2C BUS PROTOCOL

START and STOP conditions are generated by keeping the level of the SCL line high and then changing the level of the SDA line.

The START condition is generated by a high-to-low change in the SDA line when SCL is high.

The STOP condition is generated by a low-to-high change in the SDA line when SCL is low. See Figure 18-3.

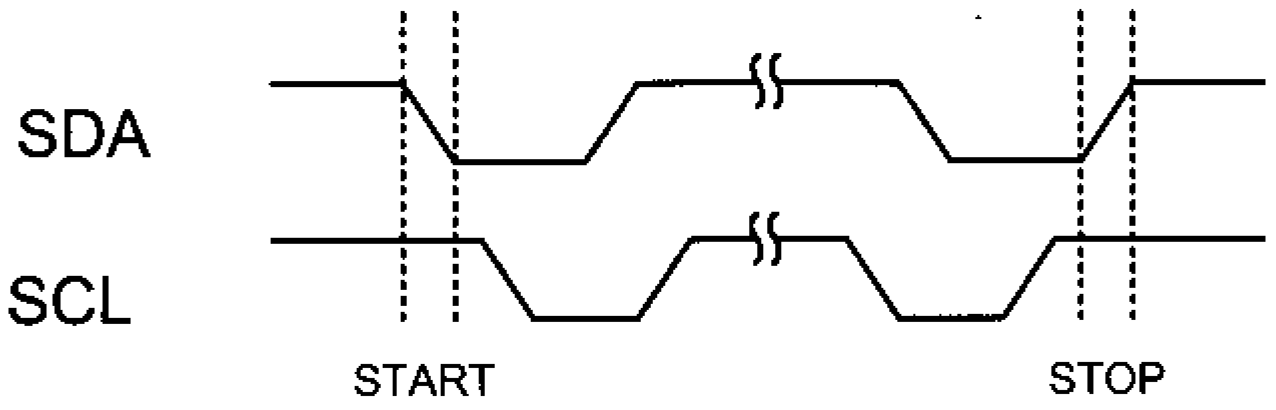


Figure 18-3. START and STOP Conditions

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.1 : I2C BUS PROTOCOL

The bus is considered busy between each pair of START and STOP conditions, and no other master tries to take control of the bus when it is busy. If a master, which has the control of the bus, wishes to initiate a new transfer and does not want to release the bus before starting the new transfer, it issues a new START condition between a pair of START and STOP conditions. It is called the REPEATED START condition.

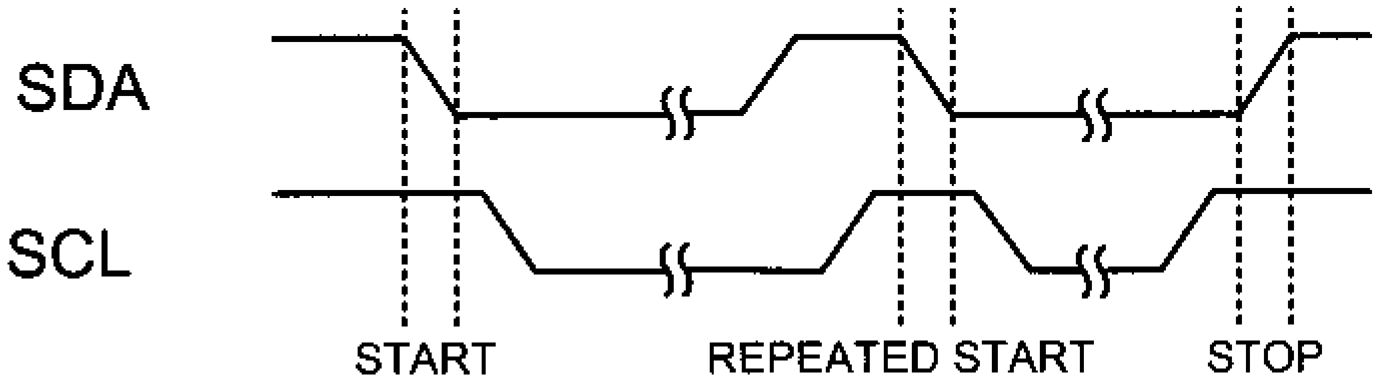
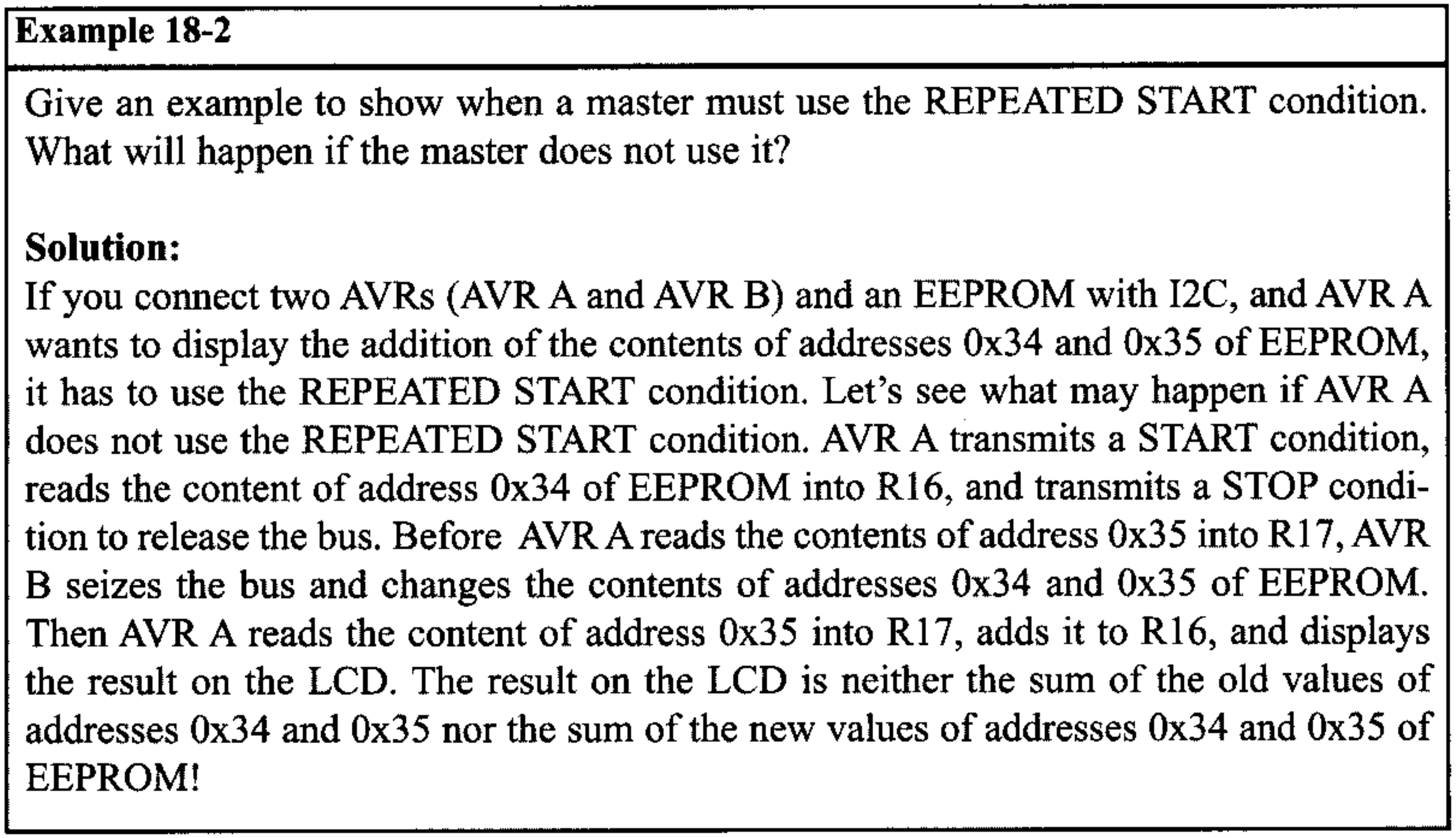


Figure 18-4. REPEATED START Condition

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.1 : I2C BUS PROTOCOL



Example 18-2

Give an example to show when a master must use the REPEATED START condition. What will happen if the master does not use it?

Solution:

If you connect two AVRs (AVR A and AVR B) and an EEPROM with I2C, and AVR A wants to display the addition of the contents of addresses 0x34 and 0x35 of EEPROM, it has to use the REPEATED START condition. Let's see what may happen if AVR A does not use the REPEATED START condition. AVR A transmits a START condition, reads the content of address 0x34 of EEPROM into R16, and transmits a STOP condition to release the bus. Before AVR A reads the contents of address 0x35 into R17, AVR B seizes the bus and changes the contents of addresses 0x34 and 0x35 of EEPROM. Then AVR A reads the content of address 0x35 into R17, adds it to R16, and displays the result on the LCD. The result on the LCD is neither the sum of the old values of addresses 0x34 and 0x35 nor the sum of the new values of addresses 0x34 and 0x35 of EEPROM!

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.1 : I2C BUS PROTOCOL

Packet format in 12C

In I2C, each address or data to be transmitted must be framed in a packet. Each packet is 9 bits long. The first 8 bits are put on the SDA line by the transmitter, and the 9th bit is an acknowledge by the receiver or it may be NACK (not acknowledge). The clock is generated by the master, regardless of whether it is the transmitter or receiver. To get an acknowledge, the transmitter releases the SDA line during the ninth clock so that the receiver can pull the SDA line low to an ACK. If the receiver doesn't pull the SDA line low, it is considered as NACK.

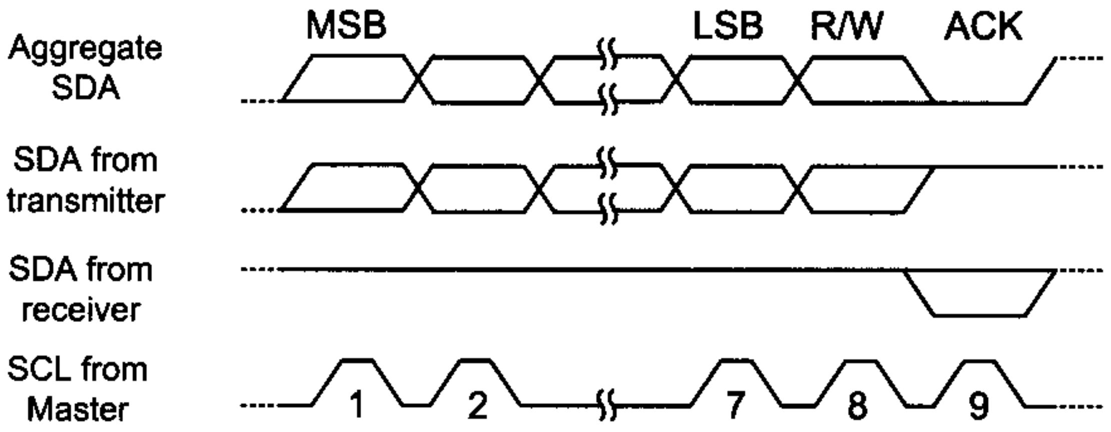


Figure 18-5. Packet Format in I2C

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.1 : I2C BUS PROTOCOL

In I2C, each packet may contain either address or data. Also notice that

START condition + address packet + one or more data packet + STOP condition

together form a complete data transfer.

Address packet format Like any other packets, all address packets transmitted on the I2C bus are nine bits long. An address packet consists of seven address bits, one READ/WRITE control bit, and an acknowledge bit.

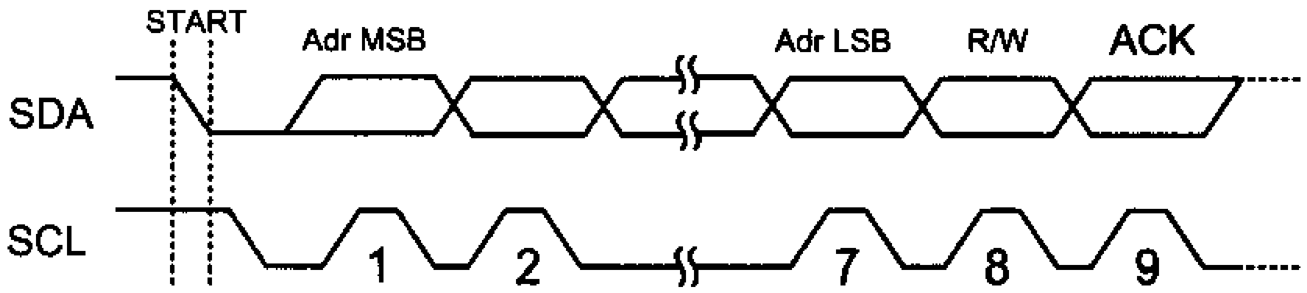


Figure 18-6. Address Packet Format in I2C

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.1 : I2C BUS PROTOCOL

Address packet format Like any other packets, all address packets transmitted on the I2C bus are nine bits long. An address packet consists of seven address bits, one READ/WRITE control bit, and an acknowledge bit (see Figure 18-6).

