

EFM[®]32

... the world's most energy friendly microcontrollers

LUFA USB Stack CDC Demo

AN0861 - Application Note

Introduction

This application note introduces the LUFA USB stack running on the EFM32 platform. Software examples implementing LUFA USB Communication Device Class (CDC) on the EFM32GG-DK3750 and the EFM32GG-STK3700 are also included.

This application note includes:

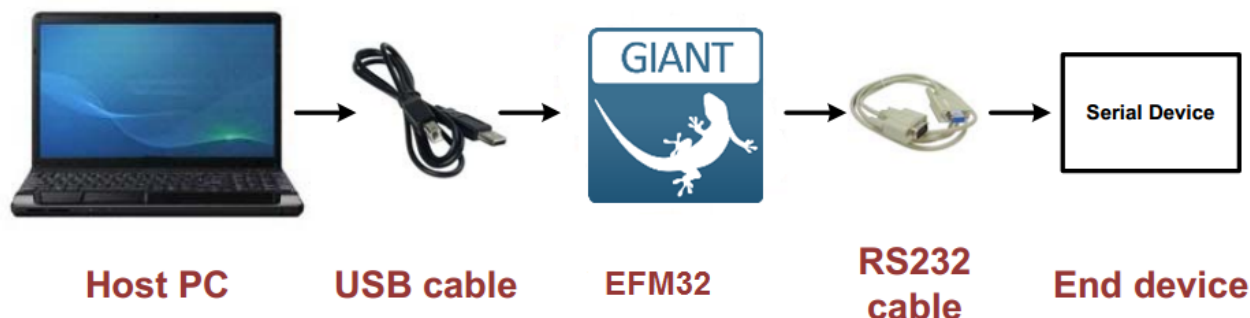
- This PDF document
- Source files (zip)
 - Example C-code
 - Multiple IDE projects



1 Introduction

USB revolutionized the PC peripheral space by making a very simple plug-and-play interface for users. As a result, many modern computers no longer support RS-232 serial COM ports, opting for the slimmer USB alternative. This can be an issue for the developer who needs a COM port for communication between a peripheral and host PC. A subset of the USB Communication Device Class (CDC) can be used to emulate a serial port providing a virtual COM port UART interface. This allows developers to use legacy applications with new products using the same COM port interface as before, with few hardware and software modifications.

Figure 1.1. USB CDC Virtual COM Port System



This application note describes the USB communications device class driver (or USB CDC) in detail and includes an implementation example for the Silicon Labs EFM32 MCU.

1.1 Assumptions

This document assumes the following:

- A working knowledge of the C programming language.
- Familiarity with the USB 2.0 specification and terms and abbreviations defined by the USB specification.
- Familiarity with Silicon Labs EFM32 development environment.

1.2 Features and Limitations

The CDC firmware implemented with this application note includes the following features:

- Emulates a serial COM port on a PC that supports the CDC Abstract Control Model (ACM).
- Provides an abstract communication interface for data transfers between the host and the device.
- Handles standard Chapter 9 USB device requests.
- Handles CDC-specific requests from USB host.
- Notifies the USB host of status using an interrupt endpoint.
- Provides data communication with the USB host using a bulk endpoint.
- The following baud rates are supported: 600, 1200, 2400, 4800, 9600, 14400, 19200, 38400, 57600, 76800, 115200 and 230400 bps.

The example does not implement the following:

- No CTS/RTS control is performed, so flow control must be set to **none** in the terminal program.
- RTS/DTR control is not implemented.

2 Relevant Documentation

EFM32 Application Notes are listed on the following website: <http://www.silabs.com/32bit-appnotes>

- **AN758 IMPLEMENTING USB COMMUNICATION DEVICE CLASS (CDC) ON SiM3U1XX MCUs** -- Provides an implementation example on porting LUFA USB CDC on SiM3U1xx MCUs.
- **AN0822 SIMPLICITY STUDIO USER'S GUIDE** -- Provides a description of the Simplicity Studio IDE features and environment.
- **AN0046 USB Hardware Design Guide** -- Provides recommendations on hardware design for implementing USB host and device applications using USB capable EFM32 micro-controllers.
- **AN0065 EFM32 as USB Device** -- Provides a description of the EFM32 USB Device stack.

3 USB Communication Device Class

The USB communications device class (CDC) is a composite USB device class, and the class may include more than one interface. The CDC is used primarily for modems, but also for ISDN, fax machines, and telephony applications for performing regular voice calls. The Abstract Control Model subclass of CDC and bridges the gap between legacy modem devices and USB devices, enabling the use of application programs designed for older modems.

3.1 Class Requests

The class requests and class notifications supported are listed in Table 3.1 (p. 4) .

Table 3.1. Abstract Control Model Requests

Request	Code	Description
SET_LINE_CODING	20h	Configures baud rate, stop-bits, parity, and number-of-character bits.
GET_LINE_CODING	21h	Requests current DTE rate, stop-bits, parity, and number-of-character bits.
SET_CONTROL_LINE_STATE	22h	RS232 signal used to tell the DCE device the DTE device is now present.

These class-specific requests are used by the host to configure and receive status info from the CDC device.

3.1.1 Set Line Coding

This request allows the host to specify typical asynchronous line-character formatting properties.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001b	SET_LINE_CODING	0	interface	size of structure	line coding structure

Table 3.2 (p. 4) defines the line coding properties.

