Keywords: MAX3421E,MAX3420E,ARM7,Keil,USB, Host, Peripheral

APPLICATION NOTE 3936

The Maxim USB Laboratory

Nov 02, 2006

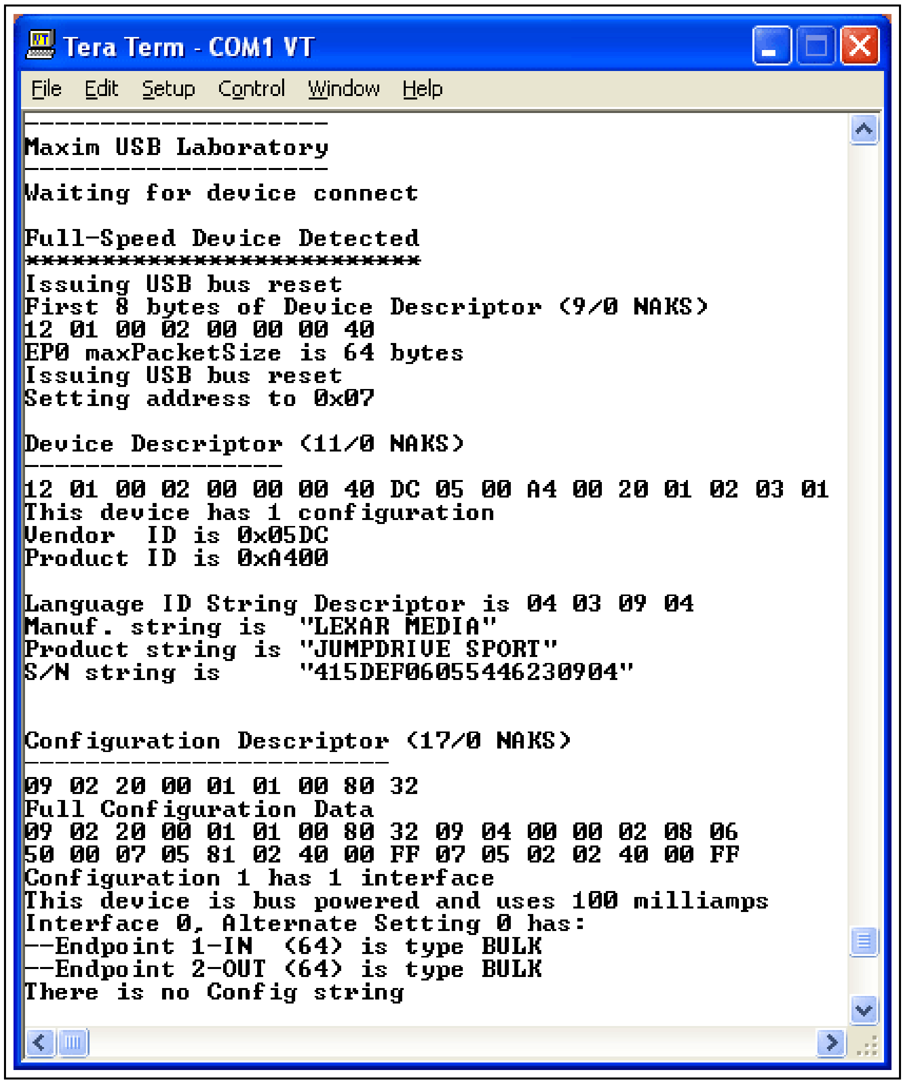
Abstract: The Maxim USB laboratory is a MAX3421E-/MAX3420E-based system using an ARM7™ processor and example software. This application note describes the system and the software that runs it. Both a USB host and peripheral are implemented in the same ARM® C code. This approach allows development and study of USB peripherals and embedded hosts with the advantage of having a reference device at the other end of the USB cable, all in the same C code.

This application note is a companion to application note 3937, "Setting up the Maxim USB laboratory." The Keil™ project containing the Maxim demo code is available for download.

Introduction

The Maxim USB laboratory is a combination of two circuit boards and a C program. With this system, you can:

- Study, operate, test, and modify a functional USB **device** based on an ARM7 microcontroller (µC) connected to a MAX3420E USB peripheral controller.
- Study, operate, test, and modify a functional USB **host** that uses the same ARM7 µC connected to a MAX3421E host controller. **Figure 1** shows data retrieved by the host from a USB memory stick.
- Connect the host to the peripheral with a USB cable and run both host and peripheral at the same time. When developing either USB host or USB peripheral code, it is very useful to have a reference design at the other end of the USB cable. By doing this, both sides of the USB cable, host and peripheral, are controllable and customizable since they are both implemented by the same C code.



```
---------------------
Maxim USB Laboratory
---------------------
Waiting for device connect

Full-Speed Device Detected
**************************
Issuing USB bus reset
First 8 bytes of Device Descriptor (9/0 NAKS)
12 01 00 02 00 00 00 40
EP0 maxPacketSize is 64 bytes
Issuing USB bus reset
Setting address to 0x07

Device Descriptor (11/0 NAKS)
------------------
12 01 00 02 00 00 00 40 DC 05 00 A4 00 20 01 02 03 01
This device has 1 configuration
Vendor  ID is 0x05DC
Product ID is 0xA400

Language ID String Descriptor is 04 03 09 04
Manuf. string is   "LEXAR MEDIA"
Product string is "JUMPDRIVE SPORT"
S/N string is      "415DEF06055446230904"


Configuration Descriptor (17/0 NAKS)
-------------------------
09 02 20 00 01 01 00 80 32
Full Configuration Data
09 02 20 00 01 01 00 80 32 09 04 00 00 02 08 06
50 00 07 05 81 02 40 00 FF 07 05 02 02 40 00 FF
Configuration 1 has 1 interface
This device is bus powered and uses 100 milliamps
Interface 0, Alternate Setting 0 has:
--Endpoint 1-IN  (64) is type BULK
--Endpoint 2-OUT (64) is type BULK
There is no Config string
```

Figure 1. The MAX3421 host/ARM retrieves enumeration data from any USB peripheral device, and reports results over a serial port connected to a PC running a terminal emulation program.

