LUFA & Studio5 Beginner's Guide
Part 2: Modifying VirtualSerial for OLIMEX162 Board

Introduction

In Part 1 of this guide, the VirtualSerial Demo in LUFA was imported to a Studio5 project and modified so that it would compile. The demo was configured to run on the USBKEY board running an AVR at90usb1287 microcontroller. Part 2 will change the Studio5 project to run on an OLIMEX162 board running an AVR at90usb162 microcontroller.

LUFA (Lightweight USB Framework for AVRs), written and maintained by Dean Camera, is available at:
http://www.fourwalledcubicle.com/LUFA.php.

This guide uses Studio5 version 5.1.208 and LUFA-120219.

The source code used in Part 2 can be downloaded here:
http://www.avrfreaks.net/index.php?name=PNphpBB2&file=viewtopic&t=119003

Part 1: Compiling the VirtualSerial Demo can be found here:
http://www.avrfreaks.net/index.php?name=PNphpBB2&file=viewtopic&t=118597

Caveat Emptor

I am not an expert in AVR, Studio5, or LUFA. In fact, I'm a rank beginner in both Studio5 and LUFA. I hope this guide helps you get started with LUFA and speeds you on the way to building USB enabled apps.
No warranty expressed or implied. YMMV.

Before We Start

Some of the tables and figures in this guide show information taken from the LUFA documentation. (As mentioned in Part 1 of the guide, the documentation needs to be downloaded separately from the library.) When included in this guide, a path to that section of the documentation is shown above the item. An example is shown in Figure 1.

LUFA Library > Related Pages > Getting Started

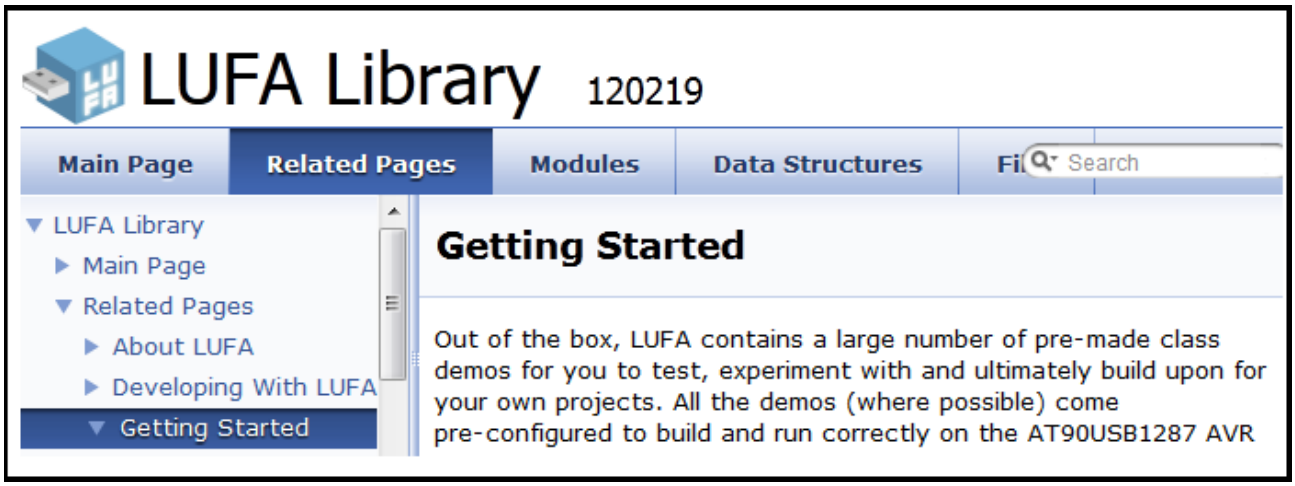

Figure 1: The Getting Started section of the LUFA Documentation, showing the left hand navigation pane.

Step 1: Select “Device: AT90USB162” in Project Properties

In Studio5’s Solution Explorer right click on VirtualSerial Project, the select ‘Properties’ from the popup menu. Studio5 will display the project properties. Select ‘Device’ from the left hand menu and then click ‘Change Device’ and select the AT90USB162.