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.1 : I2C BUS PROTOCOL

Address bits are used to address a specific slave device on the bus. The 7-bit address lets the master address a maximum of 128 slaves on the bus, although the address 0000 000 is reserved for general call and all addresses of the format 1111 xxx are reserved. That means 119 (128 -1- 8) devices can share an I2C bus.

In the I2C bus the MSB of the address is transmitted first. The eighth bit in the packet is the READ/WRITE control bit. If this bit is set, the master will read the next frame (Data) from the slave, otherwise, the master will write the next frame (Data) on the bus to the slave.

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.1 : I2C BUS PROTOCOL

When a slave detects its address on the bus, it knows that it is being addressed and it should acknowledge in the ninth SCL (ACK) cycle by changing SDA to zero. If the addressed slave is not ready or for any reason does not want to service the master, it should leave the SDA line high in the ninth clock cycle. This is considered to be NACK. In case of NACK, the master can transmit a STOP condition to terminate the transmission, or a REPEATED START condition to initiate a new transmission.

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.1 : I2C BUS PROTOCOL

Example 18-3

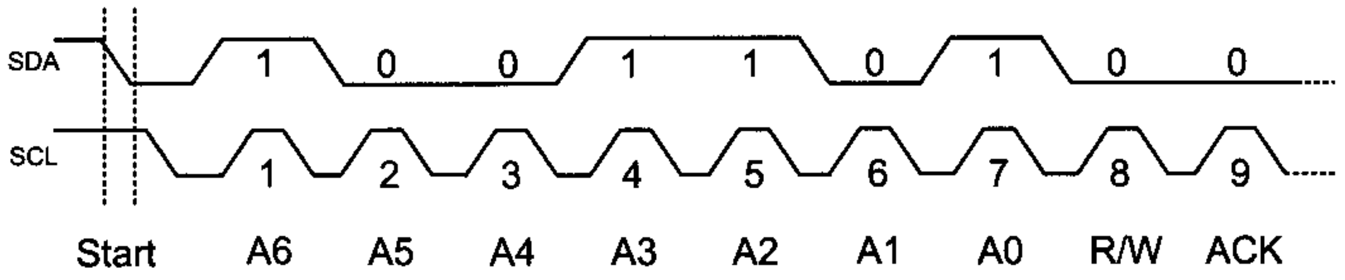
Show how a master says that it wants to write to a slave with address 1001101.

Solution:

The following actions are performed by the master:

(1) The master puts a high-to-low pulse on SDA, while SCL is high to generate a start bit condition to start the transmission.

(2) The master transmits 10011010 into the bus. The first seven bits (1001101) indicates the slave address, and the eighth bit (0) indicates a Write operation and says that the master will write the next byte (data) into the slave.



SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.1 : I2C BUS PROTOCOL

An address packet consisting of a slave address and a READ is called SLA+R, while an address packet consisting of a slave address and a WRITE is called SLA+W.

As we mentioned before, address 0000 000 is reserved for general call. This means that when a master transmits address 0000 000, all slaves respond by changing the SDA line to zero and wait to receive the data byte. This is useful when a master wants to transmit the same data byte to all slaves in the system.

Notice that the general call address cannot be used to read data from slaves because no more than one slave is able to write to the bus at a given time.

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.1 : I2C BUS PROTOCOL

Data packet format

Like other packets, data packets are 9 bits long too. The first 8 bits are a byte of data to be transmitted, and the 9th bit is ACK.

If the receiver has received the last byte of data and there is no more data to be received, or the receiver can-not receive or process more data, it will signal a NACK by leaving the SDA line high. like address packets, MSB is transmitted first.

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.1 : I2C BUS PROTOCOL

Combining address and data packets into a transmission

In I2C, normally, a transmission is started by a START condition, followed by an address packet (SLA + R/W), one or more data packets, and finished by a STOP condition. Figure 18-7 shows a typical data transmission. Try to understand each element in the figure (see Example 18-4).

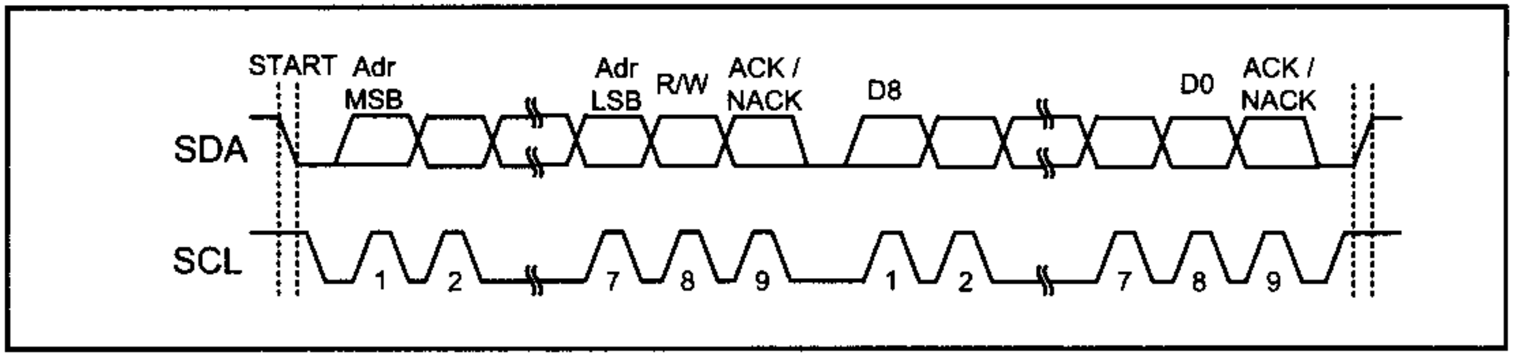
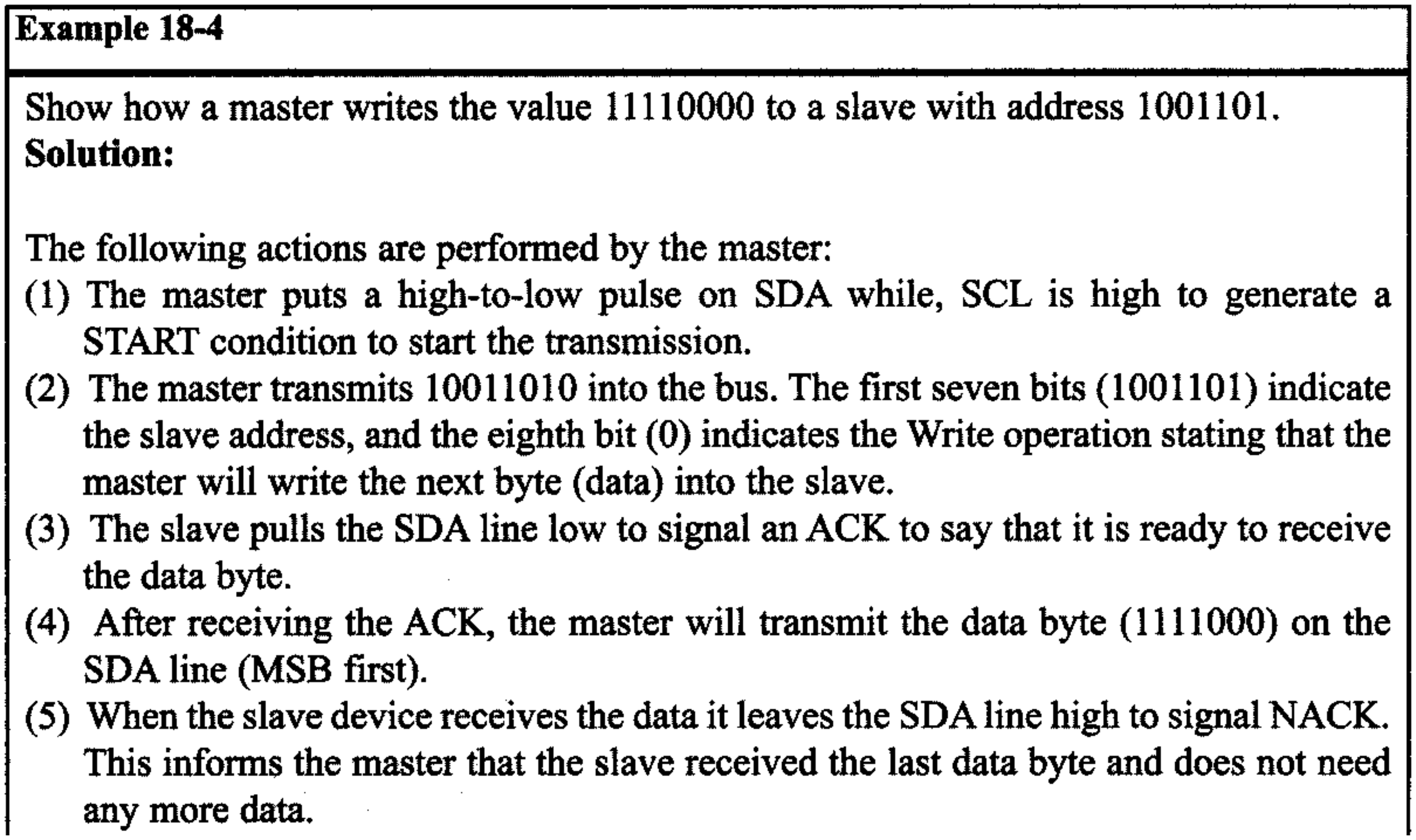


Figure 18-7. Typical Data Transmission

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.1 : I2C BUS PROTOCOL



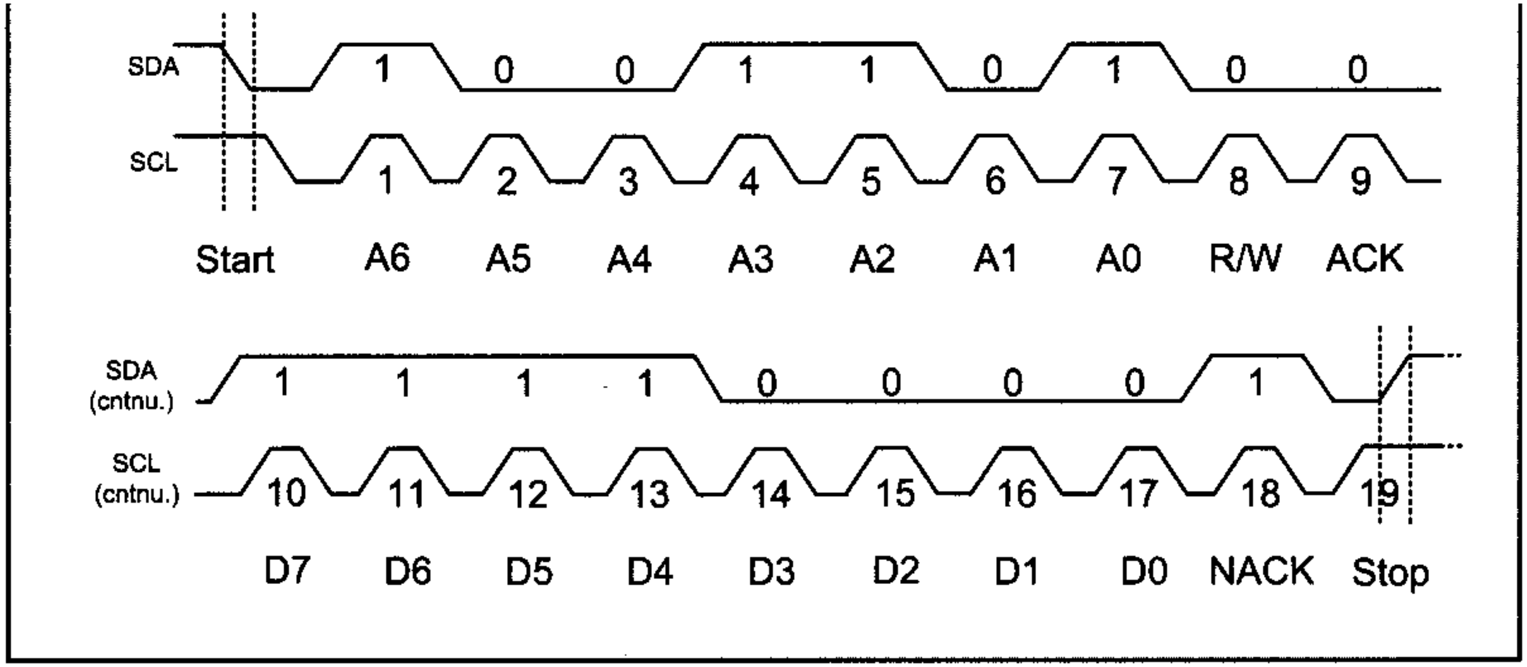
Example 18-4

Show how a master writes the value 11110000 to a slave with address 1001101.
Solution:

The following actions are performed by the master:
(1) The master puts a high-to-low pulse on SDA while, SCL is high to generate a START condition to start the transmission.
(2) The master transmits 10011010 into the bus. The first seven bits (1001101) indicate the slave address, and the eighth bit (0) indicates the Write operation stating that the master will write the next byte (data) into the slave.
(3) The slave pulls the SDA line low to signal an ACK to say that it is ready to receive the data byte.
(4) After receiving the ACK, the master will transmit the data byte (1111000) on the SDA line (MSB first).
(5) When the slave device receives the data it leaves the SDA line high to signal NACK. This informs the master that the slave received the last data byte and does not need any more data.