Other Reading

Programming guides, data sheets, and application notes for the MAX3420E and MAX3421E are available on the Maxim website:

MAX3420E
MAX3421E
Application note 3937, "Setting up the Maxim USB laboratory"

Hardware

The application C code runs on a two-board set:

- The Maxim MAX3421E evaluation (EV) kit, one board
- A Keil MCB2130, the second board

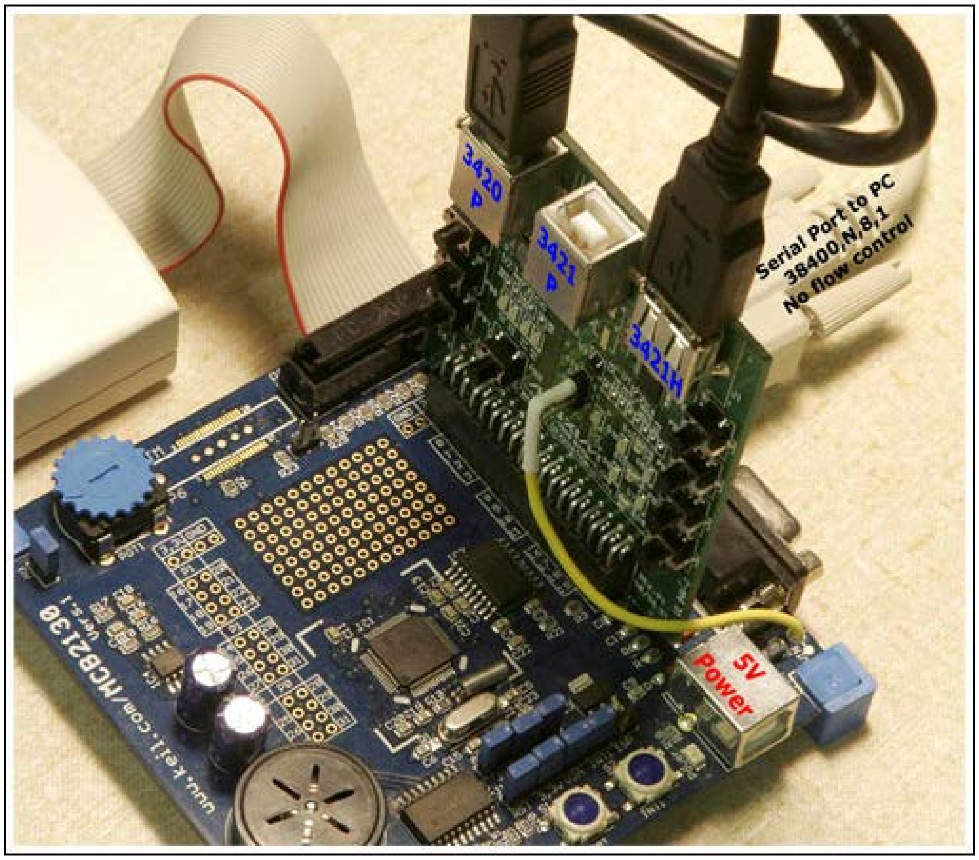


Figure 2. The Maxim EV kit plugs into a Keil MCB2130 board.

Figure 2 shows the board setup. The blue board is a Keil MCB2130 which contains an ARM7 chip, the Philips® LPC2138. This µC has two SPI™ hardware units which connect to the two Maxim USB controller chips.

The vertical board is the MAX3421E EVKIT-1. A MAX3420E peripheral controller connects to one of the ARM SPI ports, and is wired to the USB "B" connector (J5)labeled "3420 P" (P is for peripheral) in Figure 2. A MAX3421E host/peripheral controller connects to the other ARM SPI port, and is wired to the "3421P" (J2) and "3421H" (J1) connectors (H is for host). The code described in this application uses the MAX3421E as a host, so the middle USB connector (J2) on the EV-kit board is not used.

The host software uses one of the two serial ports (P1 on the MCB2130 board) to send USB descriptive information to a PC running a terminal emulation program. Terminal programs such as Tera Term Pro can emulate terminals (VT100) that recognize special "escape code" sequences sent by the program to clear the screen and home the cursor. The terminal program serial-port settings are 38400, N, 8, 1 with no flow control.

The beige box and ribbon cable in **Figure 2** is a Keil ULINK JTAG loader-debugger. This unit is supported by the Keil µVision®3 development environment. The MCB2130 board comes with an evaluation version of µVision3, a fully functional version of the Keil toolset. The evaluation version is limited in code size to 16kB.

File attachments to this application note include the full Keil project with all source files, plus the load module in .hex format. If you have the ULINK JTAG unit you can compile the code, load it, and debug it through the JTAG port. This is an excellent way to get up to speed on USB—modify working host and/or peripheral code to suit your purposes. If you do not have a ULINK box, you can still load and run the hex file using a free utility called Flash Magic, available at www.esacademy.com. Consult Maxim's application note 3937 for information about how to configure and use this utilty.

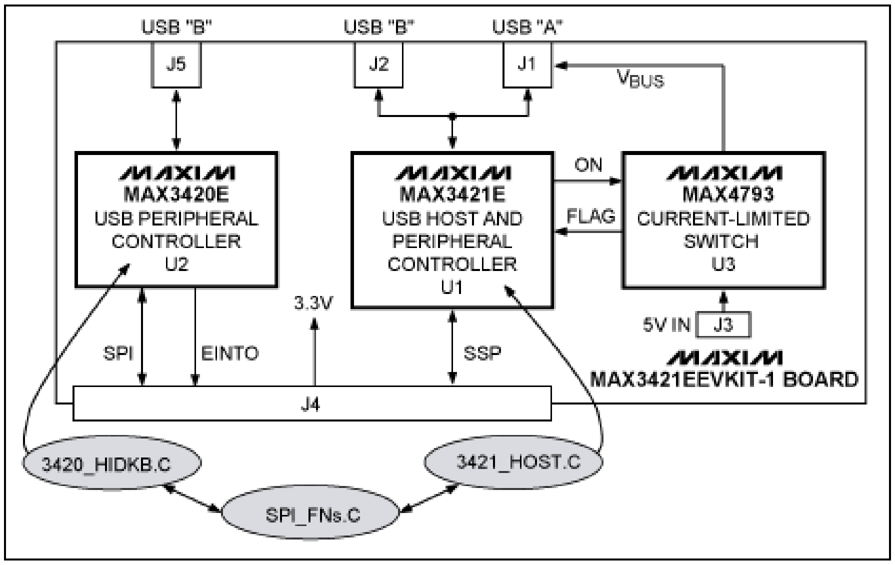


Figure 3. Block diagram of the MAX3421EEVKIT-1 board. The shaded ovals are the major software modules.

What the Code Does

Figure 3 shows a block diagram of the MAX3421EEVKIT-1 board and the software modules described in this application note. Both the MAX3420E and MAX3421E provide access to their register sets using the SPI bus. The LPC2138 contains two hardware SPI units:

- A dedicated SPI port (SPI)
- A general-purpose serial interface (SSP, Synchronous Serial Port)

The SSP is programmed to implement a second SPI port. These ports are programmed differently. The SSP, for example, has eight-deep transmit and receive FIFOs, while the SPI has only a one-deep read buffer. Both SPI ports are programmed to operate in 8-bit data mode for MAX3420E and MAX3421E compatibility. The software module **SPI_FNs.C** contains the register access functions used by both USB controllers.

