

APPLICATION NOTE 3661

# The MAX3420E Interrupt System

*Abstract: The MAX3420E adds USB peripheral functionality to any SPI master such as a microcontroller. MAX3420E operation is largely defined by the interrupt request (IRQ) bits through which it alerts the SPI master that a USB event requires service. This note describes the MAX3420E interrupt system and every interrupt request bit.*

## Introduction

The [MAX3420E](#) connects to any SPI master to implement a full-speed USB peripheral device. While the MAX3420 manages the low-level USB signaling work, the SPI master must occasionally get involved when a USB event requires attention. An INT pin on the MAX3420 indicates that attention is required, and the SPI master reads 14 interrupt request bits to determine which interrupts require service. These Interrupt ReQuest (IRQ) bits largely define the MAX3420E's operation.

---

**Note:** The SPI master can be a microcontroller, DSP, ASIC, or anything that can implement an SPI port and supply the SCLK signal. This document uses the terms "SPI master" and "microcontroller" interchangeably.

---

## MAX3420E Interrupt Logic

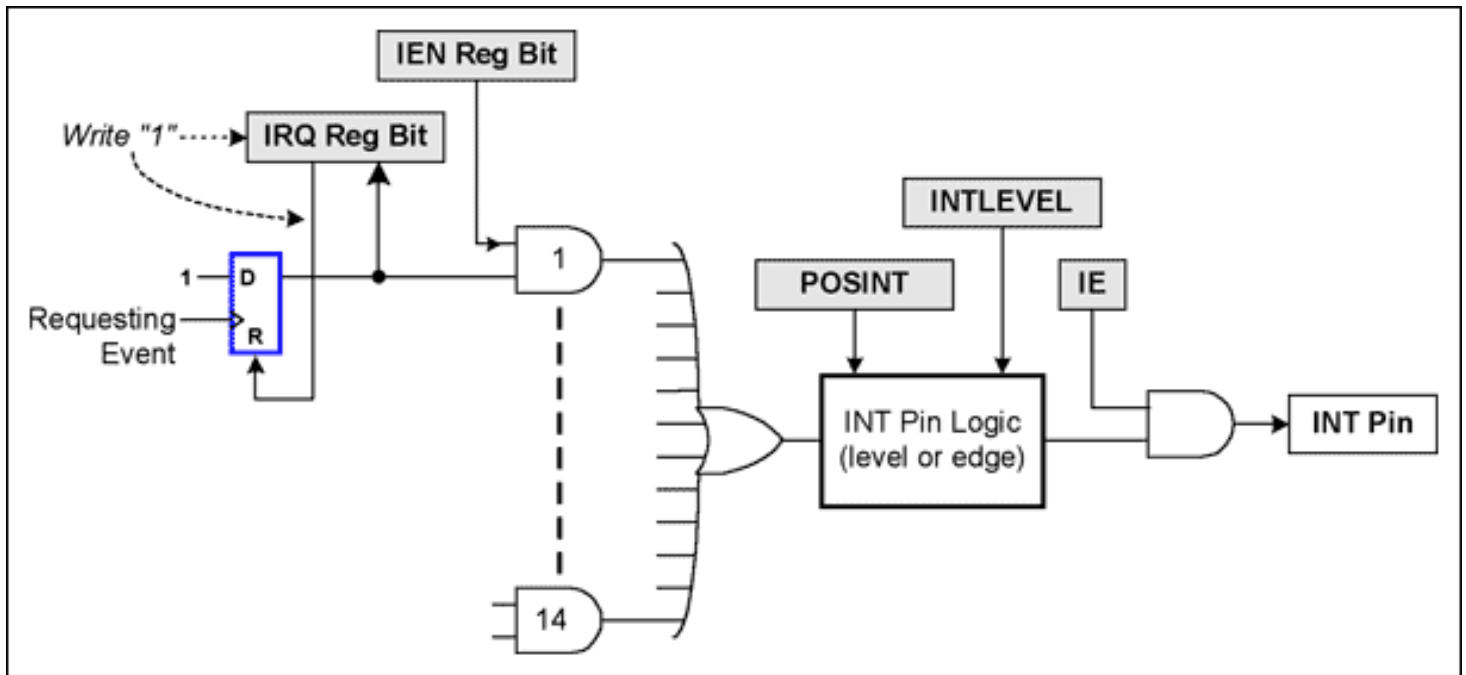


Figure 1. MAX3420E interrupt logic. Register bits are shaded.

**Figure 1** illustrates the MAX3420E interrupt logic. SPI-accessible register bits are shaded.

### IRQ Bits

Every interrupt source has a flip-flop to latch the service request. The output of this flip-flop is an IRQ, which appears in a MAX3420E register. IRQ bits have two properties:

1. **Reading** an IRQ bit returns the state of the IRQ flip-flop.
2. **Writing** a '1' to an IRQ bit clears its IRQ flip-flop, and writing a '0' to an IRQ bit leaves the flip-flop unchanged.