Table 3.2. Line Coding Format

Offset	Field	Size	Value	Description
0	dwDTERate	4	Number	Data terminal rate, in bits per second.
4	bCharFormat	1	Number	0: 1 Stop bit 1: 1.5 Stop bits 2: 2 Stop bits
5	bParityType	1	Number	Parity: 0: None 1: Odd 2: Even 3: Mark 4: Space
6	bDataBits	1	Number	Data bits (5, 6, 7, 8 or 16).

3.1.2 Get Line Coding

This request allows the host to find out the currently configured line coding. Table 3.2 (p. 4) defines the line coding properties.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100001b	GET_LINE_CODING	0	interface	size of structure	line coding structure

3.1.3 Set Control Line State

This request generates RS-232/V.24 style control signals.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001b	SET_LINE_CONTROL_STATE	control signal bitmap	interface	0	none

The following table defines control signal bitmap.

Table 3.3. Control Signal Bitmap

Bit Position	Description
15:2	Reserved (Reset to zero).
1	Carrier control for half duplex modems. This signal corresponds to V.24 signal 105 and RS232 signal RTS. 0: Deactivate carrier. 1: Activate carrier. The device ignores the value of this bit when operating in full duplex mode.
0	Indicates to DCE if DTE is present or not. This signal corresponds to V.24 signal 108/2 and RS232 signal DTR. 0: DTE is not present. 1: DTE is present

3.2 Class Notifications

Table 3.4 (p. 5) shows the class notifications supported by the Abstract Control Model.

Table 3.4. Abstract Control Model Notifications

Bit Notification	Code	Description
SERIAL_STATE	20h	Returns the current state of the carrier detects, DSR, break, and ring signal.

Serial State

This notification sends an asynchronous message containing the current UART status.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100001b	SERIAL_STATE	0	interface	2	UART state bitmap

The data field for this notification is a bit-mapped value that contains the current state of detects transmission carrier, break, ring signal, and device overrun error. These signals are typically found on a UART and are used for communication status reporting. A state is considered enabled if its respective bit is set to 1.

Note: The firmware example included with this application does not currently support state change

Table 3.5. UART State Bitmap

Bit Position	Field	Description
15:7		Reserved (future use).
6	bOverRun	Received data has been discarded due to overrun in the device.
5	bParity	A parity error occurred.
4	bFraming	A framing error occurred.
3	bRingSignal	State of ring signal detection of the device.
2	bBreak	State of break detection mechanism of the device.
1	bTxCarrier	State of transmission carrier. This signal corresponds to V.24 signal 106 and RS232 signal DSR.
0	bRxCARRIER	State of receiver carrier detection mechanism of device. This signal corresponds to V.24 signal 109 and RS232 signal DCD

3.3 Endpoint Configuration

Table 3.6 (p. 6) illustrates the endpoint configuration for the Abstract Control Model.

Table 3.6. USB Endpoint Configuration

Endpoint	Direction	Type	Max Packet Size	Description
EP0	In/Out	Control	64	Standard requests, class requests.
EP1	In	Interrupt	16	State notification from device to host.
EP2	In	Bulk	64	Data transfer from device to host.
EP3	Out	Bulk	64	Data transfer from host to device.

The following figure shows a standard CDC communication flow.