The MAX3420E attaches to the SPI unit, so the functions to talk to this chip have a "P" prefix (e.g., Prreg and Pwreg) to indicate peripheral operation. The MAX3421E attaches to the SSP unit, so its access functions have an "H" prefix (e.g., Hwreg) to indicate host operations.

Peripheral Code

The C module **3420_HIDKB.C** implements a USB peripheral device which connects to a PC using EV kit connector J5. The MAX3420E INT output pin connects to the ARM7 EINT0 (External Interrupt Zero) pin. This interrupt asserts anytime the USB peripheral implemented by the MAX3420E requires service.

The application enumerates and operates as a standard USB HID (Human Interface Device). An advantage to conforming to a standard Windows® device class is that the device driver is built into Windows, so no special driver needs to be installed. Windows recognizes the device implemented by this application as a standard keyboard. Pressing any of the four pushbuttons attached to the MAX3420E GP-IN pins thus causes the "keyboard" to type a text string into any open window that accepts text, for example Notepad or Wordpad.

Warning: This application types into any application that accepts text, such as email, compiler editors, or a Word document. Be sure to have a safe application like Notepad open and active before pressing the button. The author can confirm by experience that rogue text typed into an open C source file will not compile properly.

Host Code

The **3421_Host_EVK.C** file contains host code that instructs the MAX3421E to perform enumeration steps similar to what a PC would do when a USB device is plugged into connector J4. The **main( )** function is in this module, as is the interrupt handler for the HID keyboard code. The EINT0 interrupt handler is simple:

```
                        // EINT0 Interrupt handler--MAX3420E INT pin
void INT3420 (void) __irq
{
service_irqs();         // Do the USB thing in 3420_HIDKB_INT_EVK.C module
EXTINT      = 1;        // Clear EINT0 interrupt flag (b0)
VICVectAddr = 0;        // Dummy write to indicate end of interrupt service
}
```

The background program **main( )** contains the following endless loop:

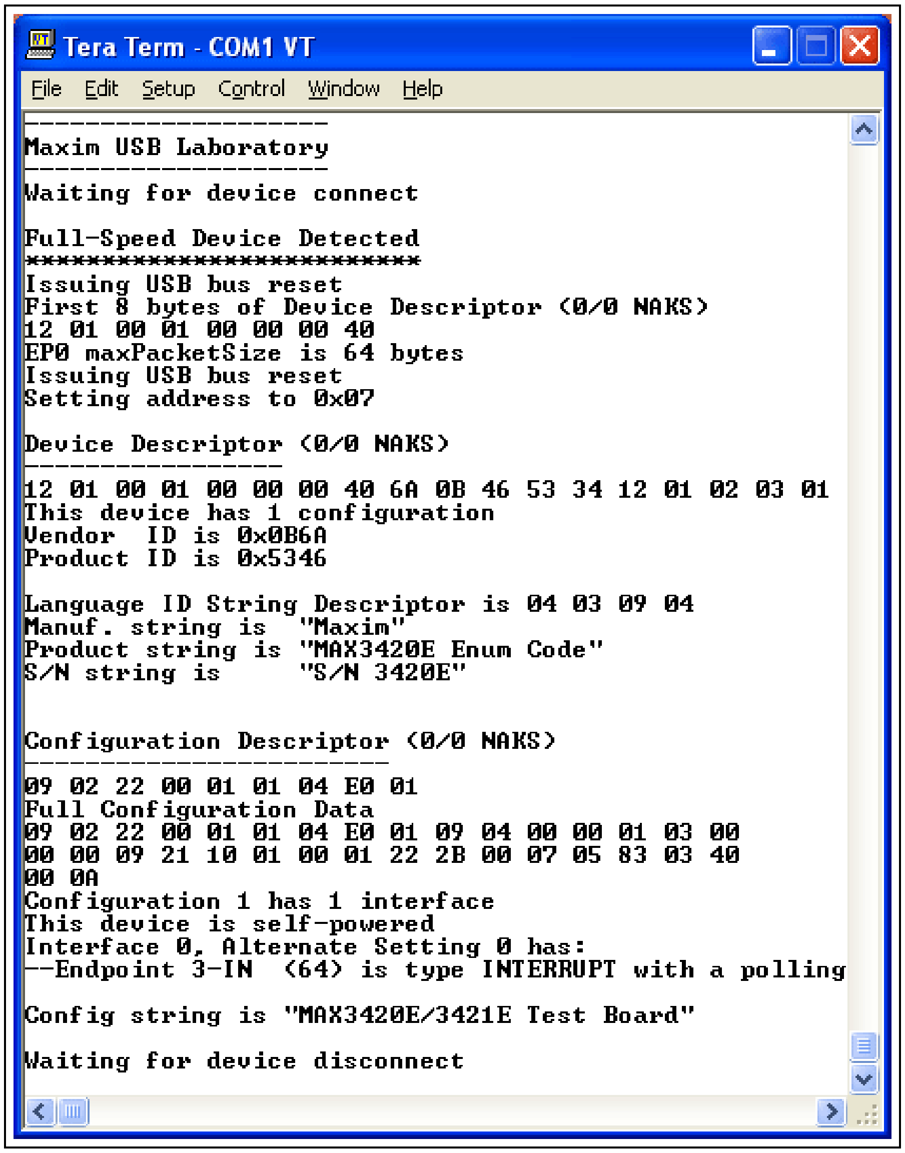
```
while(1)
  {
  detect_device();
  waitframes(200);      // Some devices require this
  enumerate_device();
  wait_for_disconnect();
  }
```

The **enumerate_device()** function does most of the work, sending USB requests to the attached device and reporting results over the serial port. Plug any USB device into EV kit J1, and this function sends enumeration requests to the device and reports results over the serial port (Figure 1).

Simultaneous Host/Peripheral Ooperation

Because the MAX3421EEVKIT-1 board contains both a USB peripheral and a USB host, and because they use separate SPI interfaces on the ARM7, the software can easily be structured so that both host and peripheral applications run simultaneously. The host application runs **main( )** to talk to the MAX3421E in the background, while the HID peripheral code activates the **service_irqs( )** function using a MAX3420E interrupt.