IRQ bits, which may be read at any time, reflect the state of the IRQ flip-flop. Following #2 above, the writing of a 1 or 0 clears selected IRQ bits without requiring a read-modify-write cycle. To illustrate, suppose that the MAX3420E implemented IRQ bits as simple register bits, where writing 1 sets the bit and writing a 0 clears it. Now we want to clear the URESIRQ bit in the USBIRQ register. **Figure 2** shows the code to do this.

```
#define      rUSBIRQ    13    // register 13
#define      bmURESIRQ  0x08 // URESIRQ is bit 4, bm means "bit mask"

unsigned char dum;

dum = rreg(rUSBIRQ);           // read the register
dum = dum & ~bmURESIRQ;       // clear one bit
wreg(rUSBIRQ, dum);           // write it back
```

Figure 2. Clearing a conventional register bit requires a RMW operation.

Because an SPI master clears a MAX3420E IRQ bit by writing a 1, and because a 0 leaves the other register bits unchanged, the SPI master can clear the URESIRQ bit by directly writing the bit mask value. The last three statements in Figure 2 can, therefore, be replaced by the single statement in **Figure 3**.

```
wreg (rUSBIRQ, bmURESIRQ); // 1 clears an IRQ bit, 0 leaves it alone
```

Figure 3. The MAX3420E IRQ bits are cleared with a single register write.

### IEN Bits

Each of the 14 MAX3420E interrupts has an associated Interrupt ENable (IEN) bit. The IEN bit is ANDed with the IRQ flip-flop output to pass or block the request from propagating to the INT pin (Figure 1). The 14 IRQ flip-flops are gated, then OR'd together to form one internal interrupt request signal which passes to the Interrupt Pin Logic Block.

Note that the IRQ bit represents the pending status of an interrupt, regardless of the state of its IEN bit. This gives firmware the option of checking for a pending interrupt without it triggering the INT pin. If your code needs to check an IRQ register for "nothing pending," a simple solution is to read the IRQ and IEN registers, AND them together, and check the bits that now represent "pending and enabled IRQs." A zero value indicates that none of the enabled interrupts is pending.

### IE Bit

The SPI master enables or disables the INT pin with the IE bit. This is often called a global Interrupt Enable since it affects all interrupts. The INT pin goes inactive when IE = 0 regardless of the states of any of the IRQ or IEN bits.

### Interrupt Pin Logic

Two register bits, INTLEVEL (see discussion below) and POSINT, control the behavior of the INT pin. These configuration bits should be set before setting IE = 1.

#### Level Mode, INTLEVEL = 1

Some microcontroller systems use a level-sensitive active-low interrupt. In this configuration the MAX3420E drives the INT pin with an open-drain transistor to ground. Since the pin can only drive low, a pullup resistor is connected between the INT pin and the logic power supply. This mode allows INT pin outputs from multiple chips, each with open-drain outputs, to be wired together and pulled up with a single resistor. Because any of the chip outputs will pull the pin low, this logic is sometimes called 'wired-OR.' For systems of this type, set INTLEVEL = 1.

#### Edge Mode, INTLEVEL = 0 (default value)

The MAX3420E INT pin can also drive an edge-active interrupt system, where the microcontroller looks for 0-1 or 1-0 transition on its interrupt input pin. This is the default MAX3420E mode, with INTLEVEL = 0. The SPI master sets the edge polarity with a second bit, POSINT. When POSINT = 1, the MAX3420E delivers a 0-1 transition for pending interrupts. When POSINT = 0 (the default value), the MAX3420E delivers a 1-0 transition for pending interrupts.

Note the following from Figure 1:

1. If an IRQ bit is set and its associated IEN bit is clear, that IRQ does not affect the INT output pin. However, the interrupt is still pending. Its state can always be read in the IRQ bit, and it can be cleared by writing a 1 to the associated register bit.
2. A pending interrupt (IRQ bit is 1) whose IEN bit makes a 0-1 transition causes an interrupt
3. The INT pin can connect to a microcontroller's interrupt system. Alternatively, the microcontroller can poll the INT pin to determine if any MAX3420E interrupts are pending. The best mode to use for polling is level mode (INTLEVEL = 1), since in edge mode the INT pin can put out pulses too narrow for the microcontroller to see (see discussion below). Remember that the level mode requires a pullup resistor from the INT pin to  $V_L$ .