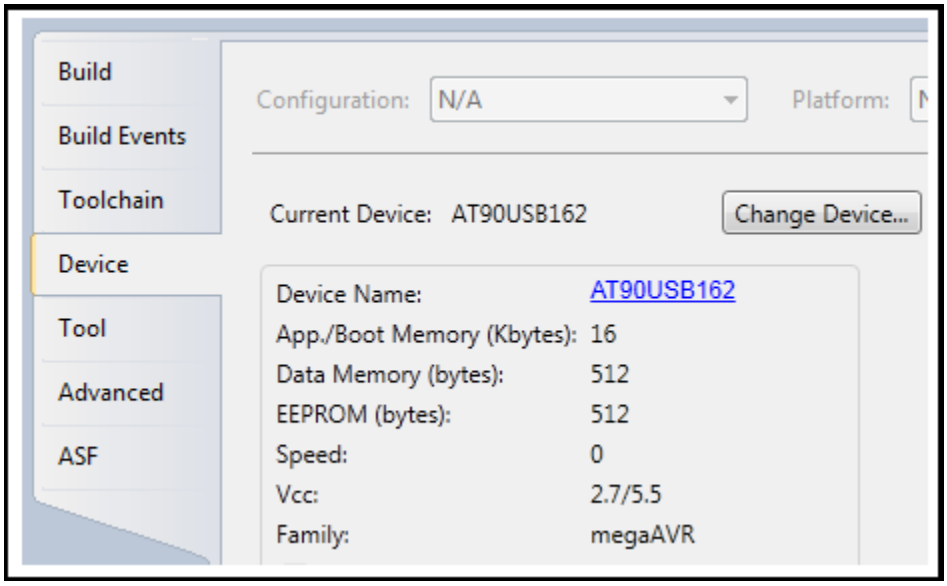

Figure 2: Project Properties: Select "Device: AT90USB162"

Step 2: Edit the Makefile in the Studio5 Project.

Open the Studio5 Project makefile using a text editor and change the MCU and BOARD variables.

```
# MCU name
# Origianl was: MCU = at90usb1287
MCU = at90usb162

# Target architecture (see library "Board Types" documentation).
ARCH = AVR8


# Target board (see library "Board Types" documentation, NONE for projects not requiring
# LUFA board drivers). If USER is selected, put custom board drivers in a directory called
# "Board" inside the application directory.
# Origianl was:  BOARD = USBKEY
BOARD = OLIMEX162

# Processor frequency.
#     This will define a symbol, F_CPU, in all source code files equal to the
#     processor frequency in Hz. You can then use this symbol in your source code to
#     calculate timings. Do NOT tack on a 'UL' at the end, this will be done
#     automatically to create a 32-bit value in your source code.
#
#     This will be an integer division of F_USB below, as it is sourced by
#     F_USB after it has run through any CPU prescalers. Note that this value
#     does not *change* the processor frequency - it should merely be updated to
#     reflect the processor speed set externally so that the code can use accurate
#     software delays.
F_CPU = 8000000
```

Figure 3: Only two changes were required to the Studio5 project makefile.

Only two changes are required. Changing the microcontroller to at90usb162 and changing the board to OLIMEX162. Since Both the old USBKEY board and the OLIMEX162 board use an 8 MHz crystal, the F_CPU parameter didn't need to be changed.

Figure 4 lists the configuration variables that are in the project makefile.

LUFA Library > Related Pages > Getting Started > Configuring the Demos

Inside each makefile, a number of configuration variables are located, with the format "<VARIABLE NAME> = <VALUE>". For each application, the important variables which should be altered are:

- **MCU**, the target processor model
- **ARCH**, the target microcontroller architecture
- **BOARD**, the target board hardware
- **F_CPU**, the target CPU master clock frequency, after any prescaling
- **F_USB**, the target raw input clock to the USB module of the processor
- **CDEFS**, the C preprocessor defines which configure options the source code
- **LUFA_PATH**, the path to the LUFA library source code
- **LUFA_OPTS**, the compile time LUFA options which configure the library features

These values should be changed to reflect the build hardware.