If you connect a USB cable from EV kit J5 to J1, the ARM processor can act as a USB host talking to a peripheral device that is...itself! **Figure 4** shows what the host code in 3421_Host.C reports when it interrogates the peripheral implemented in **3420_HIDKB.C**.



```
---------------------
Maxim USB Laboratory
---------------------
Waiting for device connect

Full-Speed Device Detected
**************************
Issuing USB bus reset
First 8 bytes of Device Descriptor (0/0 NAKS)
12 01 00 01 00 00 00 40
EP0 maxPacketSize is 64 bytes
Issuing USB bus reset
Setting address to 0x07

Device Descriptor (0/0 NAKS)
------------------
12 01 00 01 00 00 00 40 6A 0B 46 53 34 12 01 02 03 01
This device has 1 configuration
Vendor  ID is 0x0B6A
Product ID is 0x5346

Language ID String Descriptor is 04 03 09 04
Manuf. string is   "Maxim"
Product string is "MAX3420E Enum Code"
S/N string is      "S/N 3420E"


Configuration Descriptor (0/0 NAKS)
------------------------
09 02 22 00 01 01 04 E0 01
Full Configuration Data
09 02 22 00 01 01 04 E0 01 09 04 00 00 01 03 00
00 00 09 21 10 01 00 01 22 2B 00 07 05 83 03 40
00 0A
Configuration 1 has 1 interface
This device is self-powered
Interface 0, Alternate Setting 0 has:
--Endpoint 3-IN  (64) is type INTERRUPT with a polling

Config string is "MAX3420E/3421E Test Board"

Waiting for device disconnect
```

Figure 4. The demo code interrogating itself over USB.

Developing Host Code

Reading through the **enumerate_device( )** function can help you understand how a USB host operates, and how to command the MAX3421E to do the various host operations as it enumerates a USB device. While developing USB host code, it is sometimes difficult to diagnose a problem if it is not clear how the peripheral plugged into the host is responding. Having a reference peripheral over which you have total control (the **3420_HIDKB.C** module) makes it easy to know and vary device responses to test your host firmware.

Developing Peripheral Code

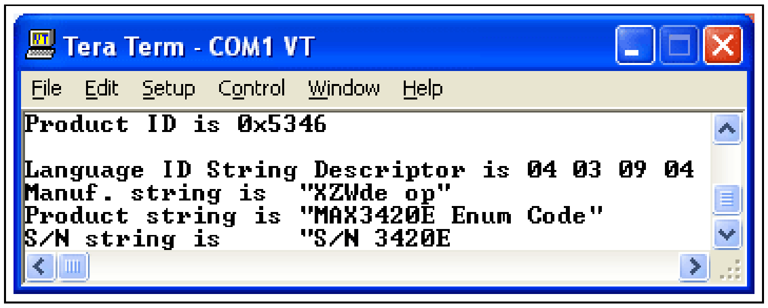
This host is an excellent interrogation tool for USB peripheral code development. The combination of the MAX3421E and the **3421_Host.C** module creates a simple and powerful USB packet generator and analyzer. You control what your peripheral device sees by writing C host code; the MAX3421E shows you how the peripheral responded.

An Example

Suppose you are writing peripheral code to handle USB strings. You code the manufacturer string as XYZ Widget Company in **hidkb_enum_tables.h** like this:

```
// STRING descriptor 1--Manufacturer ID
{
18,              // bLength
0x03,            // bDescriptorType = string
'X','Y','Z',' ','W','i','d','g','e','t',' ','C','o','m','p','a','n','y'
},
```

Compile this, load it into the ARM flash, and run it. The host program reports the following manufacturer string:



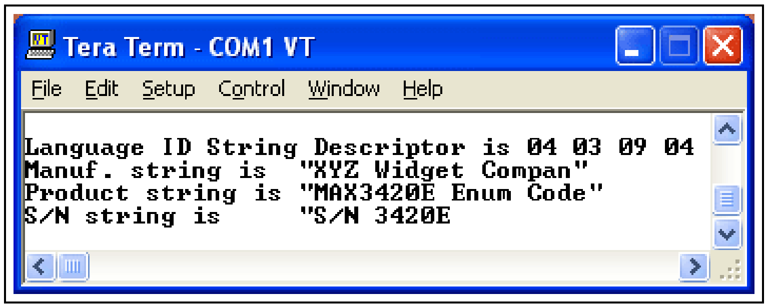
```
Product ID is 0x5346

Language ID String Descriptor is 04 03 09 04
Manuf. string is   "XZWde op"
Product string is "MAX3420E Enum Code"
S/N string is      "S/N 3420E
```

Figure 5. Not Good. Code example shows that the manufacturer's name is not want you intended to show. There is clearly a problem with the code.

XZWde op? What went wrong? Well, the USB spec calls for text strings to be expressed in Unicode, with two bytes per character. Go back to the source code for the peripheral, and change it to this:

```
// STRING descriptor 1--Manufacturer ID
        {
        35                        // bLength
        0x03,                     // bDescriptorType = string
        'X',0,'Y',0,'Z',0,' ',0,'W',0,'i',0,'d',0,'g',0,'e',0,'t',0,
        ' ',0,'C',0,'o',0,'m',0,'p',0,'a',0,'n',0,'y'  // Unicode!
        },
```

Compile, run, and you should see this:



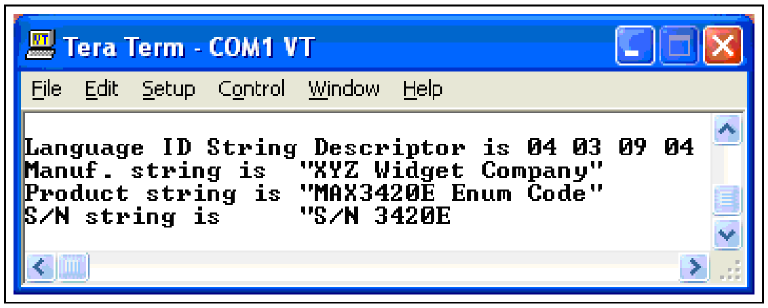
```
Language ID String Descriptor is 04 03 09 04
Manuf. string is   "XYZ Widget Compan"
Product string is "MAX3420E Enum Code"
S/N string is      "S/N 3420E
```

Figure 6. Better. Using the Unicode format, you see this result.

That is much better, but still not perfect. Where is the 'y' in Company? We carefully counted the 35 characters in the string and entered this as the first byte to indicate the string length. We missed something else: the length byte needs to include the first two bytes as well as the string. Make one final modification to the code:

```
// STRING descriptor 1--Manufacturer ID
        {
        37                 // bLength
        0x03,                     // bDescriptorType = string
        'X',0,'Y',0,'Z',0,' ',0,'W',0,'i',0,'d',0,'g',0,'e',0,'t',0,
        ' ',0,'C',0,'o',0,'m',0,'p',0,'a',0,'n',0,'y' // Unicode!
        },
```

The result should now look like this:



```
Language ID String Descriptor is 04 03 09 04
Manuf. string is   "XYZ Widget Company"
Product string is  "MAX3420E Enum Code"
S/N string is      "S/N 3420E
```

Figure 7. Best. The code now does what you thought it should.

By tweaking the peripheral code and exercising/analyzing it with the host controller and code, we see exactly what a PC would see since the reported results are for actual USB traffic sent from the MAX3420E to the MAX3421E. Best of all, we can do this on a controlled basis, without worrying about how a PC will respond to error-prone code in development.

