Microcontroller USB Library in C++

https://github.com/TimB-QNA/ClassyUSB

Tim Brocklehurst

December 26, 2024

Contents

Chapter 1

Introduction

The most pertinent question when writing a new library is why. For USB devices this is very simple. Existing libraries are highly complex, appallingly documented, and require a PhD in software engineering to add or remove components. That's ignoring the fact that many are tied to vendors, and as such the bloat is appalling, as every device has to be supported. Great for quickly throwing some code together, as a proof of concept, rubbish for actually building kit which is maintainable. For professional work, I wouldn't use the Arduino libraries, it's a good introduction to microcontrollers but it's probably not the solution you're looking for long-term.

This library attempts to fix some of those problems. There are always some tradeoffs, and its design favours ease of use over efficiency. Currently not many platforms are supported, but adding platform support is a matter of inheriting one class and writing the hardware support; everything else remains the same.

1.1 Supported Platforms

Only one platform is currently supported.

Manufacturer	IC	Core Technology
Atmel (Microchip)	SAMD21	ARM Cortex M0

Table 1.1: Supported microcontrollers

Chapter 2

Code Layout

The code laid out such that it is easy to add functionalilty and hardware support. The code is split into a number of classes which handle the hardware interface and device functions.

The main class is *usbDev*. This handles all the device-level operations and manages the device functionality. *usbDev* must be subclasssed to provide hardware support. The subclassed version is instantiated in the main program. The library treats every configuration as a composite device. In USB parlance, a composite device has multiple components which can have different functionality. The library follows the naming conventions of USB components, which provide functionality. Components can be added using the "addComponent()" member up to the available number of hardware endpoints.

For example, serial over usb (CDC_ACM) on an Atmel SAMD21 might be configured thus:

```
#include "ClassyUSB/src/hardware/samd21usbDevice.h"
#include "ClassyUSB/src/CDC/usbCDC_ACM.h"

samd21usbDevice usb;
usbCDC_ACM acm;

int main(){
  ...
  usb.setVendorID(0x03EB);
  usb.setProductID(0x2404);

  usb.initialise();
  usb.addComponent( new usbCDC(&acm) );
  ...
}
```

The device descriptors for these components is handled automatically by the library; the user need only set the vendor and product ID; refer to USB.org for advice regarding the selection of these IDs.

Chapter 3

Bibliography