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.1 : I2C BUS PROTOCOL

(6) After receiving the NACK, the master will know that no more data should be transmitted. The master changes the SDA line when the SCL line is high to transmit a STOP condition and then releases the bus.



SPI PROTOCOL and MAX7221 DISPLAY INTERFACING

SECTION 18.1 : I2C BUS PROTOCOL

Clock stretching

One of the features of the I2C protocol is clock stretching. It is a kind of flow control. If an addressed slave device is not ready to process more data it will stretch the clock by holding the clock line (SCL) low after receiving (or sending) a bit of data. Thus the master will not be able to raise the clock line (because devices are wire-ANDed) and will wait until the slave releases the SCL line to show it is ready to transfer the next bit. See

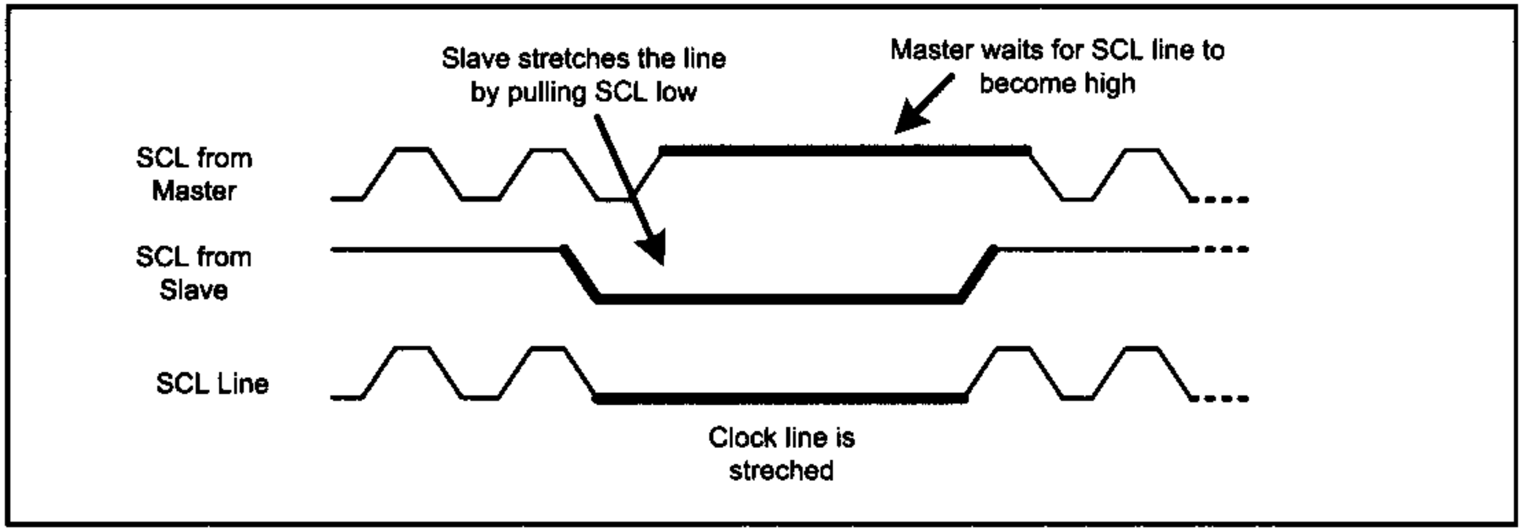

Slave stretches the line by pulling SCL low
Master waits for SCL line to become high
SCL from Master
SCL from Slave
SCL Line
Clock line is streched

Figure 18-8. Clock Stretching

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.1 : I2C BUS PROTOCOL

Arbitration

I2C protocol supports a multimaster bus system. This doesn't mean that more than one master can use the bus at the same time. Rather, each master waits for the current transmission to finish and then starts to use the bus.

But it is possible that two or more masters initiate a transmission at about the same time. In this case the arbitration happens. Each transmitter has to check the level of the bus and compare it with the level it expects; if it doesn't match, that transmitter has lost the arbitration, and will switch to slave mode. In the case of arbitration, the winning master will continue its job. Notice that neither the bus is corrupted nor the data is lost.

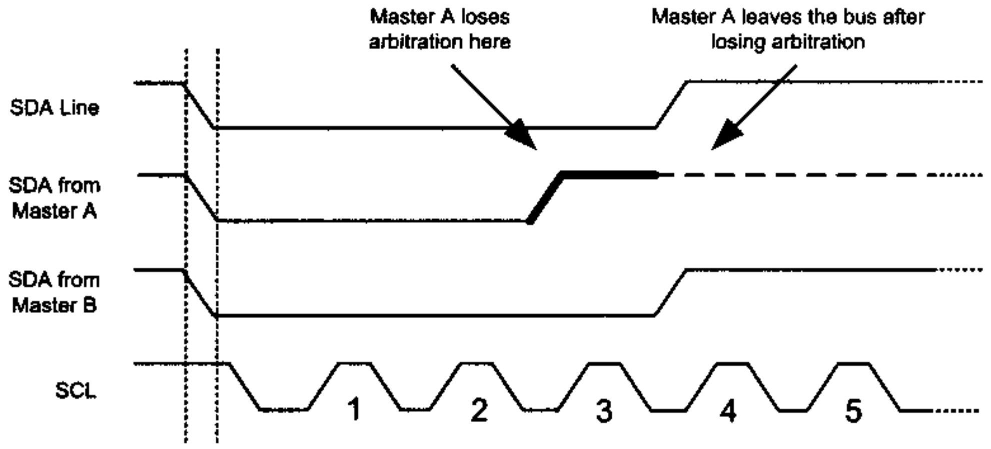
SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.1 : I2C BUS PROTOCOL

Example 18-5

Two masters, A and B, start at about the same time. What happens if master A wants to write to slave 0010 000 and master B wants to write to slave 0001 111?

Solution:

Master A will lose the arbitration in the third clock because the SDA line is different from the output of master A at the third clock. Master A switches to slave mode and leaves the bus after losing the arbitration.


Master A loses arbitration here
Master A leaves the bus after losing arbitration
SDA Line
SDA from Master A
SDA from Master B
SCL
1
2
3
4
5

Multibyte burst write

Burst mode writing is an effective means of loading consecutive locations. It is supported in I2C, SPI, and many other serial protocols. In burst mode, we provide the address of the first location, followed by the data for that location. From then on, consecutive bytes are written to consecutive memory locations.

In this mode, the I2C device internally increments the address location as long as the STOP condition is not detected. The following steps are used to send (write) multiple bytes of data in burst mode for I2C devices.

1. Generate a START condition.
2. Transmit the slave address followed by zero (for write).
3. Transmit the address of the first location.

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.1 : I2C BUS PROTOCOL

4. Transmit the data for the first location and from then on, simply provide consecutive bytes of data to be placed in consecutive memory locations.
5. Generate a STOP condition.

Figure 18-9 shows how to write 0x01, 0x02, and 0x03 to three consecutive locations starting from location 00001111 of slave 1111000.

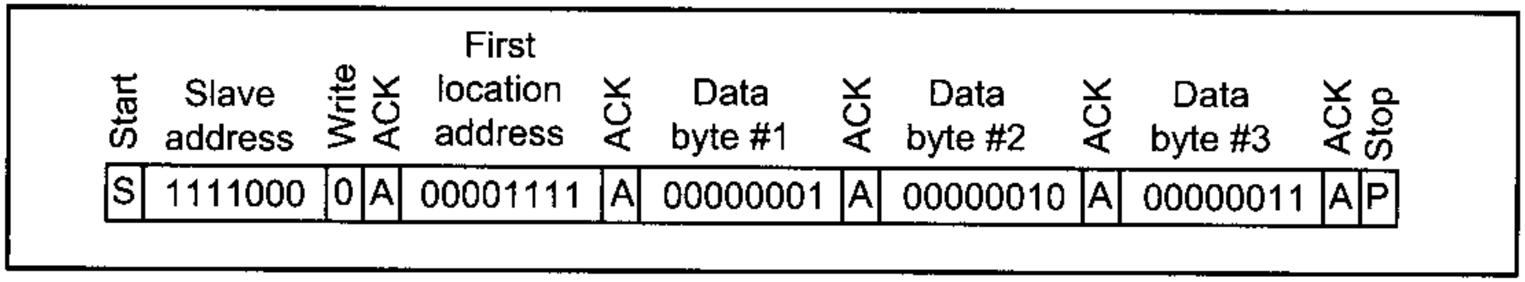


Figure 18-9. Multibyte Burst Write

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.1 : I2C BUS PROTOCOL

Multibyte burst write

Burst mode writing is an effective means of loading consecutive locations. It is supported in I2C, SPI, and many other serial protocols. In burst mode, we provide the address of the first location, followed by the data for that location. From then on, consecutive bytes are written to consecutive memory locations. In this mode, the I2C device internally increments the address location as long as the STOP condition is not detected. The following steps are used to send (write) multiple bytes of data in burst mode for I2C devices.

1. Generate a START condition.
2. Transmit the slave address followed by zero (for address write).
3. Transmit the address of the first location.
4. Generate a START (REPEATED START) condition.
5. Transmit the slave address followed by one (for READ).
6. Read the data from the first location and from then on, bring contents out from consecutive memory locations.
7. Generate a STOP condition.

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.1 : I2C BUS PROTOCOL

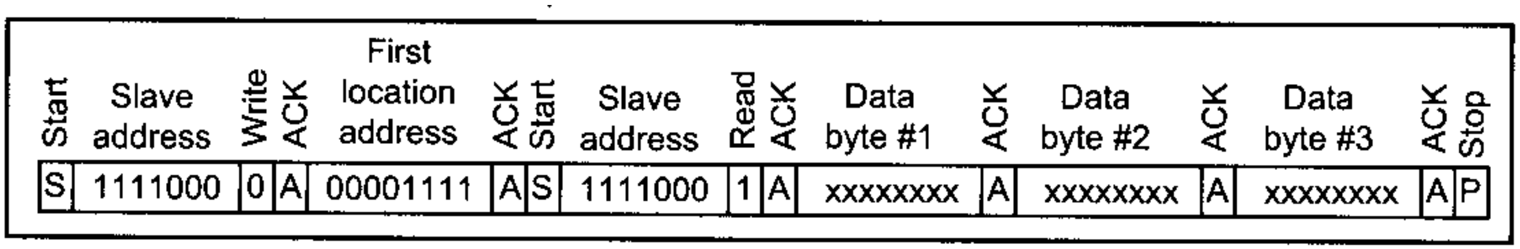
Multibyte burst write

Burst mode writing is an effective means of loading consecutive locations. It is supported in I2C, SPI, and many other serial protocols. In burst mode, we provide the address of the first location, followed by the data for that location. From then on, consecutive bytes are written to consecutive memory locations. In this mode, the I2C device internally increments the address location as long as the STOP condition is not detected. The following steps are used to send (write) multiple bytes of data in burst mode for I2C devices.

1. Generate a START condition. 2. Transmit the slave address followed by zero (for write). 3. Transmit the address of the first location. 4. Transmit the data for the first location and from then on, simply provide con-secutive bytes of data to be placed in consecutive memory locations. 5. Generate a STOP condition. Figure 18-9 shows how to write Ox01, 0x02, and 0x03 to three consecutive locations starting from location 00001111 of slave 1111000.