Code Description

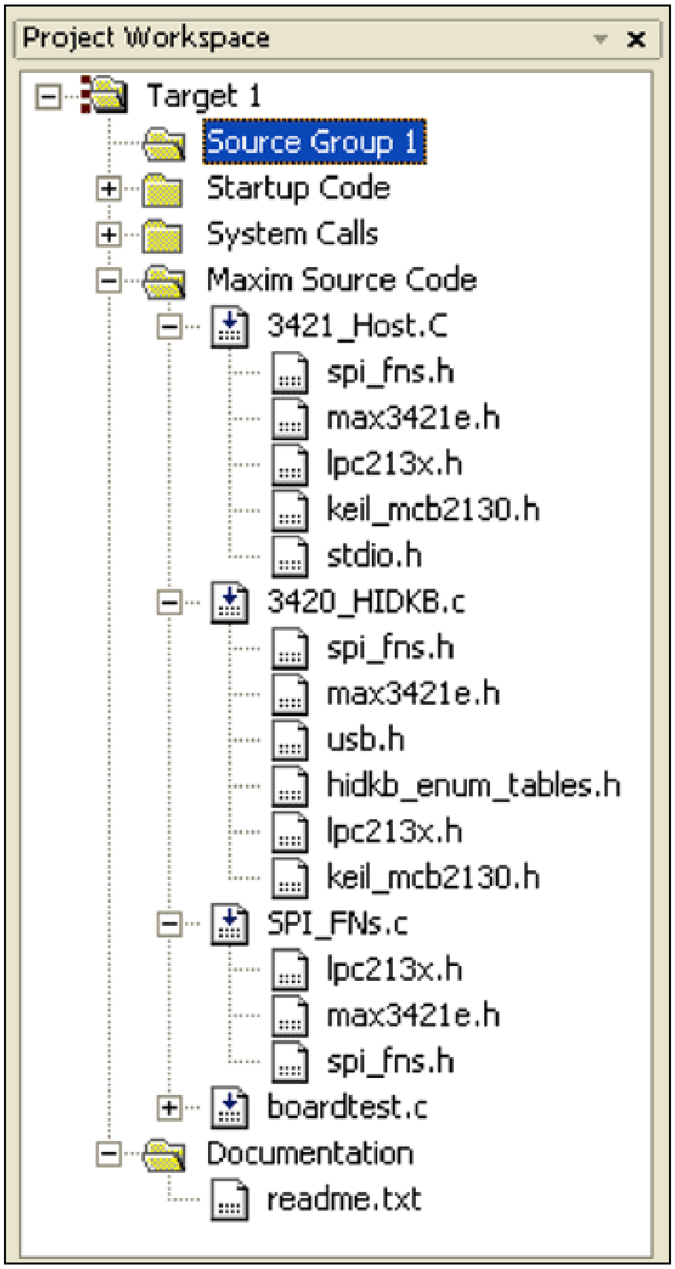


Figure 8. Structure of the Keil Project.

Figure 8 shows the three main modules, expanded to show their file dependencies. The unexpanded modules are housekeeping files required by the Keil environment.

SPI_FNs.C

This module contains utility functions called by the other two modules.

void init_PLL(void)

This function sets up the LPC2138 PLL and clock dividers. The crystal attached to the LPC2138 (F_{OSC}) is 12.000MHz. The CPU frequency is set to 48MHz according to the formula CCLK = $F_{OSC} \times M$, where M = 4. The LPC2138 uses an internal CCO (current-controlled oscillator) to multiply F to a higher internal

OSC

frequency. F_{CCO} = CCLK × 2 × P. F_{CCO} must be between 156MHz and 320MHz. For P = 2, F_{CCO} = 48MHz × 2 × 2 = 192MHz.

Note: The Keil file **startup.s** also contains startup code to set the PLL parameters. The **init_PLL()** function effectively overrides these settings.

Important Warning: Modify these settings at your own risk. Improper settings can force the LPC2138 outside of its spec limits and cause the Keil debugger to stop functioning. If this happens, the remedy is to use the Flash Magic utility to erase the LPC2138 flash memory (use the second serial port for this, connected to Keil board P2), correct the PLL settings in your code, then try again.

---

init_IO

This function initializes the LPC2138 I/O pins and SPI units (SPI and SSP):

- The SPI is clocked by CCLK/8, the maximum allowed. This SCLK is 48MHz/8 = 6MHz.
- The SSP is clocked by CCLK/(2 × VBDIV). VBDIF is set to 1, making the MAX3421E SCK signal equal to 48MHz/2 = 24MHz. The MAX3421E SCLK input can run as high as 26MHz.

About the SPI Interface
The SPI interface uses the following signals:

- MOSI Master Out, Slave In data
- MISO Master In, Slave Out data
- SCLK Serial Clock, supplied by the LPC2138
- SS# Slave Select, driven by the LPC2138

The LPC2138 SPI hardware manages the first three signals, but the SS# is directly set and cleared using a GP-OUT pin (P0.7).

About the SSP Interface
Although the SSP has a hardware SS# pin, it is more straightforward to use a GP-OUT pin for SS# (P0.20).

Hwreg
Hwritebytes
These functions write register and FIFO data to the MAX3421E using the LPC2138 SSP hardware. The H prefix indicates host operation. **Hwreg** writes a single register value; **Hwritebites** writes multiple bytes into a MAX3421E FIFO.

Hrreg
Hreadbytes
These functions read register and FIFO data from the MAX3421E using the LPC2138 SSP hardware.

More About the SSP Interface
The SSP hardware has a coding issue that results from having data FIFOs in the transmit and receive paths. In an SPI operation every 8-bit transfer *out* is always accompanied by an 8-bit transfer *in*. The LPC2138 accesses the MAX3421E SPI port using the following steps:

1. Assert SS# (low)
2. Send a command byte consisting of a register number and a direction bit
3. Send/receive one or more data bytes
4. De-assert SS# (high)

SPI writes are straightforward—everything goes *out*, and the data in the *in* FIFO may be ignored. But reads are more complex, because the read FIFO contains at least one bad byte, the one that was automatically clocked *in* during step 2. There also can be stale *in* data resulting from a previous operation that wrote multiple bytes. The *in* FIFO cannot be inhibited or flushed by hardware. Consequently, the code that reads SPI data must first manually flush the FIFO by reading bytes from the SSPDR until a flag indicating "read FIFO not empty" de-asserts.

Pwreg
PwregAS
Pwritebytes
These functions write register and FIFO data to the MAX3420E using the LPC2138 SPI hardware. The second function writes a register identical to the first function, but it also sets the ACKSTAT bit in the SPI command byte. This is a shortcut for terminating a USB CONTROL transfer. Consult application note 3598, "MAX3420E Programming Guide" for details.