# INT Pin Waveforms

## Level Mode

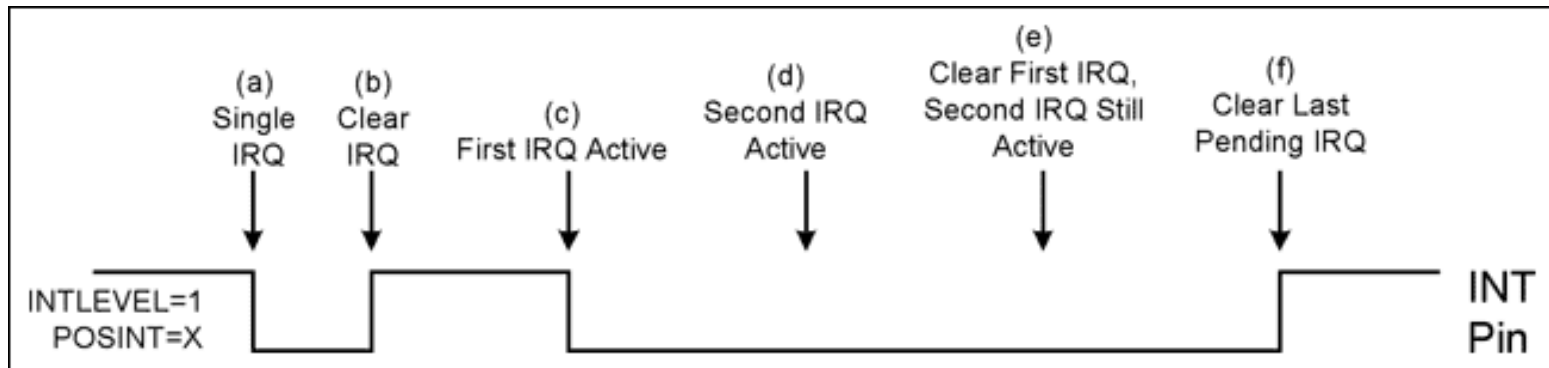


Figure 4. Behavior of the MAX3420E INT pin in LEVEL mode (INTLEVEL = 1).

**Figure 4** shows a MAX3420E INT pin waveform for level mode. The quiescent state of the INT pin is high (pulled up to  $V_L$ ). Assuming that the two interrupts in the figure have their IEN bits set to 1 and that the global IE bit is set to one, the following events occur. (The lettered items below correspond to the lettered events in Figure 4.)

1. An interrupt request arrives, causing the MAX3420E INT pin to drive low.

---

**Note:** Even though the MAX3420E interrupt output pin is called the INT pin, it can sometimes have a negative polarity (such as in level mode).

---

2. The SPI master finishes servicing the interrupt and clears its IRQ bit by writing 1 to it. The INT pin returns to its high quiescent state. The interval between (a) and (b) is the time between when the interrupt asserts its IRQ bit and the SPI master clears the IRQ bit.
3. Another interrupt request arrives, driving the INT pin low.
4. While the first interrupt request is pending, a second interrupt request arrives. The INT level does not change because at least one interrupt is pending. (Actually, two are pending at this moment.)
5. The SPI master finishes servicing one of the pending interrupts and clears its IRQ bit by writing 1 to it. The INT pin remains low because one interrupt is still pending.
6. The SPI master finishes servicing the remaining interrupt request and clears its IRQ bit by writing a 1 to it. No interrupts are pending, so the INT pin returns to its quiescent high state.

---

**Note:** An interrupt is considered to be pending if its IRQ flip-flop (Figure 1) is set.

---

This logic lends itself well to polling the INT pin. If anything in the MAX3420E requires service and its interrupt is enabled, the INT pin is low. The INT pin stays low until the microcontroller clears the last pending IRQ bit.