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.1 : I2C BUS PROTOCOL

Figure 18-10 shows how to read three consecutive locations starting from location 00001111 of slave number 1111000.



Start	Slave address	Write	ACK	First location address	ACK	Start	Slave address	Read	ACK	Data byte #1	ACK	Data byte #2	ACK	Data byte #3	ACK	Stop
S	1111000	0	A	00001111	A	S	1111000	1	A	xxxxxxxx	A	xxxxxxxx	A	xxxxxxxx	A	P

Figure 18-10. Multibyte Burst Read

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.2 : TWI (I2C)) IN THE AVR

TWI (I2C) IN THE AVR

In many applications, including AVR datasheet, I2C is referred to as Two-wire Serial Interface (TWI). From now on, in this course we use TWI to conform with the AVR data sheets.

In this section we discuss the TWI module and registers of the AVR. Then we show how to program the AVR to address a slave device and send or receive data using TWI.

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.2 : TWI (I2C)) IN THE AVR

The TWI module in the AVR is composed of four submodules: bit rate generation unit, bus interface unit, address match unit, and control unit. Figure 18-11 shows the TWI module.

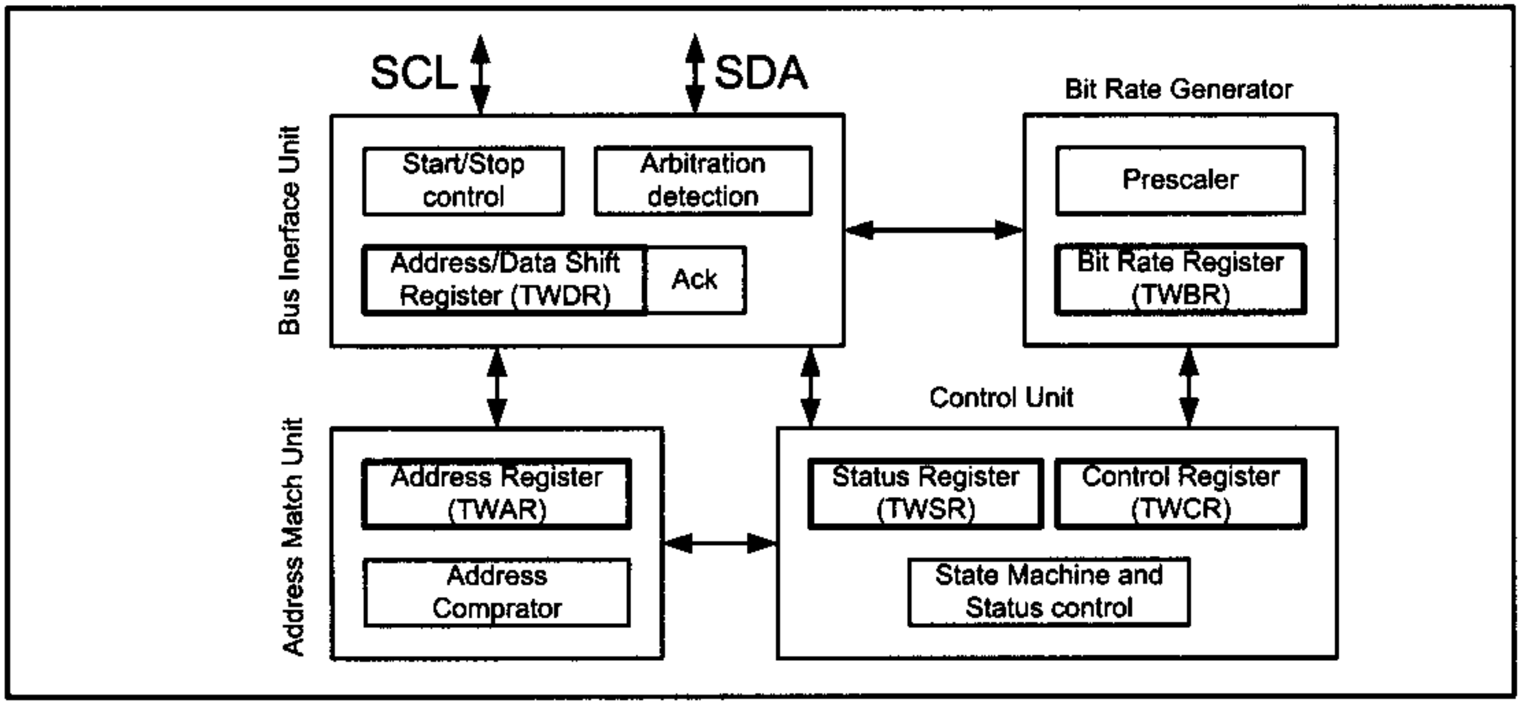


Figure 18-11. TWI (I2C) in AVR

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.2 : TWI (I2C)) IN THE AVR

The bit rate generation unit controls the frequency of the system clock (SCL) when operating in a master mode. The bus interface unit detects and generates START, REPEATED START and STOP conditions. It also detects arbitration, controls sending or receiving ACK, and also transfers packets of data or address.

The address match unit compares the received address byte with the 7-bit address in TWI address register and informs the control unit upon an address match. The control unit controls the TWI module and generates responses according to settings in the TWI control register. It also sets the contents of the status register according to current state.

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.2 : TWI (I2C)) IN THE AVR

In the AVR microcontroller, five major registers are associated with the TWI. They are

TWBR (TWI Bit rate Register),

TWCR (TWI Control Register),

TWSR (TWI Status Register),

TWAR (TWI Address Register), and

TWDR (TWI Data Register)

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.2 : TWI (I2C)) IN THE AVR

TWI Bit Rate Register (TWBR)

The following figure shows the TWBR register and its bits.

TWBR7	TWBR6	TWBR5	TWBR4	TWBR3	TWBR2	TWBR1	TWBR0

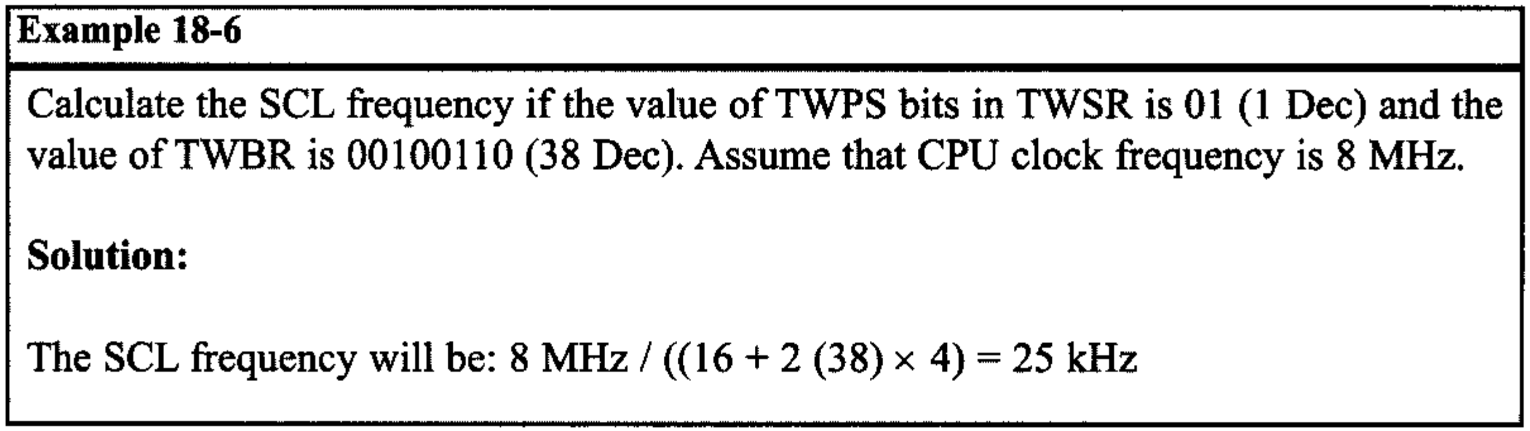
TWBR selects the division factor to control the SCL clock frequency in master mode. The SCL frequency is controlled by settings in the TWBR and the prescaler bits in the TWSR. The following equation demonstrates the relation between SCL frequency, TWBR, and TWPS hilt in TWI status register.

$$\text{SCL frequency} = \frac{\text{CPU Clock frequency}}{16 + 2\,(\text{TWBR}) \times 4^{\text{TWPS}}}$$

Notice that the value of TWBR should be 10 or higher if the TWI operates in master mode.

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.2 : TWI (I2C)) IN THE AVR

Example 18-6 shows how the frequency of SCL is calculated.



Example 18-6

Calculate the SCL frequency if the value of TWPS bits in TWSR is 01 (1 Dec) and the value of TWBR is 00100110 (38 Dec). Assume that CPU clock frequency is 8 MHz.

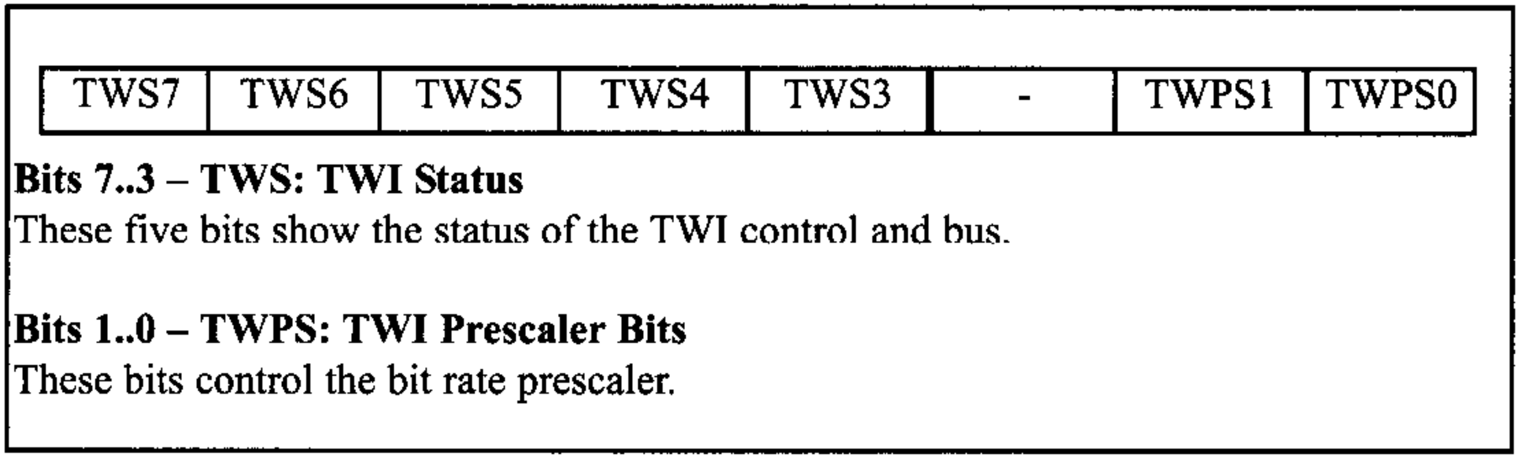
Solution:

The SCL frequency will be: 8 MHz / ((16 + 2 (38) × 4) = 25 kHz

SECTION 18.2 : TWI (I2C)) IN THE AVR

TWI Status Register (TWSR)

As you see in Figure 18-12, five bits of TWSR are dedicated to show the status of the TWI logic and bus. Notice that if you read TWSR, you will read both the status bits and the prescaler value. To check the status bits, you should mask the two LSB bits (prescaler values) to zero. In this course we do not list all of the status codes and their meanings, but we will cover some of more common ones.