Figure 4: LUFA Documentation showing the configuration variables in the project makefile.

Figure 5 shows the AVR8 microcontrollers supported by LUFA.

LUFA Library > Related Pages > About LUFA > Device and Hardware Support > Atmel 8-Bit AVR

Supported Microcontroller Models

Currently supported AVR8 models:

Part	USB Device Mode	USB Host Mode
AT90USB82	Yes	No
ATMEGA8U2	Yes	No
AT90USB162	Yes	No
ATMEGA16U2	Yes	No
ATMEGA16U4	Yes	No
ATMEGA32U2	Yes	No
ATMEGA32U4	Yes	No
ATMEGA32U6	Yes	No
AT90USB646	Yes	No
AT90USB647	Yes	Yes
AT90USB1286	Yes	No
AT90USB1287	Yes	Yes

Figure 5: List of microcontrollers supported by LUFA

Figure 6 shows the different boards supported by LUFA. Of course, you can also use LUFA with your own custom board, in which case you would specify BOARD = NONE in the makefile.

LUFA Library>Related Pages>About LUFA>Device and Hardware Support>Atmel 8-Bit AVR

Supported Atmel Boards

Currently supported Atmel AVR8 boards (see **Board Types**):

- AT90USBKEY
- ATAVRUSBRF01
- EVK527
- RZUSBSTICK
- STK525
- STK526
- XPLAIN (Excluding the blue XPLAINED family boards)

Supported Third Party Models

Currently supported third-party boards (see **Board Types** for makefile `BOARD` constant names):

- Adafruit U4 Breakout Board
- Arduino Uno
- Busware BUI
- Busware CUL V3
- Busware TUL
- Fletchtronics Bumble-B (using manufacturer recommended peripheral layout)
- Kernel Concepts USBFOO
- Linnix UDIP
- MattairTech JM-DB-U2
- Maximus USB
- Micropendous Boards (many versions)
- Microsin AVR-USB162
- Minimus USB
- Olimex AVR-USB-162
- Paranoid Studio's US2AX (V1, V2 and V3 hardware revisions)
- PJRC Teensy (1.x and 2.x versions)
- Sparkfun U2 Breakout Board
- TCNISO Blackcat USB JTAG
- Tempusdictum Benito
- Tom's USBTINY-MKII (all revisions and versions)
- Custom User Boards (with Board Drivers if desired, see **Writing LUFA Board Drivers**)

Figure 6: List of boards supported by LUFA.

The OLIMEX162 board has one button switch and one LED. Figure 7 shows the macros and functions that the LUFA board driver provides. Note that the functions listed are all static inline functions, so the program doesn't actually make a function call. The code is inserted where the function call appears.

Buttons.h and LEDs.h for this board can be found in the OLIMEX162 folder under Drivers\Board\AVR8.

From Buttons.h and LEDs.h in Drivers\Board\AVR8\OLIMEX162

```
For BOARD = OLIMEX162

Buttons
    #define BUTTONS_BUTTON1       (1 << 7)

    void Buttons_Init(void);
    void Buttons_Disable(void);
    uint8_t Buttons_GetStatus(void);

LEDs
    #define LEDS_LED1         (1 << 4)
    #define LEDS_ALL_LEDS     LEDS_LED1
    #define LEDS_NO_LEDS      0

    void LEDs_Init(void);
    void LEDs_Disable(void);
    void LEDs_TurnOnLEDs(const uint8_t LEDMask);
    void LEDs_TurnOffLEDs(const uint8_t LEDMask);
    void LEDs_SetAllLEDs(const uint8_t LEDMask);
    void LEDs_ChangeLEDs(const uint8_t LEDMask,
                         const uint8_t ActiveMask)

    LEDs_ToggleLEDs(const uint8_t LEDMask);
    uint8_t LEDs_GetLEDs(void);
```

Figure 7: Macros and functions for OLIMEX162 board's button switch and LED.

