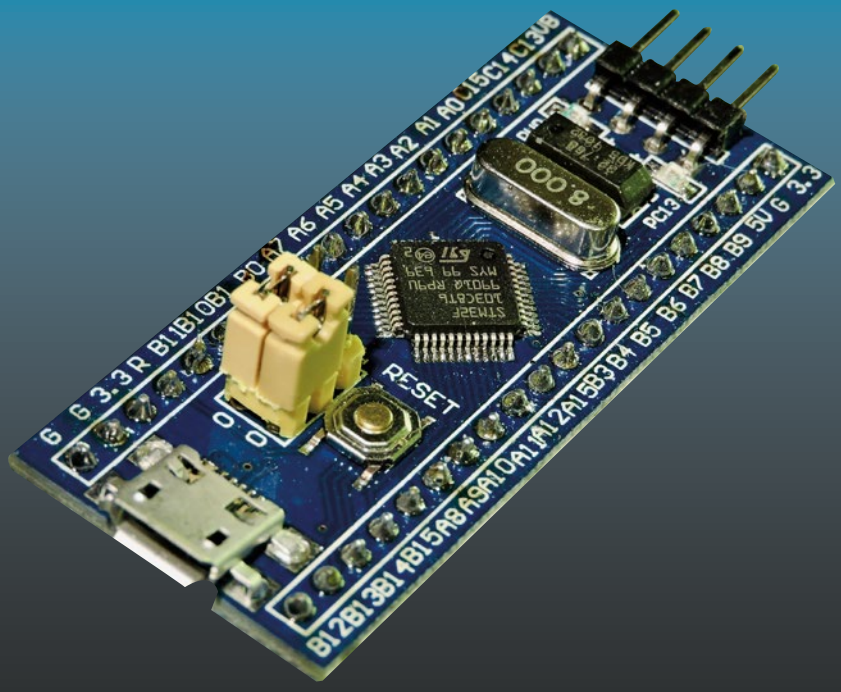
TECHNOLOGY IN ACTION™



Beginning STM32

Warren Gay

Apress®

About the Author

Warren Gay started out in electronics at an early age, dragging discarded TVs and radios home from public school. In high school he developed a fascination for programming the IBM 1130 computer, which resulted in a career-plan change to software development. Since graduating from Ryerson Polytechnical Institute, he has enjoyed a 30-plus-year software developer career, programming mainly in C/C++. Warren has been programming Linux since 1994 as an open source contributor and professionally on various Unix platforms since 1987.

Before attending Ryerson, Warren built an Intel 8008 system from scratch before there were CP/M systems and before computers got personal. In later years, Warren earned an advanced amateur radio license (call sign VE3WWG) and worked the amateur radio satellites. A high point of his ham-radio hobby was making digital contact with the Mir space station (U2MIR) in 1991.

Warren works at Datablocks.net, an enterprise-class ad-serving software services company where he programs C++ server solutions on Linux back-end systems.

CHAPTER 1

Introduction

There is considerable interest in the ARM Cortex platform today because ARM devices are found everywhere. Units containing ARM devices range from the small microcontroller embedded systems to cellphones and larger servers running Linux. Soon, ARM will also be present in higher numbers in the datacenter. These are all good reasons to become familiar with ARM technology.

With the technology ranging from microcontrollers to full servers, the question naturally arises: "Why study embedded device programming? Why not focus on end-user systems running Linux, like the Raspberry Pi?"

The simple answer is that embedded systems perform well in scenarios that are awkward for larger systems. They are frequently used to interface with the physical world. They go between the physical world and a desktop system, for example. The humble keyboard uses a dedicated MCU (microcontroller unit) to scan key switches of the keyboard and report key-press events to the desktop system. This not only reduces the amount of wiring necessary but also frees the main CPU from expending its high-performance computing on the simple task of noticing key-press events.

Other applications include embedded systems throughout a factory floor to monitor temperature, security, and fire detection. It makes little sense to use a complete desktop system for this type of purpose. Stand-alone embedded systems save money and boot instantly. Finally, the MCU's small size makes it the only choice in flying drones where weight is a critical factor.

The development of embedded systems traditionally required the resources of two disciplines:

- Hardware engineer
- Software developer

Frequently, one person is assigned the task of designing the end product. Hardware engineers specialize in the design of the electronic circuits involved, but eventually the

W. Gay, *Beginning STM32*, https://doi.org/10.1007/978-1-4842-3624-6_1

product requires software. This can be a challenge because software people generally lack the electronics know-how while the engineers often lack the software expertise. Because of reduced budgets and delivery times, the electronics engineer often becomes the software engineer as well.