## Edge Mode

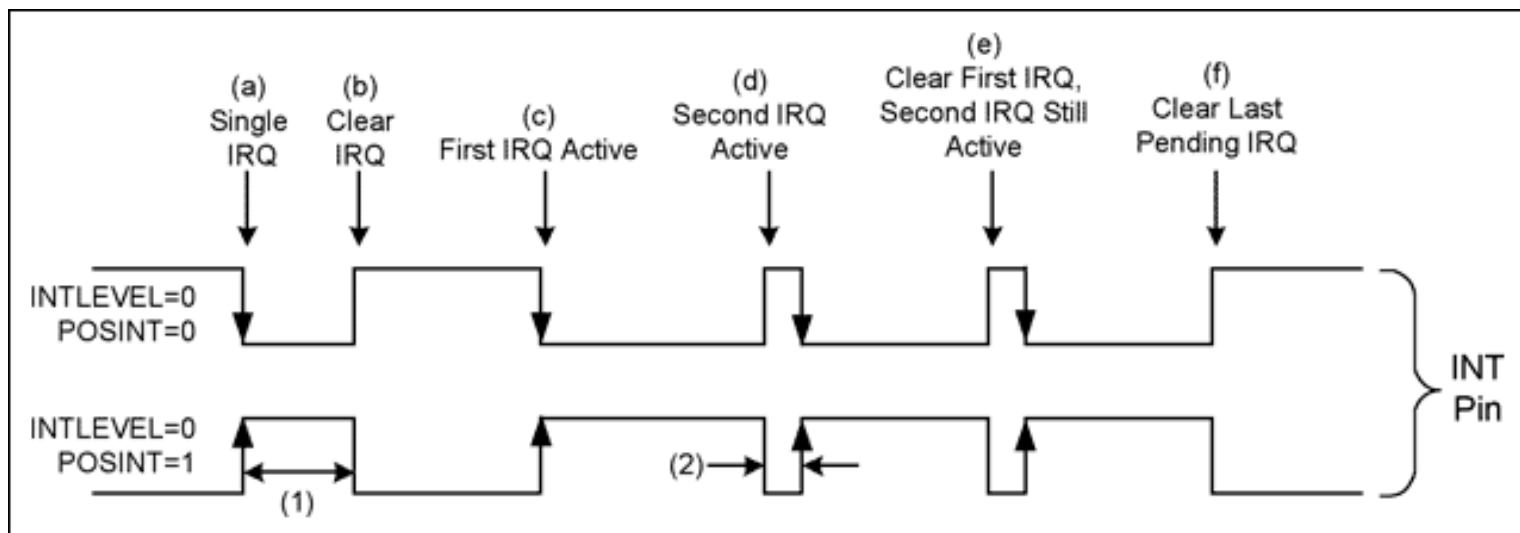


Figure 5. Behavior of the MAX3420E INT pin in EDGE mode ( $INTLEVEL = 0$ ). Interval (1) is the time it takes for the SPI master to clear the IRQ, and interval (2) is  $10.67\mu s$ .

**Figure 5** shows MAX3420E INT pin waveforms for edge mode operating in both polarities, controlled by the POSINT bit. The waveforms look similar to those in level mode, with two differences. The INT pin delivers an edge under two conditions:

1. An IRQ bit becomes active (its IRQ flip-flop makes a 0-1 transition).
2. The processor clears an IRQ bit (by writing 1 to it) and other IRQs are pending.

The second condition ensures that the processor gets an edge whenever service is still required.

Besides delivering edges, the INT pin also has an active and inactive state just as in level mode. The inactive state of the INT pin depends on the edge polarity set by the POSINT bit. In this regard the edge mode is similar to level mode, as you know if any interrupts are pending by looking at the state of the INT pin:

- In negative edge mode, the INT pin is high if no interrupts are pending; it is low if an interrupt is pending.
- In positive edge mode, the INT pin is low if no interrupts are pending; it is high if an interrupt is pending.

The following description refers to the INT pin state as active or inactive. Active means that at least one interrupt is pending; inactive means that no interrupts are pending. Again assuming that the interrupts are enabled, the following events occur. (The lettered items below correspond to the lettered events in Figure 5.)

1. An interrupt request arrives and the MAX3420E INT pin delivers an edge. The edge's polarity depends on the setting of the POSINT bit. Because the interrupt is still pending, the INT pin stays at its active state.
2. The SPI master finishes servicing the interrupt and clears its IRQ bit by writing 1 to it. The MAX3420E INT pin returns to its inactive state. The interval (1) in Figure between (a) and (b) is the time between when the interrupt asserts and the SPI master clears the IRQ bit.
3. Another interrupt request arrives and the MAX3420E INT pin delivers an edge and stays at its active state.
4. While the first interrupt request is pending, a second interrupt arrives. The MAX3420E INT pin must deliver another edge, so it pulses between its inactive and active states to deliver an edge of the proper polarity. The width of this pulse is fixed at  $10.67\mu s$  in the MAX3420E. Because interrupts are pending, the INT pin stays at the active state.
5. The SPI master finishes servicing one of the pending interrupts, and clears its IRQ bit by writing 1 to it. The INT pin delivers another edge as in step (d).
6. The SPI master finishes servicing the remaining interrupt request and clears its IRQ bit by writing a 1 to it. No interrupts are pending, so the INT pin returns to its inactive state.

# Interrupt Registers

**Table 1. The Shaded MAX3420E Registers Bits Control the Interrupt System**

Reg	Name	b7	b6	b5	b4	b3	b2	b1	b0	
0	R0	EP0FIFO	b7	b6	b5	b4	b3	b2	b1	b0
1	R1	EP1OUTFIFO	b7	b6	b5	b4	b3	b2	b1	b0
2	R2	EP2INFIFO	b7	b6	b5	b4	b3	b2	b1	b0
3	R3	EP3INFIFO	b7	b6	b5	b4	b3	b2	b1	b0
4	R4	SUDFIFO	b7	b6	b5	b4	b3	b2	b1	b0
5	R5	EPOBC	0	b6	b5	b4	b3	b2	b1	b0
6	R6	EP1OUTBC	0	b6	b5	b4	b3	b2	b1	b0
7	R7	EP2INBC	0	b6	b5	b4	b3	b2	b1	b0
8	R8	EP3INBC	0	b6	b5	b4	b3	b2	b1	b0
9	R9	EPSTALLS	0	ACKSTAT	STLSTAT	STLEP3IN	STLEP2IN	STLEP1OUT	STLEP0OUT	STLEP0IN
A	R10	CLRTOGS	EP3DISAB	EP2DISAB	EP1DISAB	CTGEP3IN	CTGEP2IN	CTGEP1OUT	0	0
B	R11	EPIRQ	0	0	SUDAVIRQ	IN3BAVIRQ	IN2BAVIRQ	OUT1DAVIRQ	OUT0DAVIRQ	INOBAVIRQ
C	R12	EPIEN	0	0	SUDAVIE	IN3BAVIE	IN2BAVIE	OUT1DAVIE	OUT0DAVIE	INOBAVIE
D	R13	USBIRQ	URES DNIRQ	VBUSIRQ	NOVBUSIRQ	SUSPIRQ	URESIRQ	BUSACTIRQ	RWUDNIRQ	OSCOKIRQ
E	R14	USBIEN	URES DNIE	VBUSIE	NOVBUSIE	SUSPIE	URESIE	BUSACTIE	RWUDNIE	OSCOKIE
F	R15	USBCTL	HOSCSTEN	VBGATE	CHIPRES	PWRDOWN	CONNECT	SIGRWU	0	0
10	R16	CPUCTL	0	0	0	0	0	0	0	IE
11	R17	PINCTL	EP3INAK	EP2INAK	EP0INAK	FDUPSPI	INTLEVEL	POSINT	GPXB	GPXA
12	R18	REVISION	0	0	0	0	0	0	0	1
13	R19	FNADDR	0	b6	b5	b4	b3	b2	b1	b0
14	R20	IOPINS	GPIN3	GPIN2	GPIN1	GPIN0	GPOUT3	GPOUT2	GPOUT1	GPOUT0