Note that for the OLIMEX162 board driver in LUFA-120219, the LED control is backwards. The LED turns on when it should turn off, and turns off when it should turn on.

Figure 8 shows the LUFA USB functions that are used in the modified VirtualSerial Demo.

```
USB_Init();
CDC_Device_USBTask(&VirtualSerial_CDC_Interface);
USB_USBTask();

CDC_Device_BytesReceived(&VirtualSerial_CDC_Interface)
CDC_Device_ReceiveByte(&VirtualSerial_CDC_Interface)
CDC_Device_SendByte(&VirtualSerial_CDC_Interface,CharIn)

CDC_Device_CreateStream(&VirtualSerial_CDC_Interface, &USBSerialStream);
fputs("Any String \n\r", &USBSerialStream);

CurrentDTRState  = (CDCInterfaceInfo->State.ControlLineStates.HostToDevice &
                    CDC_CONTROL_LINE_OUT_DTR);
```

Figure 8: LUFA USB functions used in the modified VirtualSerial Demo

Step 3: Making Changes to VirtualSerial.h and VirtualSerial.c

The USBKEY board had a Joystick with 5 switches and 4 LEDs. The OLIMEX162 board has one button switch and one LED, so all the code referencing the Joystick and LEDs was removed, and code added for the OLIMEX LED and button.

For the modified program, pressing or releasing the button switch sends a text message to the host. Commands sent by the host (from a terminal emulation program running on the PC) are used to turn the LED on and off.

Figure 9 shows the operation of the modified demo.