Figure 3.1. USB CDC Communication Flow

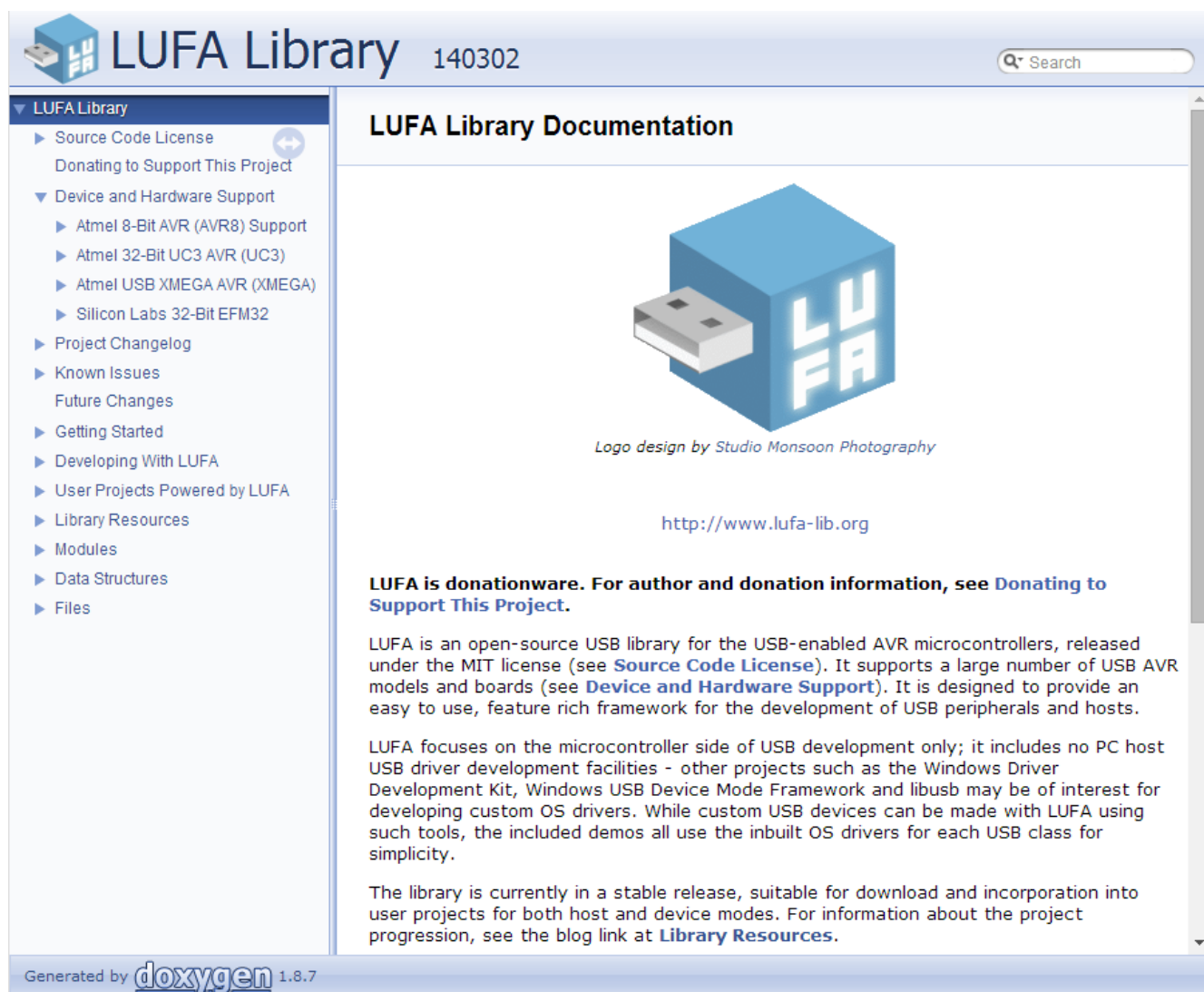
Item	Device	Endpoint	Interface	Status	Sp...	Payload
Enter text here	Ent...	Enter text h...	Enter te...	Enter ...	E.	Enter text here
Class request IN (0x21)	5	0		OK	FS	7 bytes (00 C2 01 00 00 00 08)
Class request IN (0x21)	5	0		OK	FS	7 bytes (00 C2 01 00 00 00 08)
Class request OUT (0x20)	5	0		OK	FS	7 bytes (00 E1 00 00 00 00 08)
Class request IN (0x21)	5	0		OK	FS	7 bytes (00 E1 00 00 00 00 08)
Class request OUT (0x22)	5	0		OK	FS	No data
Class request OUT (0x20)	5	0		OK	FS	7 bytes (00 E1 00 00 00 00 08)
Class request IN (0x21)	5	0		OK	FS	7 bytes (00 E1 00 00 00 00 08)
Class request OUT (0x22)	5	0		OK	FS	No data
IN transaction	5	1		ACK	FS	10 bytes (A1 20 00 00 00 00 02 00 00 00)
IN transaction	5	1		ACK	FS	10 bytes (A1 20 00 00 00 00 02 00 00 00)
APPEND - New elements appen...						
OUT transaction	5	3		ACK	FS	1 byte (61)
IN transaction	5	2		ACK	FS	1 byte (62)

4 LUFA USB Stack

The USB CDC firmware example is based on the LUFA open-source project. LUFA is an open-source complete USB stack released under the permissive MIT License. It includes support for many USB classes, both for USB Hosts and USB Devices. For USB Devices, the LUFA stack includes support for Audio Class, CDC Class, HID Class, Mass Storage Class, MIDI Class, and RNDIS Class. More information about the LUFA project can be found on the official website: <http://www.fourwalledcubicle.com/LUFA.php>


The USB CDC project contains a prebuilt LUFA USB stack documentation which is located at `.\LUFA\Documentation\html`. Double click on the `index.html`, and the documentation will show in your default browser.

Figure 4.1. USB LUFA Library Documentation



LUFA Library 140302

LUFA Library Documentation


Logo design by Studio Monsoon Photography

<http://www.lufa-lib.org>

LUFA is donationware. For author and donation information, see [Donating to Support This Project](#).

LUFA is an open-source USB library for the USB-enabled AVR microcontrollers, released under the MIT license (see [Source Code License](#)). It supports a large number of USB AVR models and boards (see [Device and Hardware Support](#)). It is designed to provide an easy to use, feature rich framework for the development of USB peripherals and hosts.

LUFA focuses on the microcontroller side of USB development only; it includes no PC host USB driver development facilities - other projects such as the Windows Driver Development Kit, Windows USB Device Mode Framework and libusb may be of interest for developing custom OS drivers. While custom USB devices can be made with LUFA using such tools, the included demos all use the inbuilt OS drivers for each USB class for simplicity.

The library is currently in a stable release, suitable for download and incorporation into user projects for both host and device modes. For information about the project progression, see the blog link at [Library Resources](#).

Generated by [doxygen](#) 1.8.7

5 EFM32 Software Examples

The software example included with this application note contains drivers and full LUFA USB stack for EFM32. USB CDC Demo is implemented in this software example. It supports two boards of EFM32, EFM32GG-STK3700 and EFM32GG-DK3750.

- The Communications Device Class(CDC) demonstration application gives a reference for implementing a CDC device acting as a Virtual COM Port(VCP). Source code is located at `.\Demos\Device\LowLevel\EFM32Demos\VCP`. The LUFA `VirtualSerial.inf` file located at this directory too. You need to supply the `.INF` file when running under Windows for the first time. This will enable Windows to use its inbuilt CDC drivers.
- This implementation supports two boards of EFM32, EFM32GG-STK3700 and EFM32GG-DK3750. Two set of project files for each board are provided in this implementation. The board's macro definition can be found in `.\LUFA\Common\BoardTypes.h`. Default setting is EFM32GG-DK3750.

```
#if !defined(__DOXYGEN__)
#define BOARD_ BOARD_DK3750
#if !defined(BOARD)
#define BOARD BOARD_DK3750
#endif
#endif
#endif
```

- Running on the EFM32GG-DK3750 board, a BSP initialize function (`BSP_Init(BSP_INIT_DEFAULT)`) is called in `VirtualSerial.c`. This is enabled by macro `BOARD_DK3750`.
- Each board has its own board specific driver code located at `.\LUFA\Drivers\Board\EFM32GG`. There are two directories, `DK3750` and `STK3700`. Button, LED and USART initialize functions are included in each directory. `DK3750\Serial.h` calls BSP functions to activate the RS232 port on the DK.
- For the EFM32GG-STK3700 board, there is no UART socket on the board. USART signals are connected to EXP Header. Connect those signals to other UART bridge devices, and you can get access it from PC host.