The MAX3420E has two sets of USB interrupts, controlled by the shaded registers in **Table 1**. The interrupt bits are grouped for EndPoint control in EPIRQ (R11) and EPIEN (R12), and USB control in USBIRQ (R13) and USBIEN (R14). The global IE bit is in the CPUCTL register.

**Table 2. The 14 MAX3420E Interrupt Sources**

Bit Name	Default	Location	Set By	Cleared By
INOBAVIRQ	1	EPIRQ.0	EP0 FIFO is ready for μP loading	Load the EPOBC register
OUT0DAVIRQ	0	EPIRQ.1	EP0-OUT FIFO has host data	Write EPIRQ = 0x02
OUT1DAVIRQ	0	EPIRQ.2	EP1-OUT FIFO has host data	Write EPIRQ = 0x04
IN2BAVIRQ	1	EPIRQ.3	EP2-IN FIFO is ready for μP loading	Load the EP2INBC register
IN3BAVIRQ	1	EPIRQ.4	EP3-IN FIFO is ready for μP loading	Load the EP3INBC register
SUDAVIRQ	0	EPIRQ.5	Setup Data is available in SUDFIFO	Write EPIRQ = 0x20
OSCOKIRQ	0	USBIRQ.0	MAX3420E Oscillator/PLL is stable	Write USBIRQ = 0x01
RWUDNIRQ	0	USBIRQ.1	SIE has finished signaling RWU	Write USBIRQ = 0x02
BUSACTIRQ	0	USBIRQ.2	Bus is active	Write USBIRQ = 0x04
URESIRQ	0	USBIRQ.3	Host started signaling bus reset	Write USBIRQ = 0x08
SUSPIRQ	0	USBIRQ.4	Host suspended the bus	Write USBIRQ = 0x10
NOVBUSIRQ	0	USBIRQ.5	V <sub>BUS</sub> comparator made 1-0 transition	Write USBIRQ = 0x20
VBUSIRQ	0	USBIRQ.6	V <sub>BUS</sub> comparator made 0-1 transition	Write USBIRQ = 0x40
URES DNIRQ	0	USBIRQ.7	Host finished signaling bus reset	Write USBIRQ = 0x80

**Table 2** shows the 14 interrupt control bits, when the MAX3420E internal logic sets them, and how the SPI master clears them.

## Interrupt Request Bits

### BAV Bits

Three Buffer Available (BAV) IRQ bits indicate that an IN endpoint FIFO is available for loading by the SPI master. The MAX3420E asserts these IRQ bits when the chip is reset, or when IN data is successfully sent from the endpoint buffer to the host. This IRQ tells the SPI master that the buffer is ready to accept new data.

**(SPI Master loads three bytes into EP3INFIFO, then writes EP3INBC = 3. MAX3420E sets EP3INBAV = 0.)**

Packet	Dir	F	Sync	IN	ADDR	ENDP	CRC5	EOP	Idle
7145	-->	S	00000001	0x96	0x03	0x3	0x13	3.00	4

Packet	Dir	F	Sync	DATA1	Data	CRC16	EOP	Idle
7146	<--	S	00000001	0xD2	00 00 08	0xF19F	2.80	6

Packet	Dir	F	Sync	ACK	EOP	Time
7147	-->	S	00000001	0x4B	3.00	31.991 ms

**Host acknowledges the IN data, MAX3420E sets EP3INBAV = 1.**

Figure 6. Bus trace of host sending an IN request to endpoint 3.