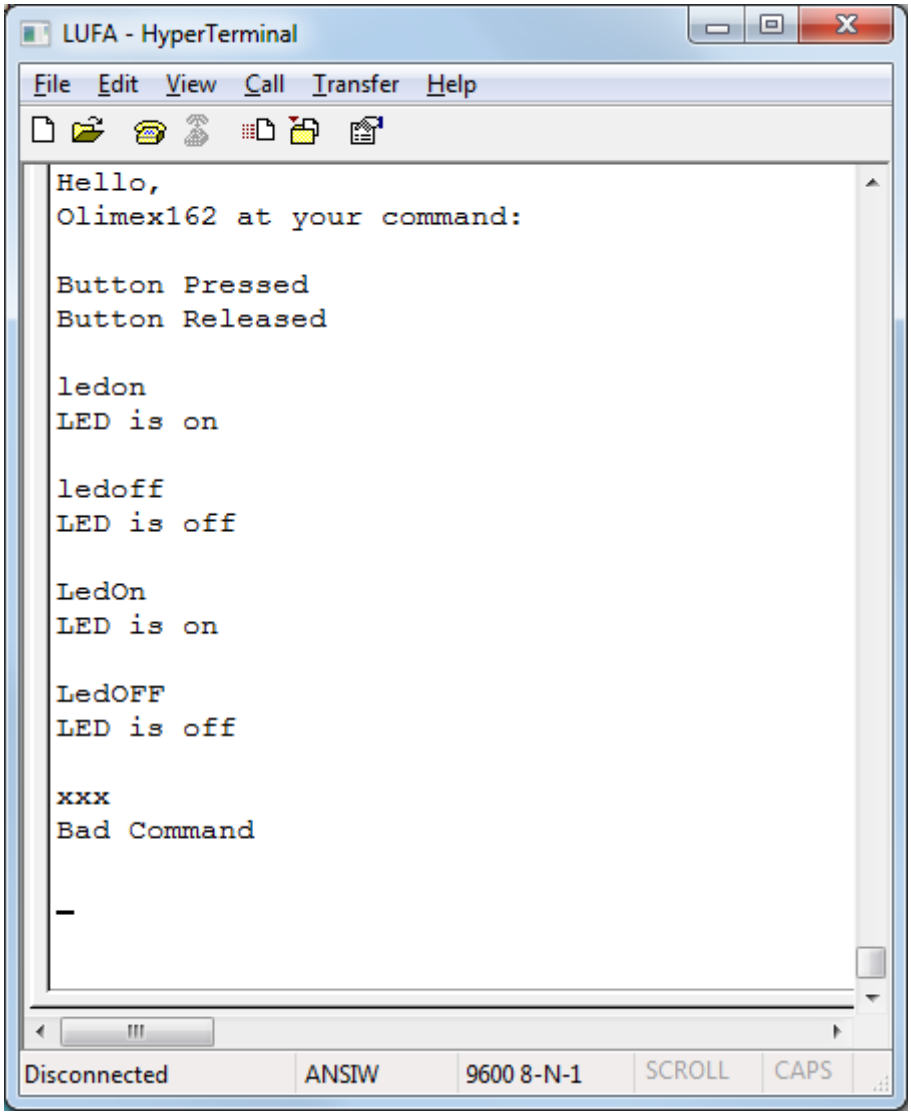

Figure 9: The Terminal program 'talking' to the OLIMEX162 board.

Note the "Hello, Olimex162 …" at the top of the screen. The board generated this when it detected that the terminal program on the PC had connected to the COM port. This was accomplished by detecting when the (virtual) DTR line was asserted. The event handler for doing this is EVENT_CDC_Device_ControLineStateChanged(…) and the mask is CDC_CONTROL_LINE_OUT_DTR.

Step 3: Descriptors.c and the Windows .inf files

Specifying VID and PID are shown in Figure 10 below. The ReleaseNumber was increased from 00.01 to 00.02 to force an update of the ManufacturerString and the ProductString.

```
const USB_Descriptor_Device_t PROGMEM DeviceDescriptor =
{
    .Header                 = {.Size = sizeof(USB_Descriptor_Device_t), .Type = DTYPE_Device},

    .USBSpecification       = VERSION_BCD(01.10),
    .Class                  = CDC_CSCP_CDCClass,
    .SubClass               = CDC_CSCP_NoSpecificSubclass,
    .Protocol               = CDC_CSCP_NoSpecificProtocol,

    .Endpoint0Size          = FIXED_CONTROL_ENDPOINT_SIZE,

    .VendorID               = 0x03EB,
    .ProductID              = 0x2044,
//      .ReleaseNumber          = VERSION_BCD(00.01),
    .ReleaseNumber          = VERSION_BCD(00.02),


    .ManufacturerStrIndex   = 0x01,
    .ProductStrIndex        = 0x02,
    .SerialNumStrIndex      = USE_INTERNAL_SERIAL,

    .NumberOfConfigurations = FIXED_NUM_CONFIGURATIONS
};
```

Figure 10: VID (Vendor ID) & PID (Product ID) are specified in file Descriptors.c located the project folder.

```
const USB_Descriptor_String_t PROGMEM ManufacturerString =
{
    .Header                 = {.Size = USB_STRING_LEN(19), .Type = DTYPE_String},

    .UnicodeString          = L"LUFA OLIMEX162 Demo"
};

/** Product descriptor string. This is a Unicode string containing the product's details
 *  and is read out upon request by the host when the appropriate string ID is requested,
 *  Descriptor.
 */
const USB_Descriptor_String_t PROGMEM ProductString =
{
    .Header                 = {.Size = USB_STRING_LEN(14), .Type = DTYPE_String},

    .UnicodeString          = L"OLIMEX162 Demo"
};
```

Figure 11: ManufacturerString & ProductString are also specified in the file Descriptors.c

Now, you might think that specifying the VID, PID, and ManufacturerString in the Descriptors.c file would be all that's necessary. But for Windows, it's not.

As Dean put it: (http://www.avrfreaks.net/index.php?name=PNphpBB2&file=viewtopic&p=938239#938239)

> Windows is derpy, and uses the strings in the driver INF file for the device, rather than the perfectly good strings it reads out from the device. You need to update the INF file and get Windows to reload it to update the strings.

My experience was that Windows would read the ProductString from the device, but used the ManufacturerString from the .inf file.