Table 5.1. USART signals on STK3700 Expansion Header

Peripheral	Peripheral pin	MCU pin	EXP Header pin number
USART/SPI	USART1_TX	PD0	4
	USART1_RX	PD1	6
	USART1_CLK	PD2	8
	USART1_CS	PD3	10

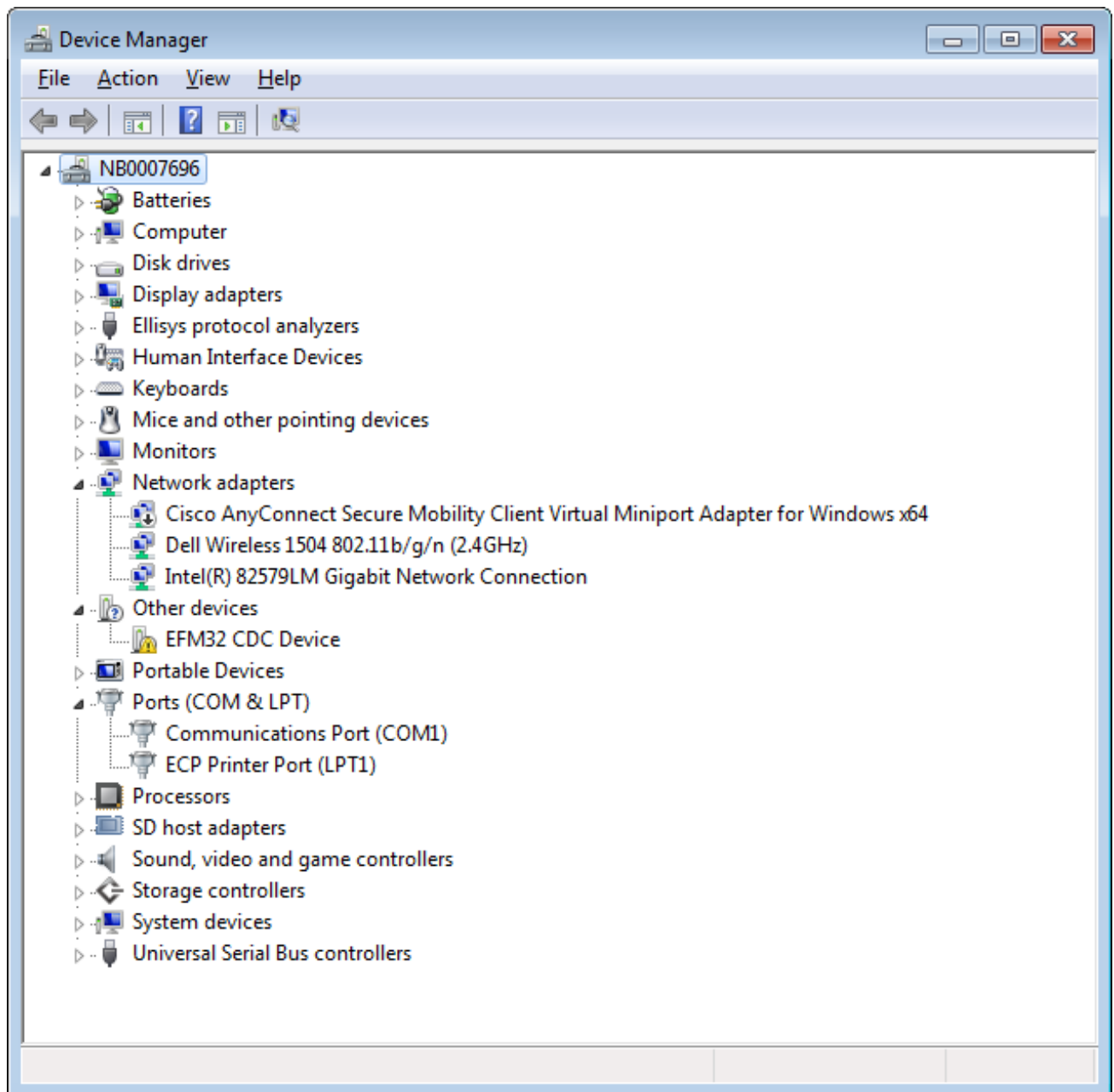
6 USB CDC Driver

The CDC class is implemented in all releases of Windows, and the operating system needs an INF file for the CDC driver. This INF file contains the Vendor ID and Product ID. If the VID/PID of the USB devices matches the INF file, Windows will load the driver described in the file. The **LUFA VirtualSerial.inf** file can be found in the **.\Demos\Device\LowLevel\EFM32Demos\VCP** directory.