There is no disadvantage to one person's performing both design aspects as long as the necessary skills are present. Whether you're an electronics engineer, software developer, hobbyist, or maker, there is nothing like real, down-to-earth practice to get you going. That is what this book is all about.

STM32F103C8T6

The device chosen for this book is the STMicroelectronics STM32F103C8T6. This part number is a mouthful, so let's break it down:

- STM32 (STMicroelectronics platform)
- F1 (device family)
- 03 (subdivision of the device family)
- C8T6 (physical manifestation affecting amount of SRAM, flash memory, and so on)

As the platform name implies, these devices are based upon a 32-bit path and are considerably more powerful than 8-bit devices as a result.

The F103 is one branch (F1 + 03) of the STM32 platform. This subdivision decides the CPU and peripheral capabilities of the device.

Finally, the C8T6 suffix further defines the capabilities of the device, like the memory capacity and clock speeds.

The STM32F103C8T6 device was chosen for this book because of the following factors:

- *very* low cost (as low as $2 US on eBay)
- availability (eBay, Amazon, AliExpress, etc.)
- advanced capability
- form factor

The STM32F103C8T6 is likely to remain the lowest-cost way for students and hobbyists alike to explore the ARM Cortex-M3 platform for quite some time. The device is readily available and is extremely capable. Finally, the form factor of the small PCB allows header strips to be soldered to the edges and plugged into a breadboard. Breadboards are the most convenient way to perform a wide array of experiments.

The MCU on a blue PCB (Figure 1-1) is affectionately known as the "Blue Pill," inspired by the movie *The Matrix*. There are some older PCBs that were red in color and were referred to as the "Red Pill." There are still others, which are black and are known as the "Black Pill." In this book, I'll be assuming you have the Blue Pill model. Apart from some USB deficiencies, there should be little other difference between it and the other models.

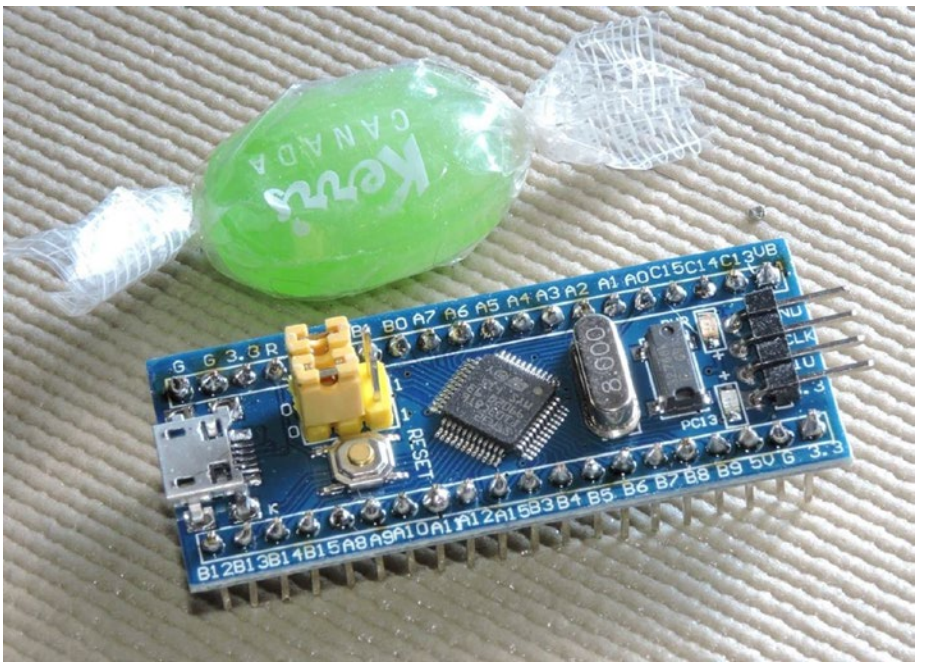


Figure 1-1. *The STM32F103C8T6 PCB (printed circuit board) with the header strips soldered in, often referred to as the "blue pill"*

Low cost has another advantage—it allows you to own *several* devices for projects involving CAN communications, for example. This book explores CAN communication using three devices connected by a common bus. Low cost means not being left out on a student budget.