**Figure 6** is a bus trace of an IN transfer, where the host requests data from the MAX3420E. Before packet 7145 arrived, the SPI master loaded the bytes 00 00 08 into the Endpoint 3-IN FIFO (EP3INFIFO). Then the SPI master wrote the EP3INBC (Endpoint 3 IN Byte Count) register with the value 3. Writing the byte count register accomplishes three things:

1. It tells the MAX3420E how many bytes to send when the IN request arrives.
2. It arms the endpoint to transfer data (instead of NAKing).
3. It clears the EP3INBAV IRQ bit.

The MAX3420E answers the next IN packet addressed to endpoint 3 with the data packet 7146. The host acknowledges error-free receipt of the data by sending ACKnowledge (ACK) packet 7147. When the MAX3420E detects the host ACK packet, it sets the EP3INBAV interrupt request bit, informing the SPI master that the endpoint FIFO can be loaded with new data.

If the IN packet arrives before the SPI master has armed the endpoint, the MAX3420E answers with a NAK handshake (**Figure 7**). The NAK handshake tells the host to retry the IN request later.

Sync	IN	ADDR	ENDP	CRC5	EOP	Idle
00000001	0x96	0x03	0x3	0x08	2.80	4

Sync	NAK	EOP	Time
00000001	0x5A	3.00	12.700 μs

Figure 7. If the MAX3420E is not ready to send data, it sends a NAK.

If there is an error during the transmission of IN data to the host, the MAX3420E automatically resends the data (and the same data toggle DATA0/DATA1) when the host retries the IN request. Only after receiving the ACK handshake from the host does the MAX3420E assert the endpoint's BAV IRQ bit to indicate that the buffer is ready for new data.

**Important Note:** Like all MAX3420E IRQ bits, the three BAV IRQ bits can also be cleared by writing a 1. **NEVER DO THIS.** Instead, use the method outlined above: clear the BAV IRQ bits by writing the IN endpoint's byte count register. This is because the MAX3420E uses an IN endpoint's BAV interrupt request bit as a lockout mechanism. This mechanism ensures that the SPI master and the MAX3420E's serial interface engine (SIE) never try to use the endpoint buffer at the same time. For example, if you cleared the BAV bit and then loaded the byte count in two separate instructions, a packet transfer could be underway as you update the byte count register and the data would be corrupt.

#### BAV IRQ Default Values

The three BAV IRQ bits (shown as 1's in the Default column of Table 2) default to 1s. This means that the SPI master will read EPIRQ = 0x19 after power-on or a reset. If any of the corresponding IEN bits are set, the INT pin will indicate that interrupts are pending.

#### Double-Buffered Endpoint EP2-IN

The MAX3420E EP2-IN endpoint is double buffered. This means that it has two sets of 64-byte FIFOs and byte count registers. Double buffering improves transfer bandwidth because the SPI master does not need to wait for a packet to transfer to the

host before loading another one. With double buffering the SPI master can load one IN FIFO at the same time that the other one is transferring its IN data to the host. The two buffers 'ping-pong' when you load the EP2INBC register. This presents the other FIFO (the second set) and byte count register to the SPI master. This double-buffering action is transparent to the firmware.

The only observable impact of double buffering is at initialization time. The IN2BAVIRQ bit asserts at power-on or chip reset. Normally your initialization code will load data into the EP2IN FIFO and then load the EP2INBC register to arm the transfer and clear the IN2BAVIRQ bit. When you do this, you may be surprised to see that the MAX3420E immediately reasserts the IN2BAVIRQ bit. This indicates that the second buffer is available for the SPI master to load.

### DAV Interrupt Request Bits

OUT endpoints have Data Available (DAV) IRQ bits to indicate that new data has been received from the host. The MAX3420E automatically processes bus retries and asserts the interrupt request only when error-free data is received. When the SPI master gets a DAV interrupt request, it reads the endpoint's byte count register to determine the data payload size. Then the SPI master reads that number of bytes from the endpoint's OUTFIFO. The SPI master clears an OUTDAV IRQ bit in the normal way by writing 1 to it. This rearms the endpoint to receive the next OUT packet.

In **Figure 8** the host sends an OUT PID and four bytes of data, which the MAX3420E transfers to its EP1OUT FIFO. When the MAX3420E verifies the transfer as error-free, it updates its EP1OUTBC register to indicate four bytes, sends the ACK packet back to the host, and asserts the EP1OUTDAV IRQ, telling the SPI master that data is available for retrieval in the endpoint 1 FIFO.