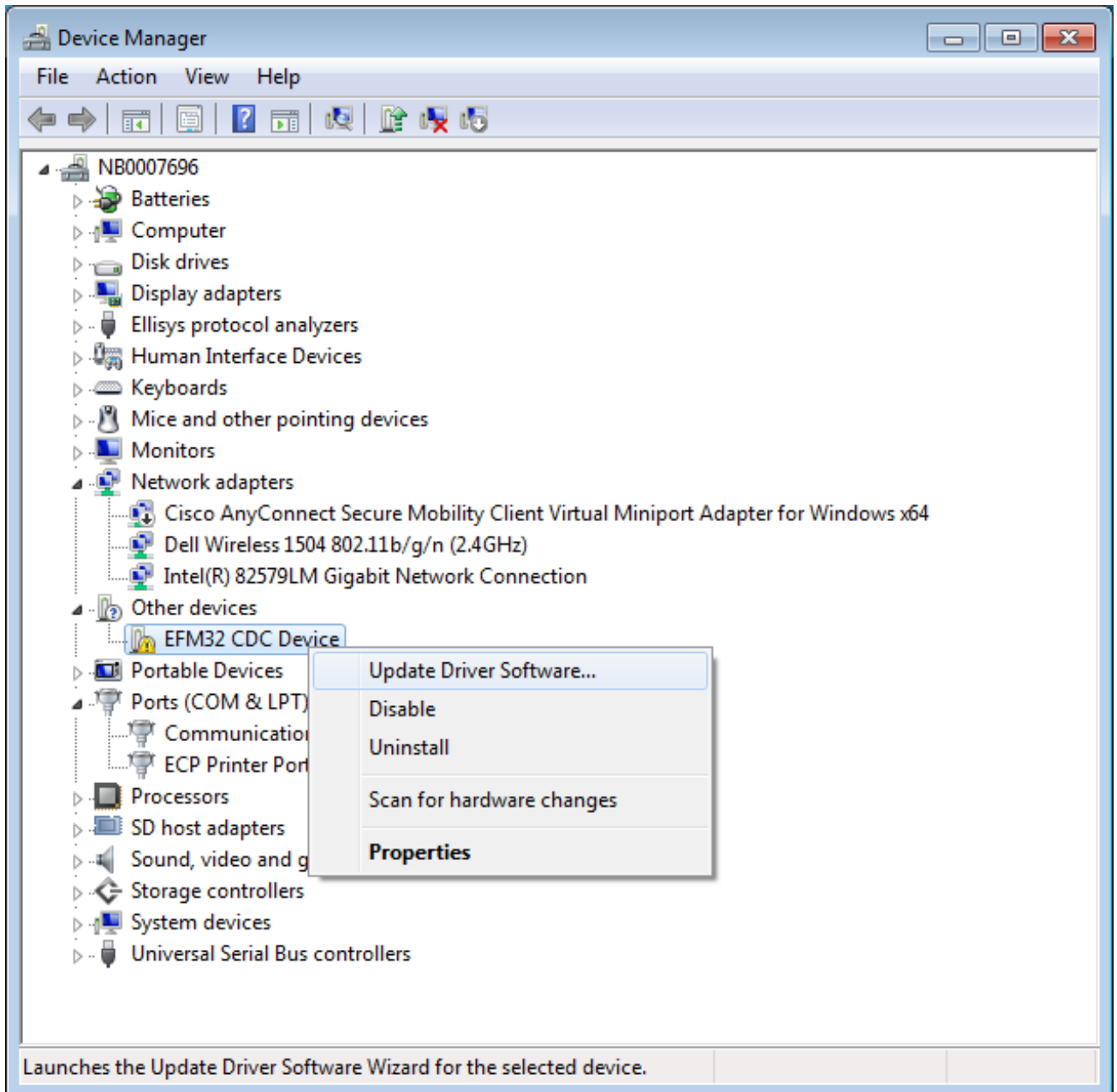
Installing the Driver

To install the driver on Windows 7:

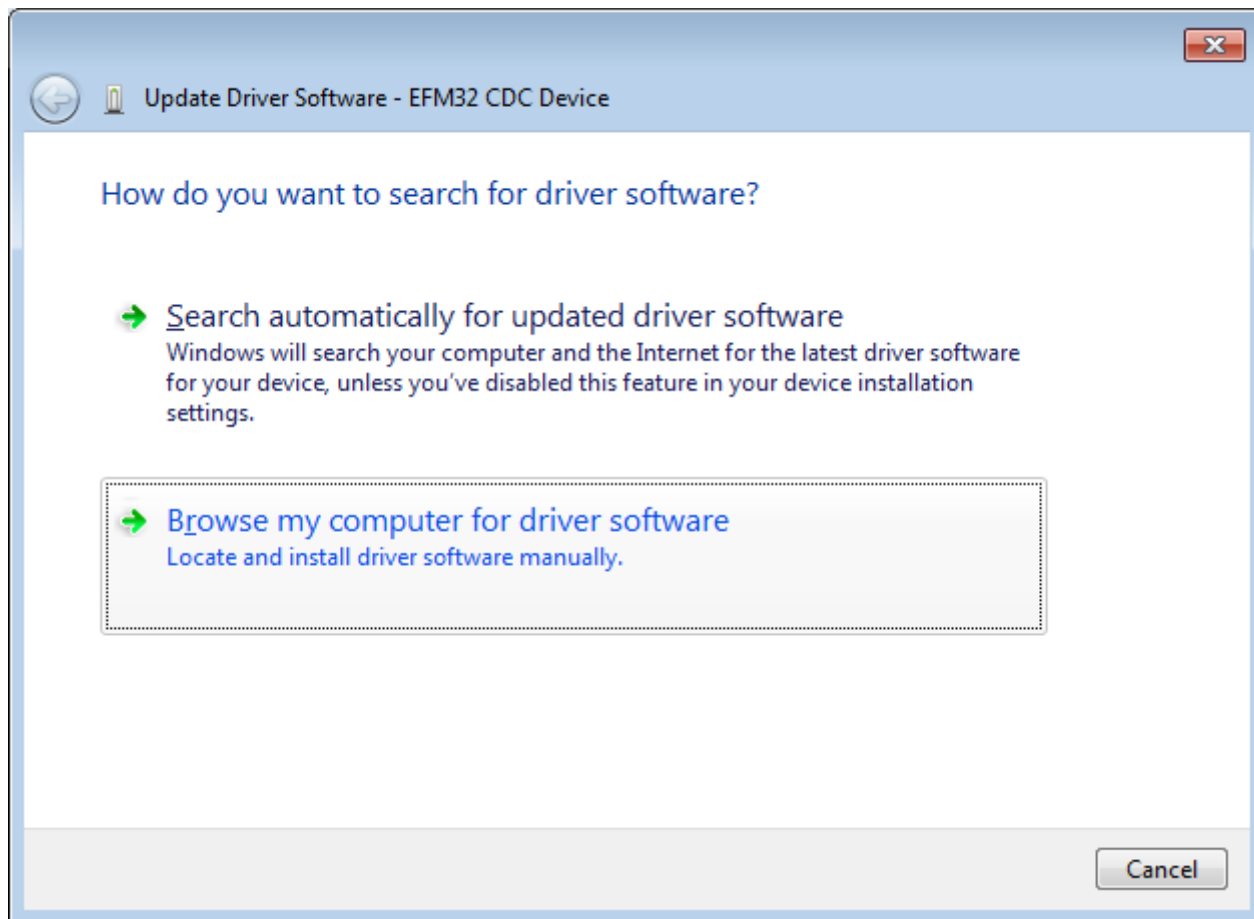
1. Build the project and download firmware to the EFM32 kit.
2. Connect the USB cable between the Device MCU plug-in board USB connector and the PC.
3. Open Device Manager. The device will appear under **Other devices** as the **EFM32 CDC Device**.



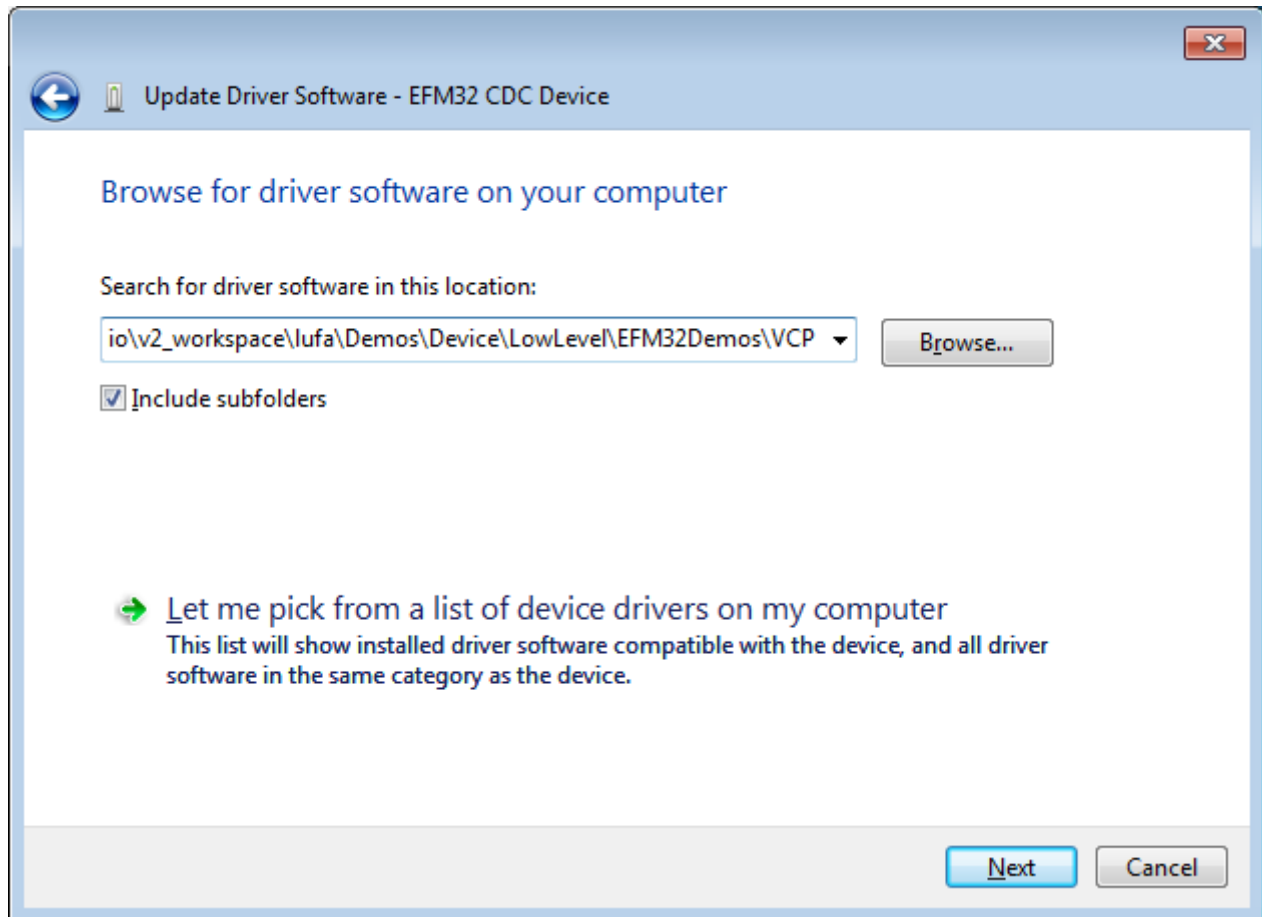
4. Right-click on the **EFM32 CDC Device** and select **Update Driver Software**.