Prreg
PrregAS
Preadbytes
These functions read register and FIFO data from the MAX3420E using the LPC2138 SPI hardware. The second function reads a register identical to the first function, but it also sets the ACKSTAT bit in the SPI command byte.

readout

This function updates the 7-segment readout connected to the MAX3421E general-purpose output pins GPO[6:0]. It preserves the setting of GPO[7], which controls the V_{BUS} switch to the USB "A" Connector (EV kit J1).

3421_Host.C

This module implements the MAX3421E USB host that interrogates USB devices and reports enumeration data over a MCB22310 serial port. For reference, Appendix A shows a LeCroy (CATC) bus trace for the host enumerating the built-in device implemented in **3420_HIDKB.C**.

There are three function types in this module:
1. Initialization
2. Utility functions
3. High-level functions called in **main( )**

The following descriptions follow this order.

Reset_Host

The MAX3421E contains a power-on reset, so this operation is not strictly necessary. However as you develop code, it is a good idea to reset the chip at the beginning of each debug session since power was probably not cycled. This starts with a clean machine, with nothing carrying over from the previous debug session.

Resetting the MAX3421E stops the on-chip oscillator. After de-asserting reset, the function waits for the OSCOKIRQ bit (Oscillator OK Interrupt Request) to go valid before returning.

initialize_3420

This function is in the **3420_HIDKB.C** module.

initialize_ARM_Interrupts

This function sets up the ARM vectored interrupts as follows:

1. EINT0 is connected to the MAX3420E INT pin, using priority 0 (highest).
2. Timer0 is used to blink an activity light, using priority 1.
3. EINT2 is connected to the MAX3421E INT pin, and given priority 2. It is not used in this program; rather, it is provided for convenience.
4. Timer1 is used to check the send/stop pushbutton (used in **3420_HIDKB.C**), using priority 3.

service_irqs

This function is in the **3420_HIDKB.C** module. It does the HID keyboard emulation. The code that initializes the ARM interrupts only needs to install this address as an interrupt vector.

waitframes

A simple way for a USB host to measure time is to count 1ms frame markers. When the µC sets the register bit SOFKAENAB = 1, the MAX3421E automatically generates these frame markers, which consist of SOF packets in full-speed mode or "keep alive" pulses in low-speed mode. The FRAMIRQ bit then asserts every millisecond.

The USB spec mandates certain long delay times, for example after resetting a device to give it a "reset recovery time." Calling the **waitframes** function is an easy way to implement these relatively long delays.

detect_device

This function returns after it detects a USB device plugged into the USB-A connector J1.

wait_for_disconnect

This function returns after a device plugged into the USB-A connector disconnects.

Send_Packet

This function is called by all functions that send USB packets. **Figure 9** shows the bus trace for an IN transaction and the C statements that produce it.

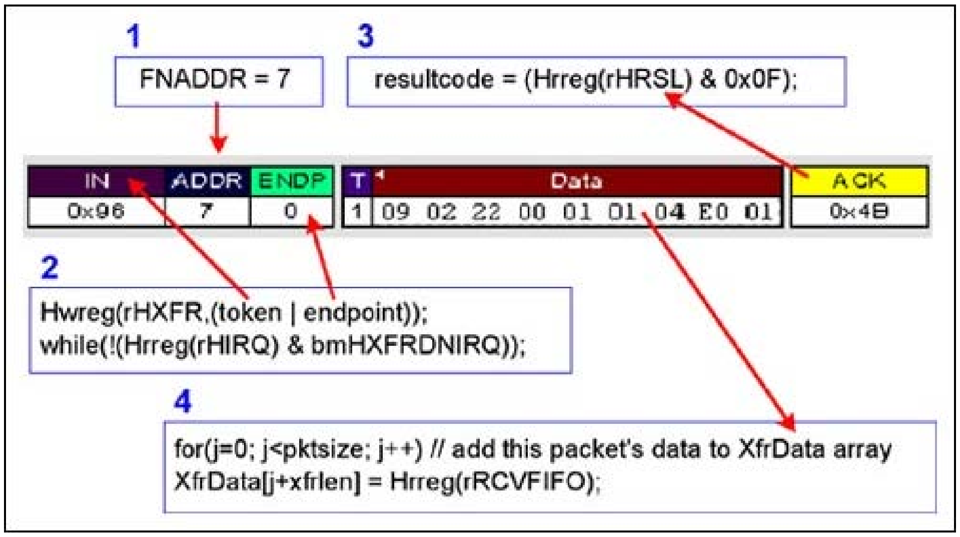

1
FNADDR = 7
3
resultcode = (Hrreg(rHRSL) & 0x0F);
IN
ADDR
ENDP
0x96
7
0
T
Data
1
09 02 22 00 01 01 04 E0 01
ACK
0x4B
2
Hwreg(rHXFR,(token | endpoint));
while(!(Hrreg(rHIRQ) & bmHXFRDNIRQ));
4
for(j=0; j<pktsize; j++) // add this packet's data to XfrData array
XfrData[j+xfrlen] = Hrreg(rRCVFIFO);

Figure 9. C function calls to the MAX3421E cause this USB bus activity.

1. The µC sets the function address in the FNADDR register. This does not need to be repeated for every packet, since the loaded value persists until the µC reloads the FNADDR register.
2. The µC launches the transfer by writing the HXFR register, specifying the USB token and endpoint. It

then waits for the HXFRDNIRQ (Host Transfer Done) IRQ to assert, signaling completion of the transfer.

3. The µC reads the HRSL (Host Result) register to determine the outcome of the transaction. Sixteen result codes indicate results such as:

   A. Success (ACK)
   B. Device busy (NAK)
   C. Unusual handshake (STALL)
   D. Device problem like timeout, babble, or bus error

4. The µC reads a byte count register, then unloads the RCVFIFO, and saves the data in a byte array.

This function handles NAK retries. It continues to relaunch the transfer until:

1. It gets the ACK handshake, or
2. It exceeds a preset limit set by the constant NAK_LIMIT.

If the device does not respond, the function retries the transfer up to the limit RETRY_LIMIT.

CTL_Write_ND

This function performs a CONTROL-Write transaction with no data stage (**_ND = no data**). This function is called only for the host Set_Address request. It makes two calls to **Send_Packet**: first for the SETUP packet, and second for the IN-Handshake packet.

IN_Transfer

This function makes as many calls to **Send_Packet** (specifying IN) as necessary to retrieve USB data records. The MAX3420E and MAX3421E endpoint FIFOs are 64 bytes long, so a long descriptor, for example, requires multiple IN requests with the packet data stitched together into a large array (XfrData[2000]).