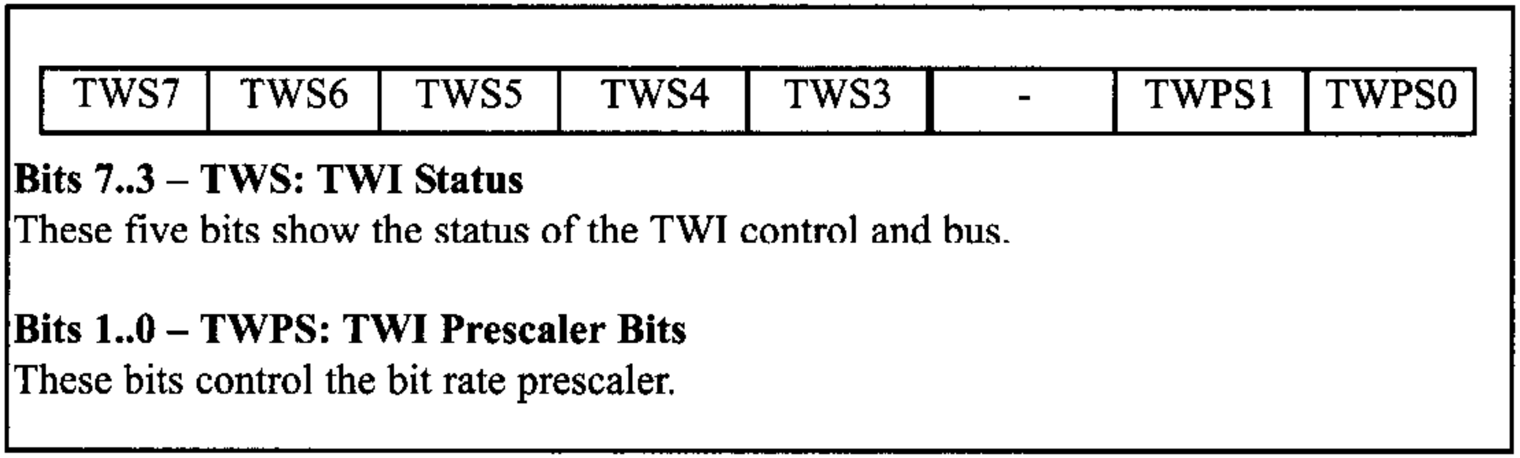
TWS7	TWS6	TWS5	TWS4	TWS3	-	TWPS1	TWPS0

Bits 7..3 – TWS: TWI Status
These five bits show the status of the TWI control and bus.

Bits 1..0 – TWPS: TWI Prescaler Bits
These bits control the bit rate prescaler.

Figure 18-12. TWSR: TWI Status Register

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.2 : TWI (I2C)) IN THE AVR



TWS7	TWS6	TWS5	TWS4	TWS3	-	TWPS1	TWPS0

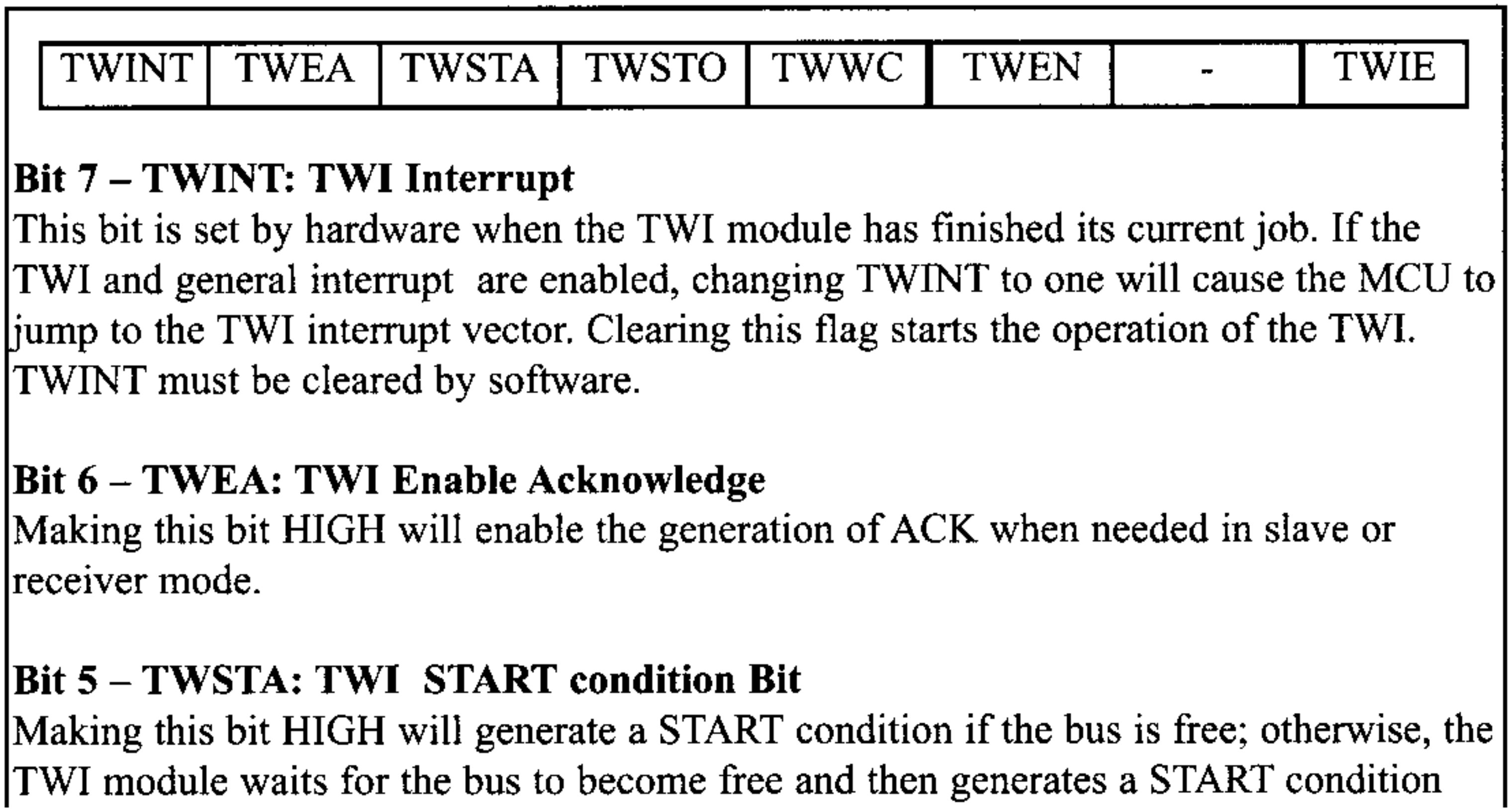
Bits 7..3 – TWS: TWI Status
These five bits show the status of the TWI control and bus.

Bits 1..0 – TWPS: TWI Prescaler Bits
These bits control the bit rate prescaler.

Figure 18-12. TWSR: TWI Status Register

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.2 : TWI (I2C)) IN THE AVR

TWI Control Register (TWCR)



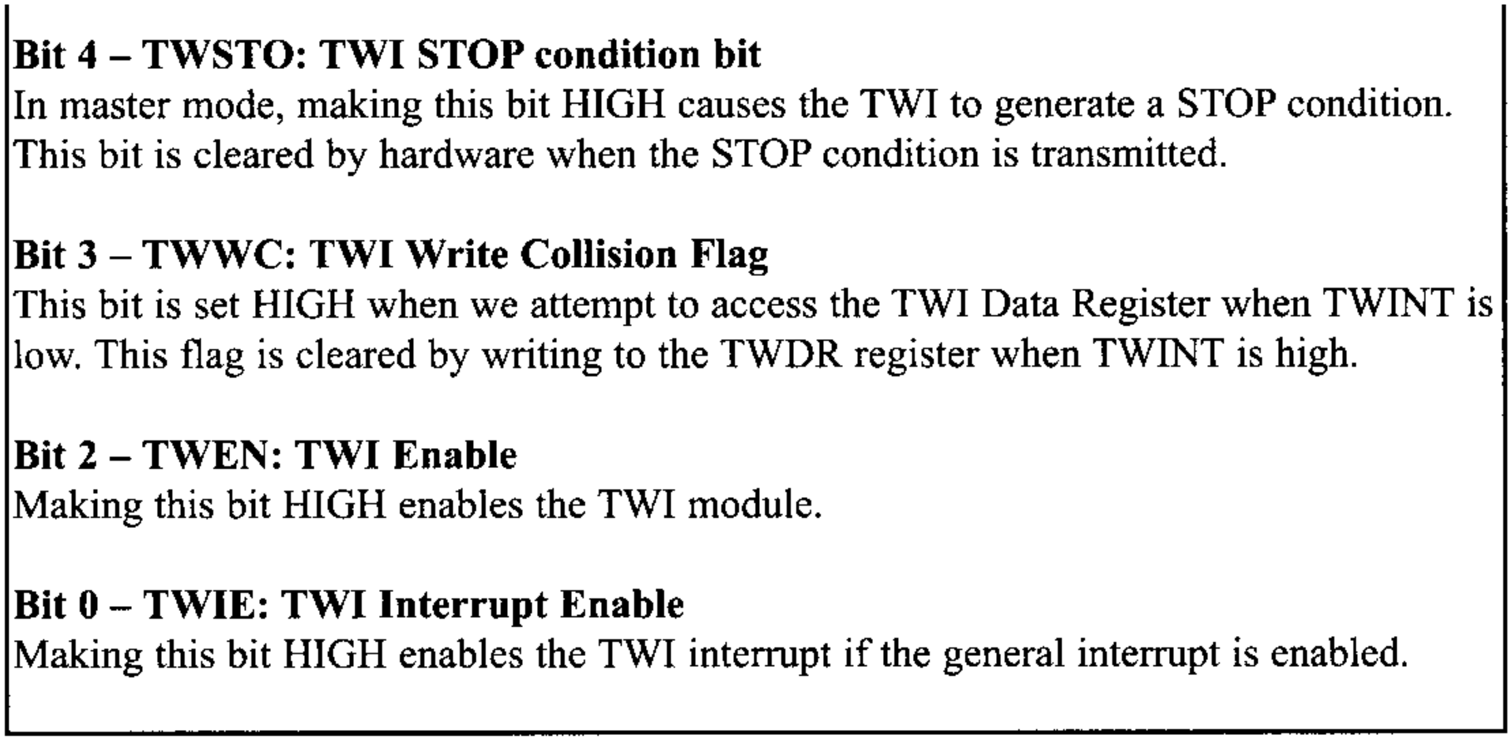
TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	-	TWIE

Bit 7 – TWINT: TWI Interrupt
This bit is set by hardware when the TWI module has finished its current job. If the TWI and general interrupt are enabled, changing TWINT to one will cause the MCU to jump to the TWI interrupt vector. Clearing this flag starts the operation of the TWI. TWINT must be cleared by software.

Bit 6 – TWEA: TWI Enable Acknowledge
Making this bit HIGH will enable the generation of ACK when needed in slave or receiver mode.

Bit 5 – TWSTA: TWI START condition Bit
Making this bit HIGH will generate a START condition if the bus is free; otherwise, the TWI module waits for the bus to become free and then generates a START condition

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.2 : TWI (I2C)) IN THE AVR



Bit 4 – TWSTO: TWI STOP condition bit
In master mode, making this bit HIGH causes the TWI to generate a STOP condition. This bit is cleared by hardware when the STOP condition is transmitted.

Bit 3 – TWWC: TWI Write Collision Flag
This bit is set HIGH when we attempt to access the TWI Data Register when TWINT is low. This flag is cleared by writing to the TWDR register when TWINT is high.

Bit 2 – TWEN: TWI Enable
Making this bit HIGH enables the TWI module.

Bit 0 – TWIE: TWI Interrupt Enable
Making this bit HIGH enables the TWI interrupt if the general interrupt is enabled.

Figure 18-13. TWCR: TWI Control Register

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.2 : TWI (I2C)) IN THE AVR

TWI Interrupt (TWINT) flag

When the TWI hardware finishes its job, it sets the TWINT bit to one. If the TWI and general interrupts are enabled, changing TWINT to HIGH will cause the MCU to jump to the TWI interrupt vector.

When the TWINT bit is set, the TWI module "stretches" the SCL line to provide enough time for software to do specified jobs. When the software finishes its job, it must clear the TWINT bit to resume the operation of the TWI module. Notice that all accesses to the TWI address, status, and data registers must be complete before clearing this flag.

If you try to write to the TWI Data Register when TWINT is low, a collision will happen and the TWI collision flag (TWWC) will be set to HIGH by hardware. Software can monitor (poll) the TWI bit to know when the TWI module finishes its job and is ready for a new command.

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.2 : TWI (I2C)) IN THE AVR

TWI Enable Acknowledge (TWEA) bit

Making this bit HIGH will enable the generation of the ACK bit if any of the following conditions are met:

1. The TWI Address Match module detects that the TWI module is addressed by receiving its own slave address from the bus.
2. A general call has been received while the TWGCE bit in the TWAR is set to one (to enable accepting of global calls).
3. A data byte has been received in each of the receiving modes, master receiver or slave receiver mode.

If you clear the TWEA bit to zero, the device will not generate ACK and will be virtually disconnected from the TWI bus.

TWI Start bit and TWI Stop bit (TWSTA and TWSTO)

To generate START or STOP conditions, you have to set the TWSTA or TWSTO bit to one respectively and then clear the TWINT flag to zero by writing a one to it.