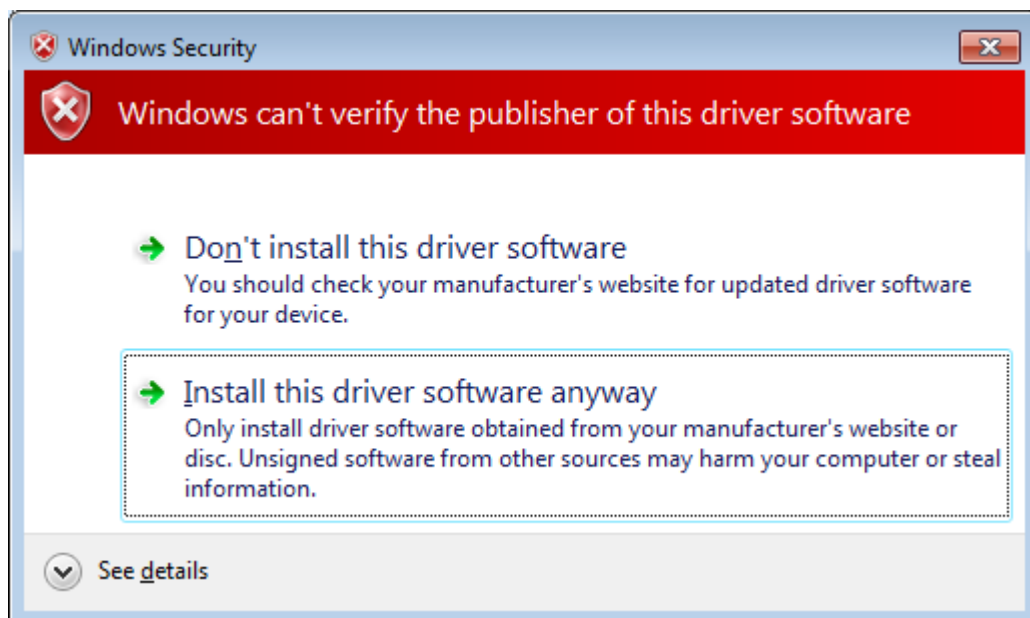
5. Select **Browse my computer for driver software**.



6. Enter the directory path of the **LUFA VirtualSerial.inf** file (`.\Demos\Device\LowLevel\EFM32Demos\VCP`). If the **Include subfolders** option is checked, entering the main `an0861_efm32_lufa_usb_cdc` directory in the workspace is sufficient.