The Windows Setup File (.inf) for the VirtualSerial Demo can be found in its LUFA folder.

The modified version is shown in Figure 12. Both the ManufacturerString (listed as MFGNAME in the .inf file) and the VID and PID values in the Descriptors.c file must be included in the .inf file.

```
;------------------------------------------------------------------------------
;  Vendor and Product ID Definitions
;------------------------------------------------------------------------------
; When developing your USB device, the VID and PID used in the PC side
; application program and the firmware on the microcontroller must match.
; Modify the below line to use your VID and PID.  Use the format as shown below.
; Note: One INF file can be used for multiple devices with different VID and PIDs.
; For each supported device, append ",USB\VID_xxxx&PID_yyyy" to the end of the line.
;------------------------------------------------------------------------------
[SourceDisksFiles]
[SourceDisksNames]
[DeviceList]
%DESCRIPTION%=DriverInstall, USB\VID_03EB&PID_2044

[DeviceList.NTamd64]
%DESCRIPTION%=DriverInstall, USB\VID_03EB&PID_2044


;------------------------------------------------------------------------------
;  String Definitions
;------------------------------------------------------------------------------
;Modify these strings to customize your device
;------------------------------------------------------------------------------
[Strings]
MFGFILENAME="CDC_vista"
DRIVERFILENAME ="usbser"
;
; Modified by Chuck99
MFGNAME="LUFA OLIMEX162 Demo"
;
INSTDISK="LUFA CDC Driver Installer"
DESCRIPTION="Communications Port"
SERVICE="USB RS-232 Emulation Driver"
```

Figure 12: Part of the modified .inf file for Windows

Table 1 shows the VID and PID values available for LUFA (non-commercial) projects.

LUFA Library > Related Pages > Developing With LUFA > VID and PID Values

VID	PID	Usage
0x03EB	0x2040	Test VID/PID (See below)
0x03EB	0x2041	Mouse Demo Application
0x03EB	0x2042	Keyboard Demo Application
0x03EB	0x2043	Joystick Demo Application
0x03EB	**0x2044**	**CDC Demo Application**
0x03EB	0x2045	Mass Storage Demo Application
0x03EB	0x2046	Audio Output Demo Application
0x03EB	0x2047	Audio Input Demo Application
0x03EB	0x2048	MIDI Demo Application
0x03EB	0x2049	MagStripe Project
0x03EB	0x204A	CDC Bootloader
0x03EB	0x204B	USB to Serial Demo Application
0x03EB	0x204C	RNDIS Demo Application
0x03EB	0x204D	Combined Keyboard and Mouse Demo Application
0x03EB	0x204E	Dual CDC Demo Application
0x03EB	0x204F	Generic HID Demo Application
0x03EB	0x2060	Benito Programmer Project
0x03EB	0x2061	Combined Mass Storage and Keyboard Demo
0x03EB	0x2062	Combined CDC and Mouse Demo
0x03EB	0x2063	Mass Storage/HID Interface Datalogger Project
0x03EB	0x2064	Interfaceless Control-Only LUFA Devices
0x03EB	0x2065	Test and Measurement Demo
0x03EB	0x2066	Multiple Report Keyboard/Mouse HID Demo
0x03EB	0x2067	HID Class Bootloader
0x03EB	0x2068	Virtual Serial/Mass Storage Demo
0x03EB	0x2069	Webserver Project
0x03EB	0x206A	Media Control Project
0x03EB	0x206B	Currently Unallocated
0x03EB	0x206C	Currently Unallocated
0x03EB	0x206D	Currently Unallocated
0x03EB	0x206E	Currently Unallocated
0x03EB	0x206F	Currently Unallocated

Table 1: Assignment of VID and PID values donated to LUFA by Atmel (for non-commercial use).

Onward and Upward

So far we've imported the LUFA demo and modified it to run on the OLIMEX162 board. The next rung of the ladder, shown in Part 3, is to create a new project in Studio5 which uses LUFA to add USB CDC connectivity.