The peripheral support of the STM32F103 is simply amazing when you consider its price. Peripherals included consist of:

- 4 x 16-bit GPIO Ports (most are 5-volt tolerant)
- 3 x USART (Universal Synchronous/Asynchronous Receiver/Transmitter)
- 2 x I2C controllers
- 2 x SPI controllers
- 2 x ADC (Analog Digital Converter)
- 2 x DMA (Direct Memory Address controllers)
- 4 x timers
- watch dog timers
- 1 x USB controller
- 1 x CAN controller
- 1 x CRC generator
- 20K static RAM
- 64K (or 128K) FLASH memory
- ARM Cortex M3 CPU, max 72 MHz clock

There are some restrictions, however. For example, the USB and CAN controllers cannot operate at the same time. Other peripherals may conflict over the I/O pins used. Most pin conflicts are managed through the AFIO (Alternate Function Input Output) configuration, allowing different pins to be used for a peripheral's function.

In the peripheral configuration, several separate clocks can be individually enabled to tailor power usage. The advanced capability of this MCU makes it suitable for study. What you learn about the STM32F103 family can be leveraged later in more advanced offerings like the STM32F407.

The flash memory is officially listed at 64K bytes, but you may find that it supports 128K. This is covered in Chapter 2 and permits good-sized applications to be flashed to the device.

What You Need

Let's briefly cover some items that you might want to acquire. Certainly, number one on the list is the Blue Pill device (see Figure 1-1). I recommend that you purchase units that include the header strips to be soldered onto the PCB so that you can easily use the unit on a breadboard (or presoldered, if you prefer).

These units are Buy-it-Now priced on eBay at around $2.13 US, with free shipping from various sellers. To find these deals, simply use the part number STM32F103C8T6 for your search. Chapters 18–19 use three of these units communicating with each other over a CAN bus. If you'd like to perform those experiments, be sure to obtain at least three units. Otherwise, the demo projects only involve one unit at a time. A spare is always recommended in case of an unhappy accident.

ST-Link V2 Programming Unit

The next essential piece of hardware is a programming adapter. Fortunately, these are also very economically priced. These can be found on eBay for about $2.17 US, with free shipping. Simply search for "ST-Link." Be sure to get the "V2" programmer since there is no point in using the inferior older unit.

Most auctions will include four detachable wires to connect the unit to your STM32 device. Try to buy a unit that includes these unless you already have a cable. Figure 1-2 illustrates the USB programmer, usable from Windows, Raspberry Pi, Linux, MacOS, and FreeBSD.