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.2 : TWI (I2C)) IN THE AVR

TWI Data Register (TWDR)

In Receive mode, the last received byte will be in the TWDR, and in Transmit mode, you should write the next byte into TWDR to be transmitted. As we mentioned before, you can access the TWDR only when the TWIE is set to one otherwise collision happens. This means the Data Register cannot be initialized by the user before the first interrupt occurs.

TWI Address Register (TWAR)

TWAR contains the 7-bit slave address to which the TWI will respond when working as slave. The eighth bit (LSB) of TWAR is TWGCE (TWI General Call Recognition Enable). It controls recognition of general call address (00). If this bit is set to one, receiving of a general call address will cause an interrupt request.

AVR TWI PROGRAMMING IN ASSEMBLY and C

Here we will focus on the simplest form of TWI programming without checking the status register. In most applications, if you are not dealing with critical systems and there is not more than one master on a single bus, you can use this method.

If you want to deal with multimaster or critical designs you must check the value of the status flag. In I2C protocol, a device can be either master or slave. In this section we will discuss the steps of programming in each mode. Programming of the AVR TWI in master operating mode To work in master operating mode, we must be able to initialize the TWI, transmit a START condition, send or receive data, and transmit a STOP condition.

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.3 : AVR TWI PROGRAMMING IN ASSEMBLY & C

Initialization To initialize the TWI module to operate in master operating mode, we should do the following steps: 1. Set the TWI module clock frequency by setting the values of the TWBR reg-ister and the TWPS bits in the TWSR register. 2. Enable the TWI module by setting the TWEN bit in the TWCR register to one. Transmit START condition To start data transfer in master operating mode, we must transmit a START condition. This is done by setting the TWEN, TWSTA, and TWINT bits of TWCR to one. Setting the TWEN bit to one enables the TWI module. Setting the TWSTA bit to one tells the TWI to initiate a START condition when the bus is free, and set-ting the TWINT bit to one clears the interrupt flag to initiate operation of the TWI module to transmit the START condition. Then we should poll the TWINT flag in the TWCR register to see whether the START condition transmitted completely.

Send data To send a byte of data, after transmitting the START condition, we should do the following steps: 1. Copy the data byte to the TWDR. 2. Set the TWEN and TWINT bits of the TWCR register to one to start sending the byte. 3. Poll the TWINT flag in the TWCR register to see whether the byte transmitted completely. Notice that right after the START condition, we should transmit SLA + W (Slave Address + Write) or SLA + R (Slave Address + Read). As we mentioned in the first section, right after sending SLA+W we should write to the slave, and right after sending SLA+R we should read from it. To transmit SLA + R, SLA + W, and to write a byte of data to a slave we use a function called I2C_WIRITE.

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.3 : AVR TWI PROGRAMMING IN ASSEMBLY & C

Initialization

To initialize the TWI module to operate in master operating mode, we should do the following steps:

1. Set the TWI module clock frequency by setting the values of the TWBR register and the TWPS bits in the TWSR register.
2. Enable the TWI module by setting the TWEN bit in the TWCR register to one.

Transmit START condition

To start data transfer in master operating mode, we must transmit a START condition. This is done by setting the TWEN, TWSTA, and TWINT bits of TWCR to one.

Setting the TWEN bit to one enables the TWI module. Setting the TWSTA bit to one tells the TWI to initiate a START condition when the bus is free, and setting the TWINT bit to one clears the interrupt flag to initiate operation of the TWI module to transmit the START condition. Then we should poll the TWINT flag in the TWCR register to see whether the START condition transmitted completely.

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.3 : AVR TWI PROGRAMMING IN ASSEMBLY & C

Send data

To send a byte of data, after transmitting the START condition, we should do the following steps:

1. Copy the data byte to the TWDR.
2. Set the TWEN and TWINT bits of the TWCR register to one to start sending the byte.
3. Poll the TWINT flag in the TWCR register to see whether the byte transmitted completely.

Notice that right after the START condition, we should transmit SLA + W (Slave Address + Write) or SLA + R (Slave Address + Read). As we mentioned in the first section, right after sending SLA+W we should write to the slave, and right after sending SLA+R we should read from it. To transmit SLA + R, SLA + W, and to write a byte of data to a slave we use a function called I2C_WIRITE.

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING

SECTION 18.3 : AVR TWI PROGRAMMING IN ASSEMBLY & C

Receive data

To receive a byte of data, after transmitting of SLA + R, we should do the following steps:

1. Set the TWEN and TWINT bits of the TWCR register to one to start receiving a byte. Notice that if you want to return ACK after receiving data you should also set the TWEA bit of the TWCR register to one.
2. Poll the TWINT flag in the TWCR register to see whether a byte has been received completely.
3. Copy the received byte from the TWDR to another register to save it.

Transmit STOP condition

To stop data transfer, we must transmit a STOP condition. This is done by setting the TWEN, TWSTO, and TWINT bits of the TWCR register to one. Notice that we cannot poll the TWINT flag after transmitting the STOP condition.

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.3 : AVR TWI PROGRAMMING IN ASSEMBLY & C

SEE EXAMPLE 18-1

Reading a byte in master mode in Assembly

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.3 : AVR TWI PROGRAMMING IN ASSEMBLY & C

C programming of the AVR TWI in master operating mode

Program 18-3 shows how a master writes 11110000 to a slave with address 1101000. This program is the C version of Program 18-1. Program 18-4 shows how a master reads from a slave with address 1101000 and displays the result on Port A. This program is the C version of Program 18-2.

SEE EXAMPLE 18-3

Writing a byte in master mode in C

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.3 : AVR TWI PROGRAMMING IN ASSEMBLY & C

Programming of the AVR TWI in slave operating mode

To work in slave operating mode, we must be able to initialize the TWI and we must also be able to send or receive data. In slave mode we cannot transmit START or STOP conditions. A slave device should listen to the bus and wait to be addressed by a master device or general call.

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.3 : AVR TWI PROGRAMMING IN ASSEMBLY & C

Initialization

To initialize the TWI module to operate in slave operating mode, we should do the following steps:

1. Set the slave address by setting the values for the TWAR registers. As we mentioned before, the upper seven bits of TWAR are the slave address, and the eighth bit is TWGCE. If you set this bit to one, the TWI will respond to the general call address ($00); otherwise, it will ignore the general call address.
2. Enable the TWI module by setting the TWEN bit in the TWCR register to one.
3. Set the TWEN, TWINT, and TWEA bits of TWCR to one to enable the TWI and acknowledge generation.

Notice that we cannot combine steps 2 and 3 into a single step. We have to enable the TWI module before doing the third step.

SECTION 18.3 : AVR TWI PROGRAMMING IN ASSEMBLY & C

Listen to the bus

After initializing the TWI module, a slave device should listen to the bus to detect when it is addressed by a master device. When the TWI module detects its own address on the bus, it returns ACK and then sets the TWINT flag in the TWCR register to one. We should poll the TWINT flag to see when the slave is addressed by a master device.

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING

SECTION 18.3 : AVR TWI PROGRAMMING IN ASSEMBLY & C

Send data

After being addressed by a master device for read, we should do the following steps to send a byte of data:

1. Copy the data byte to the TWDR.
2. Set the TWEN, TWEA, and TWINT bits of the TWCR register to one to start sending the byte. Notice that if you expect not to receive ACK after receiving data you can leave the TWEA cleared. It will have no effect on generation of ACK by master and will only change the internal state of the TWI module We recommend that you set the TWEA bit of the TWCR register to one anyway.
3. Poll the TWINT flag in the TWCR register to see when the byte is completely transmitted.

Receive data

After being addressed by a master device, we should do the following steps to receive a byte of data:

1. Set the TWEN and TWINT bits of the TWCR register to one to start receiving a byte. Notice that if you want to return ACK after receiving data you should also set the TWEA bit of the TWCR register to one.
2. Poll the TWINT flag in the TWCR register to see whether a byte has been received completely.
3. Copy the received byte from the TWDR to another register to save it.

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.3 : AVR TWI PROGRAMMING IN ASSEMBLY & C

Programs 18-5 and 18-6 show how to initialize the TWI module to operate in slave mode. In Program 18-5 the TWI module listens to the bus and waits to be addressed by a master device. Then it transmits the letter '0' to the master device.

See Example 18-5

Writing a byte in slave mode

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.3 : AVR TWI PROGRAMMING IN ASSEMBLY & C

In Program 18-6 the TWI module listens to the bus and waits to be addressed by a master device. Then it reads a byte of data from the master device and displays it on Port A.

See Example 18-6
Reading a byte in Slave Mode

SECTION 18.3 : AVR TWI PROGRAMMING IN ASSEMBLY & C

C programming of the AVR TWI in slave operating mode

Program 18-7 is the C version of Program 18-5. Program 18-7 shows how to initialize the TWI module to operate in slave mode. In Program 18-7 the TWI module listens to the bus and waits to be addressed by a master device. Then it transmits the letter '0' to the master device.

See Example 18-7

Writing a byte in slave mode in C

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.3 : AVR TWI PROGRAMMING IN ASSEMBLY & C

Program 18-8 is the C version of Program 18-6. In Program 18-8 the TWI module listens to the bus and waits to be addressed by a master device. Then it reads a byte of data from the master device and displays it on Port A.

See Example 18-8

Reading a Byte in Slave Mode in C

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.4 : DS1307 RTC INTERFACING AND PROGRAMMING

The real-time clock (RTC) is a widely used device that provides accurate time and date information for many applications. Many systems such as the x86 PC come with such a chip on the motherboard. The RTC chip in the x86 PC provides the time components of hour, minute, and second, in addition to the date/calendar components of year, month, and day.

Many RTC chips use an internal battery, which keeps the time and date even when the power is off. Although some microcontrollers, such as the DS5000T and some of AVRs, come with the RTC already embedded into the chip, we have to interface the vast majority of them to an external RTC chip.

One of the most widely used RTC chips is the DS12887 from Dallas Semiconductor/Maxim Corp. This chip is found in the vast majority of x86 PCs. The original IBM PC/AT used the MC14618B RTC from Motorola (now Freescale). The DS 12887 is the replacement for that chip. It uses an internal lithium battery to keep operating for over 10 years in the absence of external power. The DS12887 is a parallel RTC with 8 pins for the data bus.

SECTION 18.4 : DS1307 RTC INTERFACING AND PROGRAMMING

The DS1307 is a serial RTC with an 12C bus. In this section, we interface and program the DS1307 RTC. According to the DS1307 data sheet from Maxim, "The clock/calendar provides seconds, minutes, hours, day, date, month, and year information. The end of the month date is automatically adjusted for months with fewer than 31 days, including corrections for leap year.

The clock operates in either the 24-hour or 12-hour format with AM/PM indicator. The DS1307 has a built-in power-sense circuit that detects power failures and automatically switches to the battery supply." The DS1307 does not support the Daylight Savings Time option.

SECTION 18.4 : DS1307 RTC INTERFACING AND PROGRAMMING

Next, we describe the pins of the DS1307. See Figure 18-14.