CTL_Read

This function performs the three-stage CONTROL-Read USB transaction. It makes three calls:

1. To **Send_Packet** to issue the SETUP packet
2. To **IN_Transfer** to get the data into XfrData[]
3. To **Send_Packet** to issue the OUT-Handshake packet

enumerate_device

This function executes the 10 transfers shown in Appendix A. Because it calls the previous functions to do the low-level USB work, it is easy to follow the enumeration steps by studying this function.

print_error

This function checks the 4-bit HRSL value (item 3 in Figure 9) to print the outcome of the last host transfer. If the passed value is zero, it prints nothing. It returns the HRSL value so the calling function can easily check for an error, e.g.,

```
if(print_error(HR)) return;
```

3420_HIDKB.C

This module implements a USB peripheral using the MAX3420E. It conforms to the standard HID class to automatically type text as if it had originated from a standard keyboard. The entry point is the **service_irqs** function, which is invoked as an interrupt service routine in response to the LPC2138 EINT1 pin. EINT1 is connected to the MAX3420E INT output pin.

Reset_Peripheral

This function resets the MAX3420E in the same manner as **Reset_Host**.

initialize_3420

This function does three things:

1. Configures the MAX3420E SPI interface for full-duplex operation (separate MOSI and MISO pins)
2. Configures the MAX3420E INT pin as positive-edge active
3. Enables the MAX3420E interrupts used by the application

service_irqs

This function directly handles bus reset and device disconnect. It also calls **do_SETUP** to handle enumeration and **do_IN3** to do the keyboard typing.

Note: To keep code size down, this application does not handle USB suspend-resume. An example of how to do this is in the application note 3690, "USB Enumeration Code (and More) for the MAX3420E." The enumeration code in AN3690 was used as the basis for **3420_HIDKB.C**.

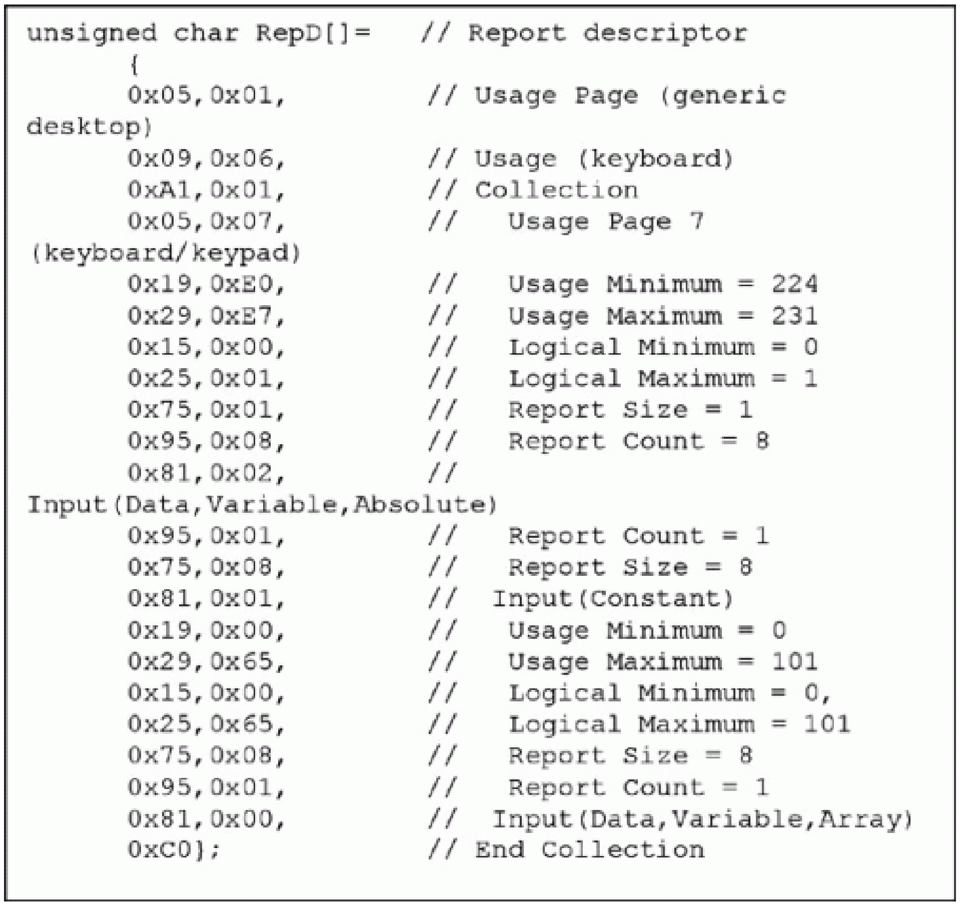
---

do_SETUP

This function handles device enumeration by inspecting various bytes in the 8-byte SETUP packet, then calling functions to handle the specific request. The code handles only the USB Standard request type, since the application does not implement USB Class or Vendor requests.

do_IN3

This function sends keystrokes according to the Message[] character array in **HIDKB_enum_tables.h**. The format of three bytes per character is defined in the HID Report Descriptor in the same include file, as shown in **Figure 10**.



```
unsigned char RepD[]=   // Report descriptor
       {
       0x05,0x01,         // Usage Page (generic
desktop)
       0x09,0x06,         // Usage (keyboard)
       0xA1,0x01,         // Collection
       0x05,0x07,         //   Usage Page 7
(keyboard/keypad)
       0x19,0xE0,         //   Usage Minimum = 224
       0x29,0xE7,         //   Usage Maximum = 231
       0x15,0x00,         //   Logical Minimum = 0
       0x25,0x01,         //   Logical Maximum = 1
       0x75,0x01,         //   Report Size = 1
       0x95,0x08,         //   Report Count = 8
       0x81,0x02,         //
Input(Data,Variable,Absolute)
       0x95,0x01,         //   Report Count = 1
       0x75,0x08,         //   Report Size = 8
       0x81,0x01,         //  Input(Constant)
       0x19,0x00,         //   Usage Minimum = 0
       0x29,0x65,         //   Usage Maximum = 101
       0x15,0x00,         //   Logical Minimum = 0,
       0x25,0x65,         //   Logical Maximum = 101
       0x75,0x08,         //   Report Size = 8
       0x95,0x01,         //   Report Count = 1
       0x81,0x00,         //  Input(Data,Variable,Array)
       0xC0};             // End Collection
```