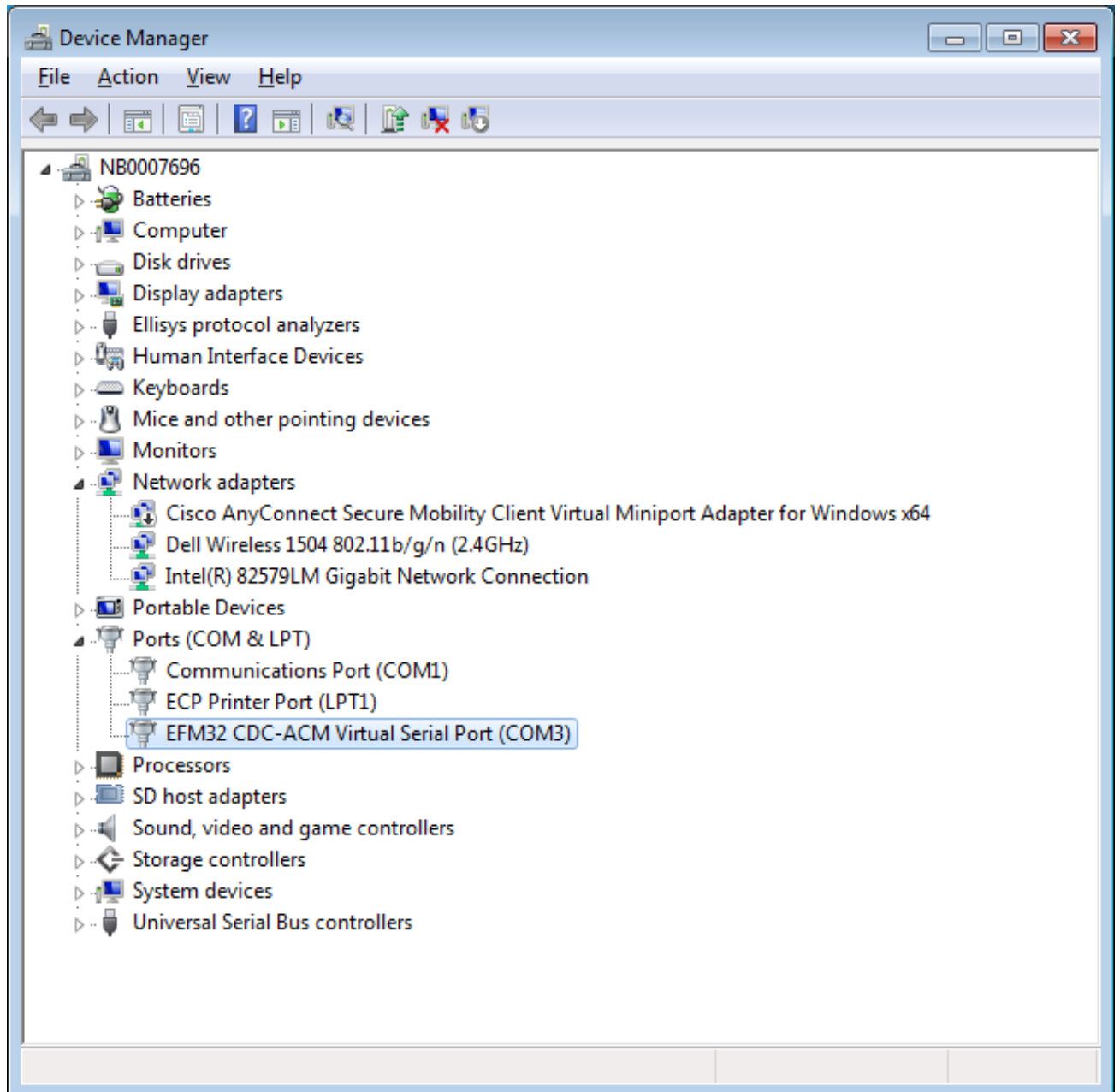


7. Windows will display a warning. Select **Install this driver software anyway**.



8. When the driver finishes installing, Windows will report the installation results.

9. Open Device Manager and observe the device. It will now appear under **Ports (COM & LPT)** with an assigned COM port number.



7 Revision History

7.1 Revision 1.00

2014-07-01

Initial revision.

A Disclaimer and Trademarks

A.1 Disclaimer

Silicon Laboratories intends to provide customers with the latest, accurate, and in-depth documentation of all peripherals and modules available for system and software implementers using or intending to use the Silicon Laboratories products. Characterization data, available modules and peripherals, memory sizes and memory addresses refer to each specific device, and "Typical" parameters provided can and do vary in different applications. Application examples described herein are for illustrative purposes only. Silicon Laboratories reserves the right to make changes without further notice and limitation to product information, specifications, and descriptions herein, and does not give warranties as to the accuracy or completeness of the included information. Silicon Laboratories shall have no liability for the consequences of use of the information supplied herein. This document does not imply or express copyright licenses granted hereunder to design or fabricate any integrated circuits. The products must not be used within any Life Support System without the specific written consent of Silicon Laboratories. A "Life Support System" is any product or system intended to support or sustain life and/or health, which, if it fails, can be reasonably expected to result in significant personal injury or death. Silicon Laboratories products are generally not intended for military applications. Silicon Laboratories products shall under no circumstances be used in weapons of mass destruction including (but not limited to) nuclear, biological or chemical weapons, or missiles capable of delivering such weapons.

A.2 Trademark Information

Silicon Laboratories Inc., Silicon Laboratories, Silicon Labs, SiLabs and the Silicon Labs logo, CMEMS[®], EFM, EFM32, EFR, Energy Micro, Energy Micro logo and combinations thereof, "the world's most energy friendly microcontrollers", Ember[®], EZLink[®], EZMac[®], EZRadio[®], EZRadioPRO[®], DSPLL[®], ISOmodem[®], Precision32[®], ProSLIC[®], SiPHY[®], USBXpress[®] and others are trademarks or registered trademarks of Silicon Laboratories Inc. ARM, CORTEX, Cortex-M3 and THUMB are trademarks or registered trademarks of ARM Holdings. Keil is a registered trademark of ARM Limited. All other products or brand names mentioned herein are trademarks of their respective holders.

B Contact Information

Silicon Laboratories Inc.

400 West Cesar Chavez

Austin, TX 78701

Please visit the Silicon Labs Technical Support web page:

<http://www.silabs.com/support/pages/contacttechnicalsupport.aspx>

and register to submit a technical support request.

Table of Contents

- 1. Introduction 2
 - 1.1. Assumptions 2
 - 1.2. Features and Limitations 2
- 2. Relevant Documentation 3
- 3. USB Communication Device Class 4
 - 3.1. Class Requests 4
 - 3.2. Class Notifications 5
 - 3.3. Endpoint Configuration 6
- 4. LUFA USB Stack 7
- 5. EFM32 Software Examples 8
- 6. USB CDC Driver 9
- 7. Revision History 14
 - 7.1. Revision 1.00 14
- A. Disclaimer and Trademarks 15
 - A.1. Disclaimer 15
 - A.2. Trademark Information 15
- B. Contact Information 16
 - B.1. 16

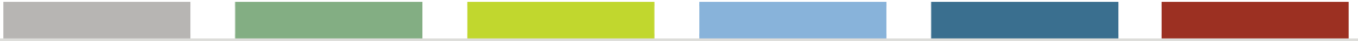
List of Figures

1.1. USB CDC Virtual COM Port System	2
3.1. USB CDC Communication Flow	6
4.1. USB LUFA Library Documentation	7

List of Tables

3.1. Abstract Control Model Requests	4
3.2. Line Coding Format	4
3.3. Control Signal Bitmap	5
3.4. Abstract Control Model Notifications	5
3.5. UART State Bitmap	6
3.6. USB Endpoint Configuration	6
5.1. USART signals on STK3700 Expansion Header	8

silabs.com



ZERO
ARM Cortex-M0+

TINY
ARM Cortex-M3

GECKO
ARM Cortex-M3

LEOPARD
ARM Cortex-M3

GIANT
ARM Cortex-M3

WONDER
ARM Cortex-M4