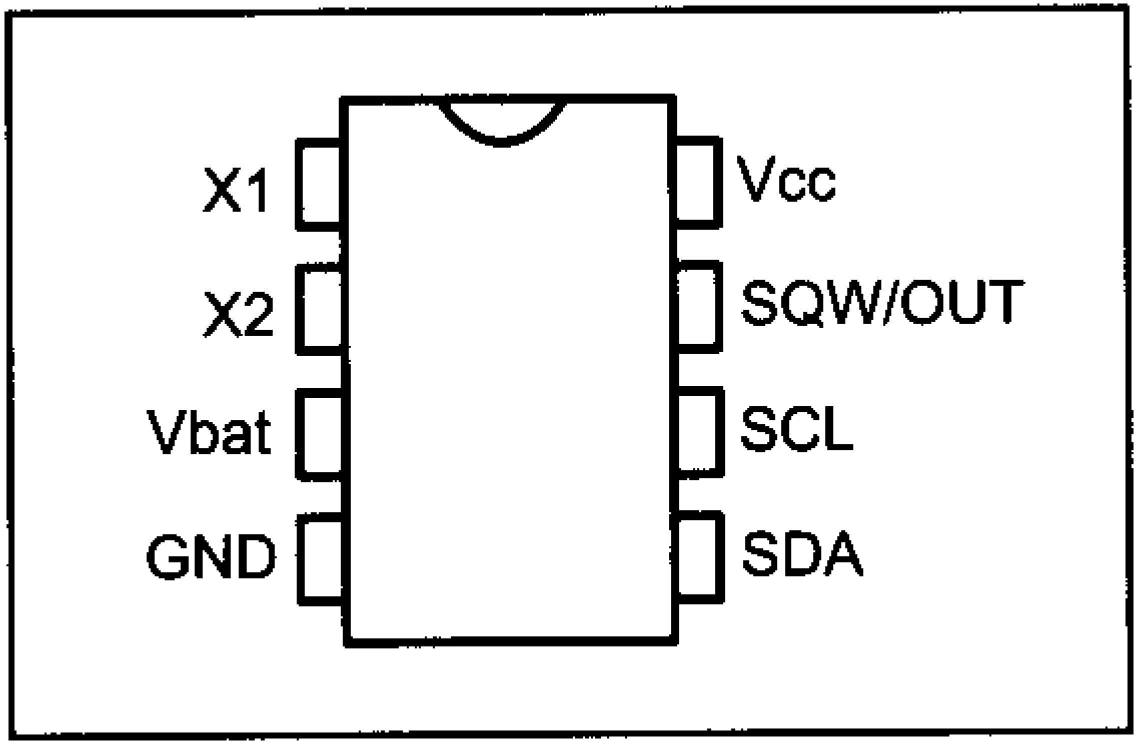


Figure 18-14. DS1307 Pin Out

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.4 : DS1307 RTC INTERFACING AND PROGRAMMING

X1-X2

These are input pins that allow the DS1307 connection to an external crystal oscillator to provide the clock source to the chip. We must use the standard 32.768 kHz quartz crystal. The accuracy of the clock depends on the quality of this crystal oscillator. Heat can cause a drift on the oscillator. To avoid this, we can use the DS32KHZ chip, which automatically adjusts for temperature variations. Notice that when using the DS32KHZ or similar clock generator s. We only need to connect X1 because the X2 loopback is required.

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.4 : DS1307 RTC INTERFACING AND PROGRAMMING

Vbat

Pin 3 can be connected to an external +3 V lithium battery, thereby providing the power source to the chip when the external supply voltage is not available. We must connect this pin to ground if it is not used. A 48mAhr lithium battery can provide the power needed for more than 10 years to back up the chip.

GND

Pin 4 is the ground.

SDA (Serial Data)

Pin 5 is the SDA pin and must be connected to the SDA line of the I2C bus.

SCL (Serial Clock)

Pin 6 is the SCL pin and must be connected to the SCL line of the I2C bus.

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.4 : DS1307 RTC INTERFACING AND PROGRAMMING

SWQ/OUT

Pin 7 is an output pin providing 1 kHz, 4 kHz, 8 kHz, or 32 kHz frequency if enabled. This pin needs an external pull-up resistor to generate the frequency because it is open drain.

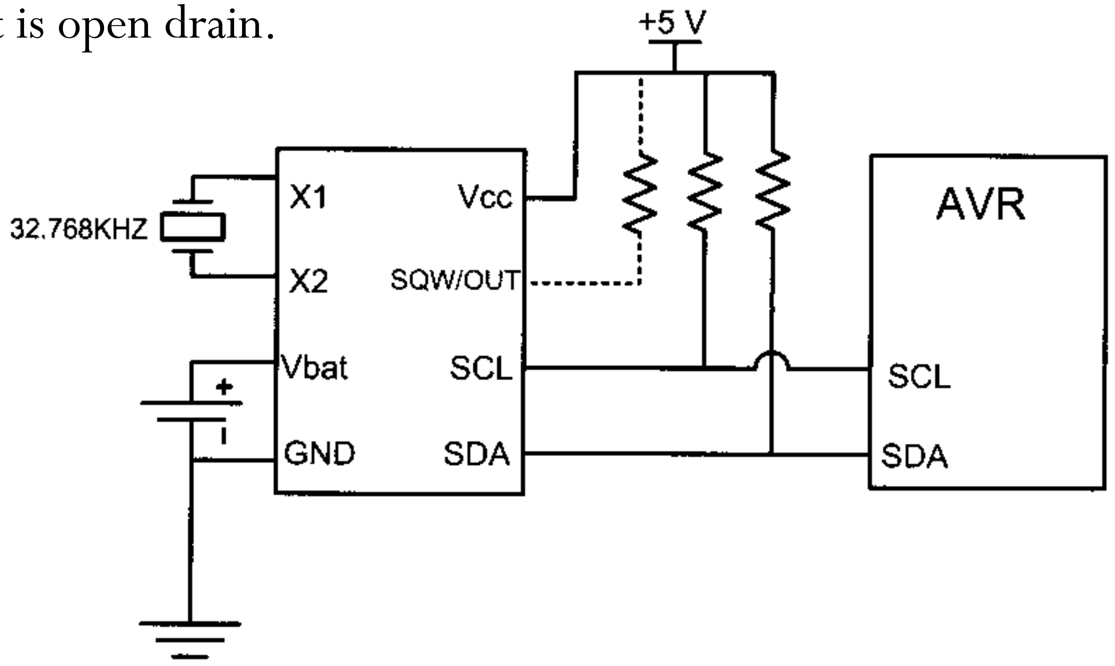


Figure 18-15. DS1307 Power Connection Options

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.4 : DS1307 RTC INTERFACING AND PROGRAMMING

Vcc

Pin 8 is used as the primary voltage supply to the chip. This primary voltage source is generally set to +5 V. When Vcc falls below the Vbat level, the DS1307 switches to Vbat and the external lithium battery provides power to the RTC.

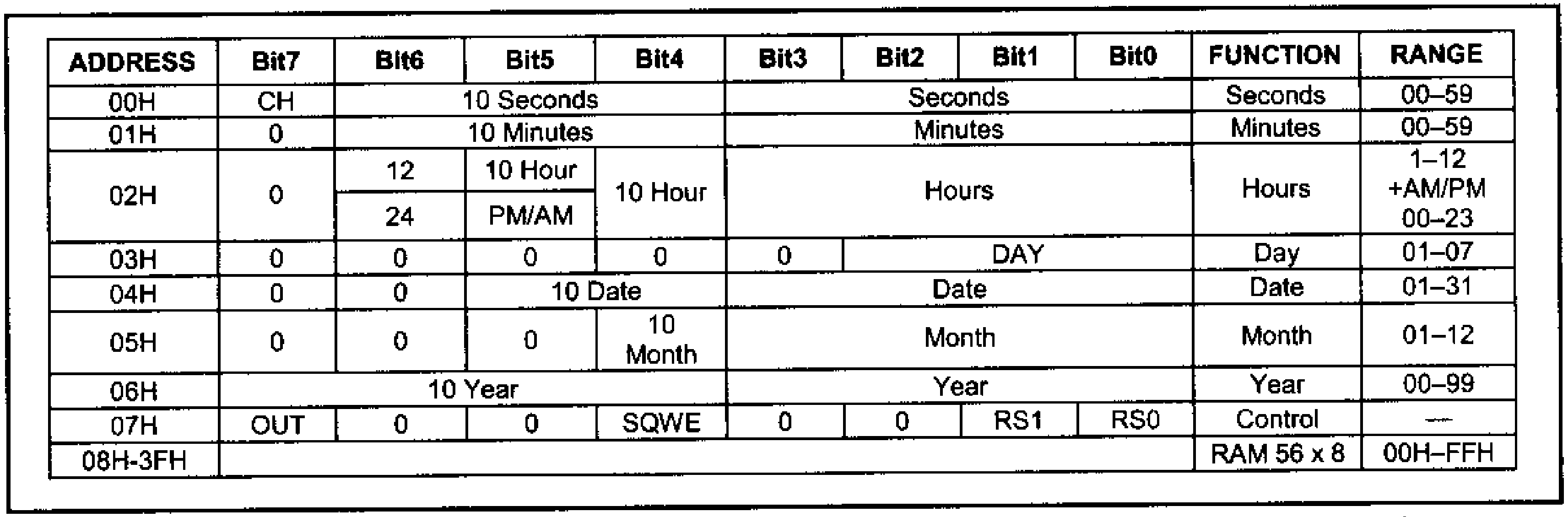
According to the DS1307 data sheet, "upon power-up, the device switches from Vbat to Vccl when Vcc1 is greater than Vbat+0.2 Volts." Also notice that the device is accessible only when Vcc is more than 1.25 x Vbat. Because we can connect the standard 3 V lithium battery to the Vbat pin, the Vcc voltage level must remain above 3.2 V in order for the Vcc to remain as the primary voltage source to the chip, and it must be more than 3.75 V if you want to access the chip.

Address map of the DS1307

The DS1307 has a total of 64 bytes of RAM space with addresses 00-3FH. The first seven locations, 00-06, are set aside for RTC values of time and date. The next byte is used for the control register. It is located at address 07 in hex. That leaves 56 bytes, from addresses 07H to 3FH, available for general-purpose data storage. That means the entire 64 bytes of RAM are accessible directly for read or write.

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.4 : DS1307 RTC INTERFACING AND PROGRAMMING

Figure 18-16 shows the address map of the DS1307.



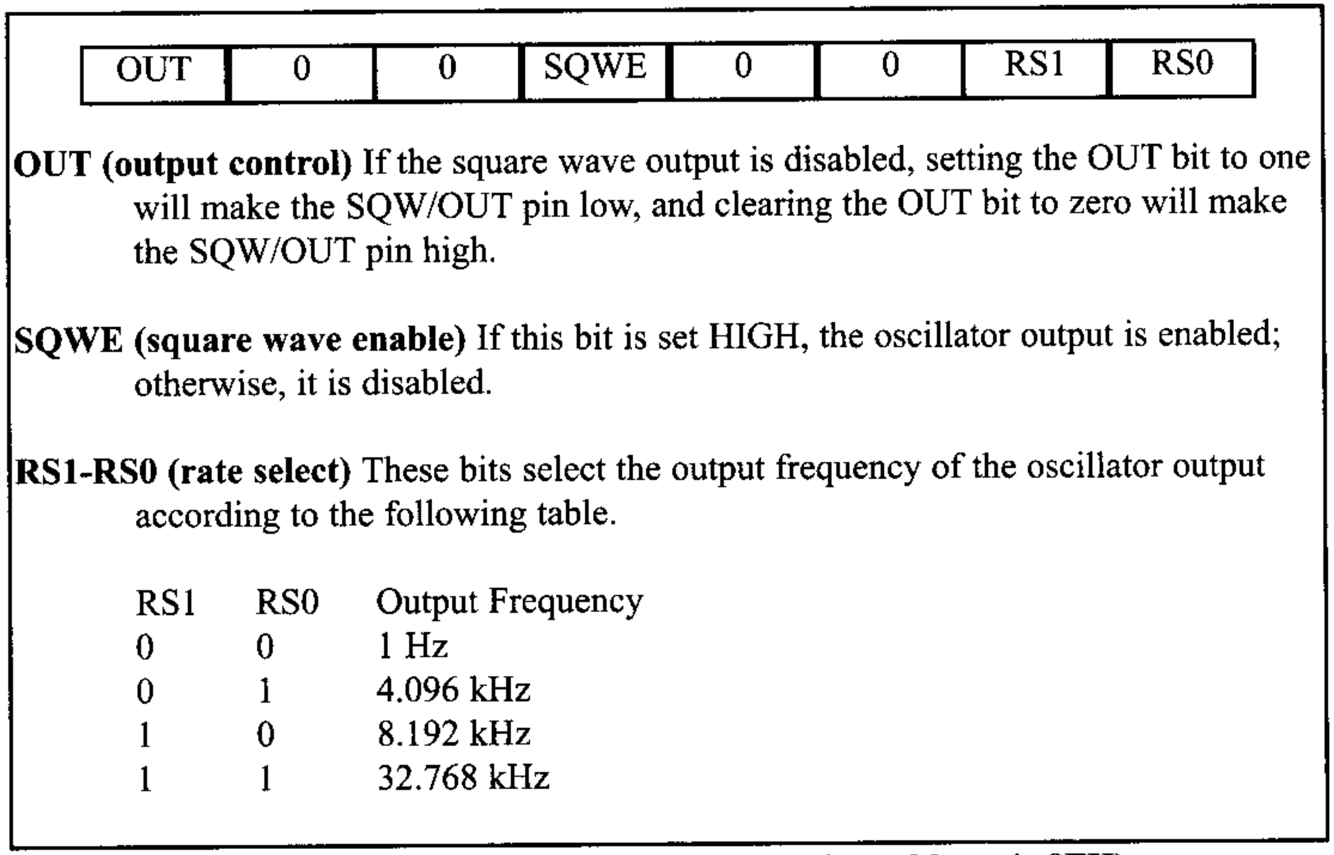
ADDRESS	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	FUNCTION	RANGE
00H	CH	10 Seconds			Seconds				Seconds	00–59
01H	0	10 Minutes			Minutes				Minutes	00–59
02H	0	12 24	10 Hour PM/AM	10 Hour	Hours				Hours	1–12 +AM/PM 00–23
03H	0	0	0	0	0	DAY			Day	01–07
04H	0	0	10 Date		Date				Date	01–31
05H	0	0	0	10 Month	Month				Month	01–12
06H	10 Year				Year				Year	00–99
07H	OUT	0	0	SQWE	0	0	RS1	RS0	Control	—
08H-3FH									RAM 56 x 8	00H–FFH

Figure 18-16. Simplified Block Diagram of DS1307 (Maxim/Dallas Semiconductor)

SECTION 18.4 : DS1307 RTC INTERFACING AND PROGRAMMING

The DS1307 control register

As shown in Figure 18-16, the control register has an address of 07H. In the DS1307 control register, the bits control the function of the SQW/OUT pin



OUT	0	0	SQWE	0	0	RS1	RS0

OUT (output control) If the square wave output is disabled, setting the OUT bit to one will make the SQW/OUT pin low, and clearing the OUT bit to zero will make the SQW/OUT pin high.