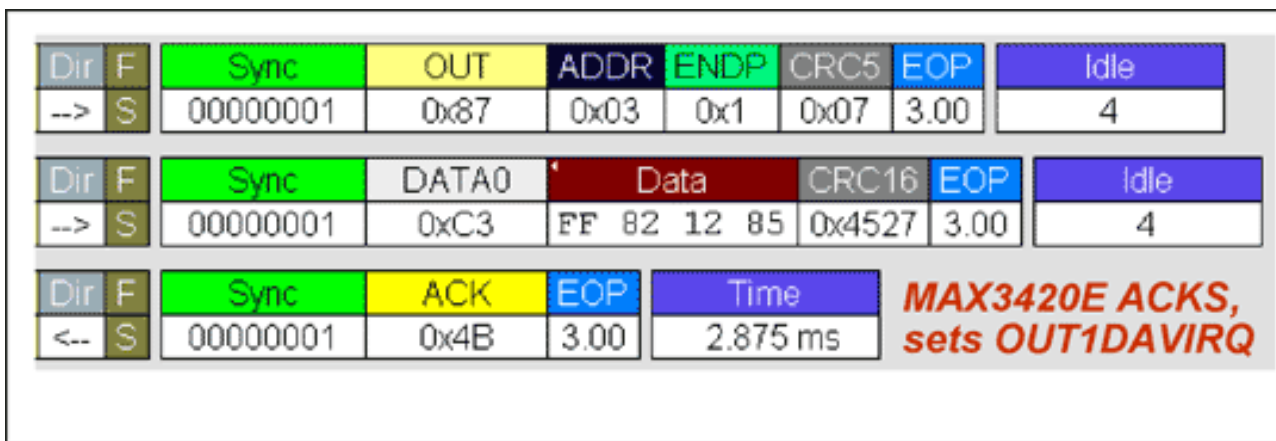


Figure 8. Bus trace of the host sending an OUT packet to endpoint 1.

### Double-Buffered Endpoint EP1-OUT

The MAX3420E EP1-OUT endpoint is double buffered, meaning that it has two sets of 64-byte FIFOs and byte count registers. The double buffering means that the OUT1DAVIRQ could immediately reassert after the SPI master clears it, if another host packet is waiting.

### SUDAV Interrupt Request Bit

When the host sends the MAX3420E a CONTROL transfer, the MAX3420E stores the eight SETUP bytes in an 8-byte FIFO that the SPI master reads from the SUDFIFO register. Because the peripheral always accepts host data in this buffer, the SUDAVIRQ acts just like an OUT endpoint FIFO, asserting its SUDAV IRQ when new data from the host is available. A SETUP packet always contains eight bytes, so no byte count register is required for SETUP data.

### OSCOK Interrupt Request Bit

When the MAX3420E powers up, comes out of chip reset, or exits its power-down state, the internal oscillator and PLL require time to start up and stabilize. The OSCillator OK (OSCOK) IRQ indicates that the MAX3420E is ready for operation.

```

#define      rUSBCTL      15
#define      bmCHIPRES    0x20
#define      rUSBIRQ      13
#define      bmOSCOIRQ    0x01

void Reset_MAX(void)
{
BYTE dum;
wreg(rUSBCTL,bmCHIPRES);      // chip reset
wreg(rUSBCTL,0x00);          // remove the reset
//
// Chip reset stops the oscillator. Wait for it to stabilize.
//
do
{
dum=rreg(rUSBIRQ);
dum &= bmOSCOIRQ;
}
while (dum==0);
}

```

Figure 9. Example code resets the MAX3420E and waits for OSCOK before exiting.

**Figure 9** is example code that resets the MAX3420E using its CHIPRES register bit. Because a chip reset stops the internal oscillator, after the code removes the reset signal by setting CHIPRES = 0 it should wait for the oscillator to stabilize before using the MAX3420E.

#### **RWUDN Interrupt Request Bit**

When in the suspended state, a USB peripheral may signal Remote WakeUp (RWU) to tell the host to resume bus activity. The USB specification defines a remote wakeup signal as a 1ms to 15ms K-state. The SPI master initiates RWU signaling by setting the Signal Remote WakeUp (SIGRWU) bit to 1.

When the SPI master sets the SIGRWU bit, the MAX3420E waits 5ms, drives the K-state for 10ms, and then asserts the Remote WakeUp Done interrupt request (RWUDNIRQ) bit. The 5ms delay ensures that another USB requirement is met: the bus must be idle (J-state) for at least 5ms before a peripheral initiates resume signaling.

```

SETBIT(rUSBCTL,bmSIGRWU)      // signal resume
while ((rreg(rUSBIRQ)&bmRWUDNIRQ)==0) ; // spin until resume signaling done
CLRBIT(rUSBCTL,bmSIGRWU)     // remove the RESUME signal
wreg(rUSBIRQ,bmRWUDNIRQ);    // clear the IRQ

```

Figure 10. Example code to signal remote wakeup.

**Figure 10** shows example code to signal remote wakeup. Note that the MAX3420E times the signal and asserts the IRQ upon completion. The MAX3420E does this for all timed USB events, delivering an interrupt when complete so the SPI master does not need to time the signaling intervals.

The code in Figure 10 sets the SIGRWU bit, then loops on the RWUDNIRQ bit to determine when the 10ms signal has elapsed. It then sets SIGRWU = 0 and clears the IRQ bit. Naturally, in a multitasking SPI master you would respond to the RWUDNIRQ interrupt request rather than wasting time directly checking the IRQ bit.

The SPI master should turn off the SIGRWU bit within 5ms of receiving the RWUDNIRQ interrupt. If it does not, the MAX3420E starts another 10ms K-state, and repeats this sequence (wait 5ms, 10ms, K-state) until SIGRWU = 0. Setting SIGRWU = 0 while RWU signaling is underway does not terminate the RWU signaling.