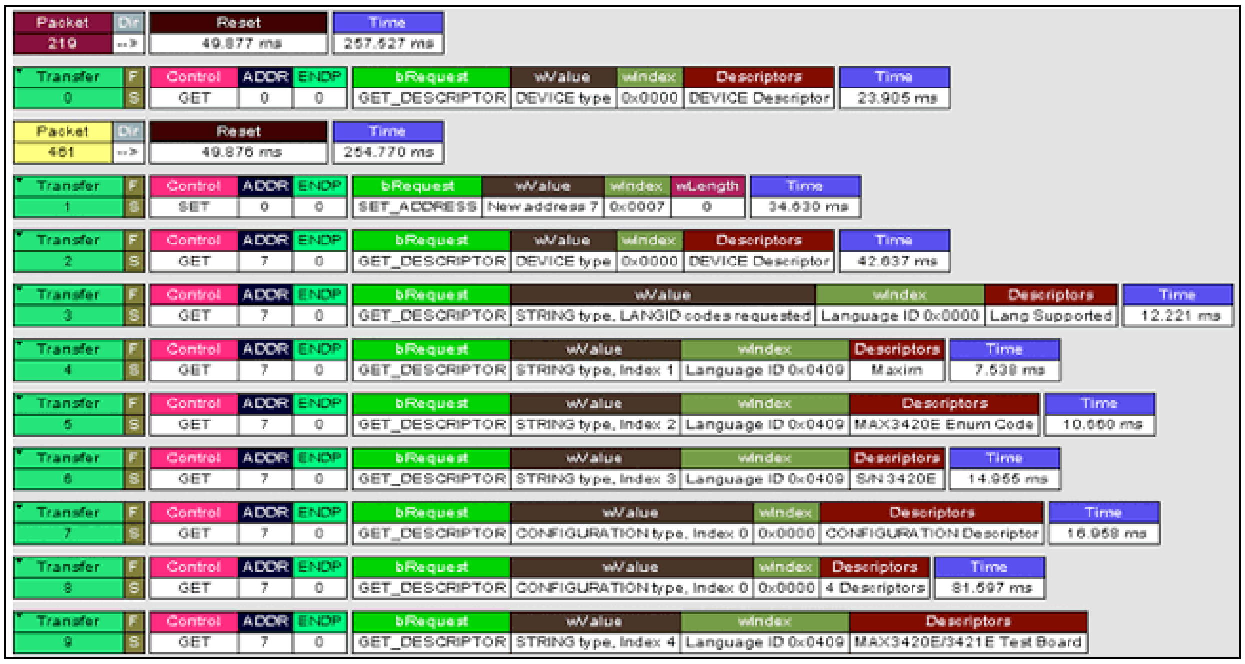
Figure 10. This Report Descriptor specifies the 3-byte data format for a keyboard keystroke.

Final Thoughts and a Disclaimer

The USB device that you plug into J1 could be any of the one billion or so devices available, so needless to say, there can be a wide range of responses to host enumeration requests. Any device that has been compliance-tested (evidenced by displaying the USB logo) should exhibit no errors when enumerated. The error checking in this program consists of inspecting the HRSL values after every host transaction. This error checking has not been extensively tested since many of the error conditions are difficult to force.

Appendix A

USB Bus Trace for 3420E_HIDKB.C Enumeration



```
Packet 219 | Dir --> | Reset 49.877 ms | Time 257.527 ms
Transfer 0 | F S | Control GET | ADDR 0 | ENDP 0 | bRequest GET_DESCRIPTOR | wValue DEVICE type | wIndex 0x0000 | Descriptors DEVICE Descriptor | Time 23.905 ms
Packet 461 | Dir --> | Reset 49.876 ms | Time 254.770 ms
Transfer 1 | F S | Control SET | ADDR 0 | ENDP 0 | bRequest SET_ADDRESS | wValue New address 7 | wIndex 0x0007 | wLength 0 | Time 34.630 ms
Transfer 2 | F S | Control GET | ADDR 7 | ENDP 0 | bRequest GET_DESCRIPTOR | wValue DEVICE type | wIndex 0x0000 | Descriptors DEVICE Descriptor | Time 42.637 ms
Transfer 3 | F S | Control GET | ADDR 7 | ENDP 0 | bRequest GET_DESCRIPTOR | wValue STRING type, LANGID codes requested | wIndex Language ID 0x0000 | Descriptors Lang Supported | Time 12.221 ms
Transfer 4 | F S | Control GET | ADDR 7 | ENDP 0 | bRequest GET_DESCRIPTOR | wValue STRING type, Index 1 | wIndex Language ID 0x0409 | Descriptors Maxim | Time 7.638 ms
Transfer 5 | F S | Control GET | ADDR 7 | ENDP 0 | bRequest GET_DESCRIPTOR | wValue STRING type, Index 2 | wIndex Language ID 0x0409 | Descriptors MAX3420E Enum Code | Time 10.660 ms
Transfer 6 | F S | Control GET | ADDR 7 | ENDP 0 | bRequest GET_DESCRIPTOR | wValue STRING type, Index 3 | wIndex Language ID 0x0409 | Descriptors S/N 3420E | Time 14.955 ms
Transfer 7 | F S | Control GET | ADDR 7 | ENDP 0 | bRequest GET_DESCRIPTOR | wValue CONFIGURATION type, Index 0 | wIndex 0x0000 | Descriptors CONFIGURATION Descriptor | Time 16.958 ms
Transfer 8 | F S | Control GET | ADDR 7 | ENDP 0 | bRequest GET_DESCRIPTOR | wValue CONFIGURATION type, Index 0 | wIndex 0x0000 | Descriptors 4 Descriptors | Time 81.597 ms
Transfer 9 | F S | Control GET | ADDR 7 | ENDP 0 | bRequest GET_DESCRIPTOR | wValue STRING type, Index 4 | wIndex Language ID 0x0409 | Descriptors MAX3420E/3421E Test Board
```

More detailed image (PDF, 234kB)

ARM is a registered trademark and registered service mark of ARM Limited.
ARM7 is a trademark of ARM Limited.
Keil is a registered trademark and registered service mark of ARM Limited.
Philips is a registered trademark of Koninklijke Philips Electronics N.V. Ltd.
Windows is a registered trademark and registered service mark of Microsoft Corporation.

Related Parts		
MAX3420E	USB Peripheral Controller with SPI Interface	Free Samples
MAX3420E	USB Peripheral Controller with SPI Interface	Free Samples
MAX3420E	USB Peripheral Controller with SPI Interface	Free Samples
MAX3421E	USB Peripheral/Host Controller with SPI Interface	Free Samples
MAX3421E	USB Peripheral/Host Controller with SPI Interface	Free Samples
MAX3421E	USB Peripheral/Host Controller with SPI Interface	Free Samples
MAX3421EVKIT-1	Evaluation Kit for the MAX3421E/MAX3420E	
MAX3421EVKIT-1	Evaluation Kit for the MAX3421E/MAX3420E	
MAX3421EVKIT-1	Evaluation Kit for the MAX3421E/MAX3420E	
MAX4793	200mA/250mA/300mA Current-Limit Switches	Free Samples
MAX4793	200mA/250mA/300mA Current-Limit Switches	Free Samples
MAX4793	200mA/250mA/300mA Current-Limit Switches	Free Samples