SQWE (square wave enable) If this bit is set HIGH, the oscillator output is enabled; otherwise, it is disabled.

RS1-RS0 (rate select) These bits select the output frequency of the oscillator output according to the following table.

RS1	RS0	Output Frequency
0	0	1 Hz
0	1	4.096 kHz
1	0	8.192 kHz
1	1	32.768 kHz

Figure 18-17. DS1307 Control Register (Write location address is 8FH)

SECTION 18.4 : DS1307 RTC INTERFACING AND PROGRAMMING

CH bit in address 00

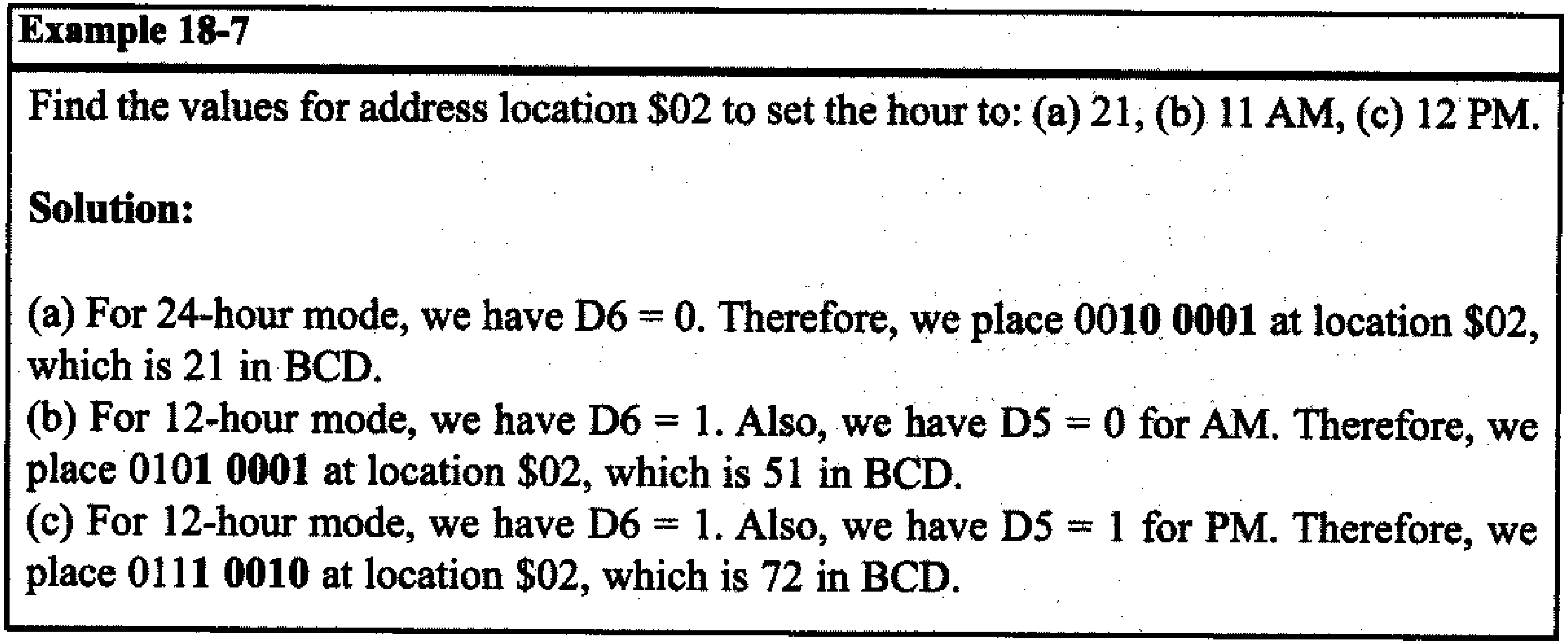
One of the most important bits in the Seconds address location in the DS 1307 is the CH (Clock Halt) bit. It is the seventh bit of address location 00. Setting the CH bit to one disables the oscillator, while setting CH to zero enables the oscillator. The CH bit is undefined upon reset. In order to enable the oscillator, we must clear the CH during initial configuration.

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.4 : DS1307 RTC INTERFACING AND PROGRAMMING

Time and date address locations and modes

The byte addresses 0-6 are set aside for the time and date, as shown in Figure 18-16. The DS 1307 provides data in BCD format only. Notice the data range for the hour mode. We can select 12-hour or 24-hour mode with bit 6 of hour location 02. When D6 = 1, the 12-hour mode is selected, and D6 = 0 provides us the 24-hour mode. In the 12-hour mode, we decide the AM and PM with the bit 5. If D5 = 0, the AM is selected and D5 = 1 is for the PM. See Example 18-7.

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.4 : DS1307 RTC INTERFACING AND PROGRAMMING



Example 18-7

Find the values for address location $02 to set the hour to: (a) 21, (b) 11 AM, (c) 12 PM.

Solution:

(a) For 24-hour mode, we have D6 = 0. Therefore, we place **0010 0001** at location $02, which is 21 in BCD.
(b) For 12-hour mode, we have D6 = 1. Also, we have D5 = 0 for AM. Therefore, we place 0101 **0001** at location $02, which is 51 in BCD.
(c) For 12-hour mode, we have D6 = 1. Also, we have D5 = 1 for PM. Therefore, we place 0111 **0010** at location $02, which is 72 in BCD.

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.4 : DS1307 RTC INTERFACING AND PROGRAMMING

Register pointer In DS 1307

there is a register pointer that specifies the byte that will be accessed in the next read or write command. After each read or write operation, the content of the register pointer is automatically incremented. It is useful in multibyte read or write. Writing to DS1307 To set the value of the register pointer and write one or more bytes of data to DS1307, you can use the following steps:

1. 1. To access the DS 1307 for a write operation, after sending a START condition, you should transmit the address of DS 1307 (1001101) followed by 0 to indicate a write operation.
2. 2. The first byte of data in the write operation will set the register pointer. For example, if you want to access the control register you should send 0x07.
3. 3. If you want only to set the register pointer you should skip this step. If you want to write one or more bytes of data, you should transmit them one byte at a time. Remember that the register pointer is automatically incremented and you can simply transmit bytes of data to consecutive locations in a multibyte burst write.
4. 4. Transmit a STOP bit condition.

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.4 : DS1307 RTC INTERFACING AND PROGRAMMING

Reading from DS1307

Notice that before reading a byte you should load the address of the byte to the register pointer by doing a write operation as mentioned before. To read one or more bytes of data from the DS 1307 you should do the following steps:

1. To access the DS 1307 for a read operation, after sending a START condition, you should transmit the address of DS1307 (1001101) followed by 1 to indicate a read operation.

2. Now you can read one or more bytes of data. Remember that the register point-er indicates which address will be read. Also notice that the register pointer is automatically incremented and you can simply receive consecutive bytes of data in a multibyte burst read.

3. Transmit a STOP bit condition.

Setting the time in Assembly Program 18-9 initializes the clock at 16:58:55 using the 24-hour clock mode. It uses the single-byte operation for writing into the control register of the DS1307 and Multibyte burst mode for writing seconds, minutes, and hours. Notice that in this program we assume that there is only one master on the bus and we do not deal with checking the status register.

SECTION 18.4 : DS1307 RTC INTERFACING AND PROGRAMMING

Setting the time in Assembly

Program 18-9 initializes the clock at 16:58:55 using the 24-hour clock mode. It uses the single-byte operation for writing into the control register of the DS1307 and Multibyte burst mode for writing seconds, minutes, and hours. Notice that in this program we assume that there is only one master on the bus and we do not deal with checking the status register.

See Example 18-9

Setting the Time in Assembly

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.4 : DS1307 RTC INTERFACING AND PROGRAMMING

Setting the date in Assembly

Program 18-10 shows how to set the date to October 19th, 2009. It uses the single-byte operation for writing into the control register of the DS1307 and Multibyte burst mode for writing day, month, and year. As you can see in the program, to access the location of the date, you should write 0x04 into the register pointer and then you can use Multibyte burst write to write the values of month and year in the consecutive locations. Also, notice that in this code we assume that there is only one master on the bus and we do not deal with checking the status register.

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.4 : DS1307 RTC INTERFACING AND PROGRAMMING

See Example 18-10

Setting the Date in Assembly

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.4 : DS1307 RTC INTERFACING AND PROGRAMMING

Setting the time in C

Programs 18-11 and 18-12 are the C versions of the last two programs. Notice that you have to make the optimization level 00 (optimization 0); otherwise, the compiler would omit the line "for (int k = 0 ; k<100 ; k++) " and the program would not work correctly.

See Example 18-11
Setting the time in C

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.4 : DS1307 RTC INTERFACING AND PROGRAMMING

See Example 19-12

Setting the Date in C

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.4 : DS1307 RTC INTERFACING AND PROGRAMMING

Setting, reading, and displaying time and date in C

Program 18-13 is the complete C code for setting, reading, and displaying the time and date. The times and dates are sent to the IBM PC screen via the serial port after they are converted from packed BCD to ASCII.

See Example 18-13

A complete DS1307 Code Example in C

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING

SECTION 18.5 : TWI PROGRAMMING WITH CHECKING STATUS REGISTER

In this section we discuss TWI programming with checking the value of status register. By checking the value of the status register you can monitor the TWI module current state and operation. This helps you to detect an error when it happens and resolve it at the same time. This is an advanced topic and used only if you are connecting 12C to multiple masters.

As we mentioned before, there are four modes of operation

- master transmitter,
- master receiver,
- slave transmitter, and
- slave receiver.

We will discuss each mode separately because each mode has its own special status codes. For each mode of operation there is a flowchart that shows the sequence of steps in each mode and also a figure that summarizes most of the status values for each mode in a single table.

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.5 : TWI PROGRAMMING WITH CHECKING STATUS REGISTER

I2C BUS

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING

SECTION 18.5 : TWI PROGRAMMING WITH CHECKING STATUS REGISTER

I2C BUS

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING

SECTION 18.5 : TWI PROGRAMMING WITH CHECKING STATUS REGISTER

I2C BUS

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING

SECTION 18.5 : TWI PROGRAMMING WITH CHECKING STATUS REGISTER

I2C BUS

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING

SECTION 18.5 : TWI PROGRAMMING WITH CHECKING STATUS REGISTER

I2C BUS

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING

SECTION 18.5 : TWI PROGRAMMING WITH CHECKING STATUS REGISTER

I2C BUS

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.5 : TWI PROGRAMMING WITH CHECKING STATUS REGISTER

I2C BUS

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.5 : TWI PROGRAMMING WITH CHECKING STATUS REGISTER

I2C BUS

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.5 : TWI PROGRAMMING WITH CHECKING STATUS REGISTER

I2C BUS

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.5 : TWI PROGRAMMING WITH CHECKING STATUS REGISTER

I2C BUS

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING

SECTION 18.5 : TWI PROGRAMMING WITH CHECKING STATUS REGISTER

I2C BUS

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING

SECTION 18.5 : TWI PROGRAMMING WITH CHECKING STATUS REGISTER

I2C BUS

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING

SECTION 18.5 : TWI PROGRAMMING WITH CHECKING STATUS REGISTER

I2C BUS

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING

SECTION 18.5 : TWI PROGRAMMING WITH CHECKING STATUS REGISTER

I2C BUS

SPI PROTOCOL and MAX7221 DISPLAY INTERFACING
SECTION 18.5 : TWI PROGRAMMING WITH CHECKING STATUS REGISTER

I2C BUS