If the MAX3420E is in the power-down state (PWRDOWN = 1) when the SPI master sets SIGRWU = 1, the MAX3420E automatically restarts its oscillator and waits for it to stabilize before initiating the RWU signaling. The SPI master does not need to check the OSCOK IRQ in this case.

#### **BUSACT Interrupt Request Bit**

The MAX3420E sets the BUSACT IRQ bit when it detects the SYNC pattern at the beginning of a USB packet. A USB bus reset is not considered bus activity and, therefore, does not activate the BUSACK interrupt request.

#### **URES and URESDN Interrupt Request Bits**

A USB host resets a peripheral by driving at least 50ms of the Single-Ended-zero (SE0) state (both D+ and D- driven low). The

MAX3420E asserts the USB RESet IRQ (URESIRQ) after detecting 2.5 $\mu$ s of SE0. Then it asserts the USB RESet DoNe IRQ (URESJNIRO) when the host completes the reset signaling.

Because the SPI master needs to monitor USB bus reset events, the MAX3420E does not clear the URESIE, URESJNI, or IE interrupt enable bits during a bus reset. It does clear all the other interrupt enable bits in the EPIEN and USBIEN registers during a bus reset.

### ***SUSP Interrupt Request Bit***

When the MAX3420E detects 3ms of bus inactivity (a constant J-state), it asserts the SUSP interrupt request (SUSPIRO). If the peripheral using the MAX3420E is bus-powered, it must enter a low power state to minimize the current it draws from  $V_{BUS}$ . In this case the SPI master responds by shutting down power-consuming peripherals, and then putting the MAX3420E into a low-power mode by setting PWRDOWN = 1. This stops the MAX3420E oscillator and puts the MAX3420E into its lowest power state.

Two programming notes are worth mentioning:

- Clearing the SUSPIRO bit does not stop the interrupt from reasserting 3ms later. To avoid repeated SUSPEND interrupts while the bus is suspended, clear the SUSPEND IEN bit until the bus resumes signaling.
- The internal suspend timer logic is clocked by the MAX3420E's internal oscillator. Therefore, if you power the chip down (set PWRDOWN = 1) and then try to clear the SUSPIRO bit by writing a 1 to it, the MAX3420E will not clear the bit. The MAX3420E needs the now-stopped internal clock.

### ***VBUS and NOVBUS Interrupt Request Bits***

A self-powered peripheral can detect if it is plugged into USB and powered using these interrupts. The interrupts are activated by the internal  $V_{BUS}$  comparator, which compares the VBCOMP pin voltage with an internal reference voltage. They are edge sensitive, asserting when the  $V_{BUS}$  voltage is applied (VBUSIRQ) or removed (NOVBUSIRQ).

A bus-powered peripheral does not need to detect  $V_{BUS}$  since it is powered by  $V_{BUS}$ . This frees the VBCOMP pin for use as a general-purpose input. The VBCOMP pin has no internal pullup resistor in this application, so a pullup should be connected between the VBCOMP pin and  $V_L$ .

## **Programming Tips**

### ***Clearing the IEN bits***

#### **Chip Reset**

All IE bits are cleared during a chip reset. A chip reset is defined as one of the following:

1.  $V_L$  Power is applied to the MAX3420E (power-on reset).
2. The MAX3420E RES# pin is driven low.
3. The SPI master sets the bit CHIPRES = 1.

#### **Bus Reset**

All IE bits except three are cleared when the MAX3420E detects a USB bus reset (3ms of bus inactivity). The SPI master may need to accept bus reset-related interrupts in order to monitor the status of the bus reset signaling. Consequently, the following IE bits are unaffected by a bus reset:

1. URESIE
2. URESJNI
3. IE (global interrupt enable)

Because a USB bus reset clears most of the IE bits, the controlling firmware should be organized so the desired interrupts are re-enabled when the bus reset completes.

### ***Clearing BAV and DAV IRQ bits***

Remember that DAV IRQ bits are cleared in the normal way, by writing a one to the bit. BAV bits (for IN endpoints) are cleared differently, that is by writing the byte count register.

**More Information**

For technical support: [www.maxim-ic.com/support](http://www.maxim-ic.com/support)

For samples: [www.maxim-ic.com/samples](http://www.maxim-ic.com/samples)

Other questions and comments: [www.maxim-ic.com/contact](http://www.maxim-ic.com/contact)

---

**Automatic Updates**

Would you like to be automatically notified when new application notes are published in your areas of interest? [Sign up for EE-Mail™](#).

---

**Related Parts**

MAX3420E: [QuickView](#) -- [Full \(PDF\) Data Sheet](#) -- [Free Samples](#)

AN3661, AN 3661, APP3661, Appnote3661, Appnote 3661

Copyright © by Maxim Integrated Products

Additional legal notices: [www.maxim-ic.com/legal](http://www.maxim-ic.com/legal)