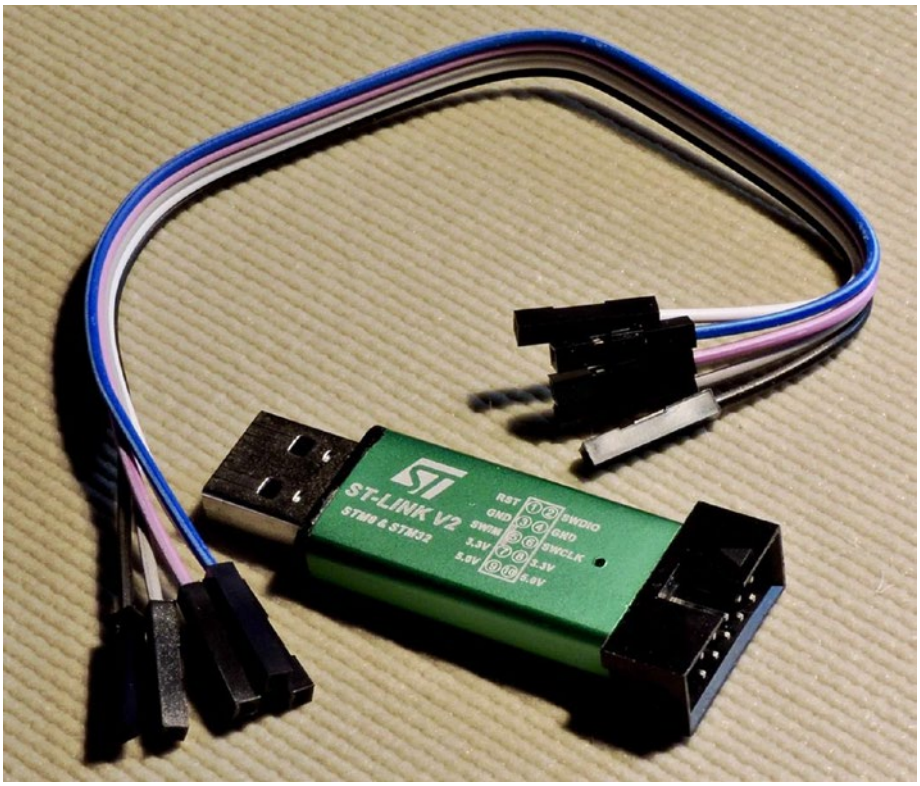


Figure 1-2. *ST-Link V2 programmer and cable*

The STM32F103C8T6 device can be programmed in multiple ways, but this book will only use the ST-Link V2 USB programmer. This will simplify things for you when doing project development and allows remote debugging.

A USB extension cable is useful with this unit. If you don't have one, you might consider getting one.

Breadboard

This almost goes without saying, but a breadboard is necessary to prototype experiments. The breadboard is a solderless way to quickly wire up experiments, try them, and then pull out the wires for the next experiment.

Many of the projects in this book are small, requiring space for one Blue Pill device and perhaps some LEDs or a chip or two. However, other experiments, like the one in Chapters 18–19, use three units communicating with each other over a CAN bus.

I recommend that you obtain a breadboard that will fit four units (this leaves a little extra hookup space). Alternatively, you could simply buy four small breadboards, though this is less convenient.

Figure 1-3 illustrates the breadboard that I am using in this book. It is not only large enough, but also has power rails at the top and bottom of each strip. The power rails are recommended, since this eases the wiring.

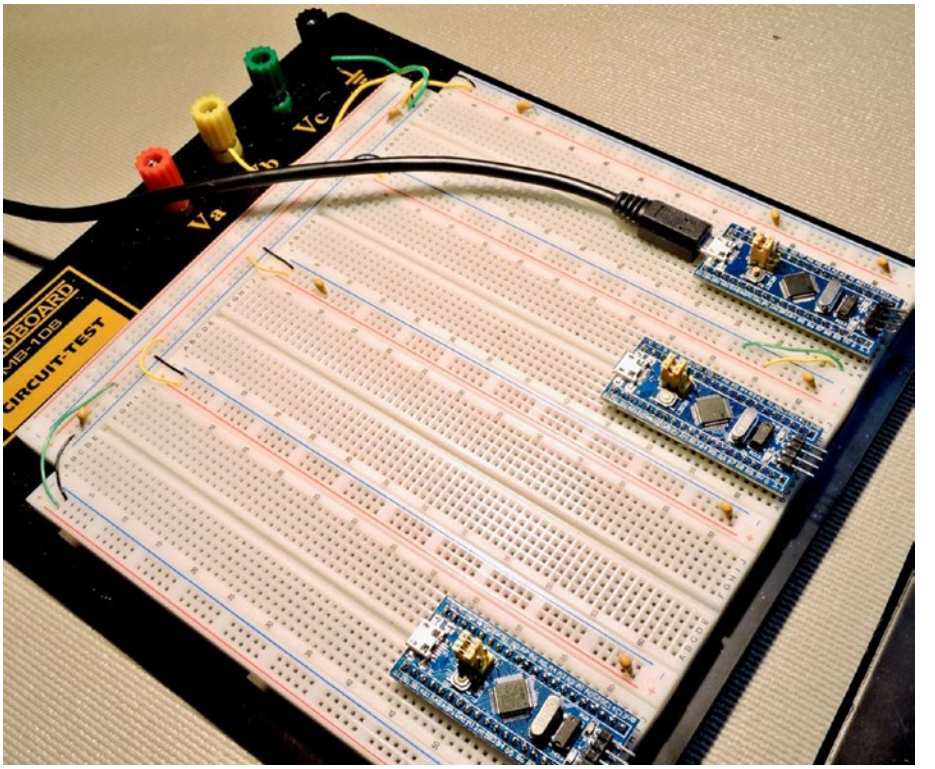


Figure 1-3. *A breadboard with supply rails*

DuPont (Jumper) Wires

You might not give much thought to the wiring of a breadboard, but you will find that DuPont wires can make a huge difference. Yes, you can cut and strip your own AWG22 (or AWG24) gauge wires, but this is inconvenient and time consuming. It is far more convenient to have a small box of wires ready to go. Figure 1-4 illustrates a small random collection of DuPont wires.

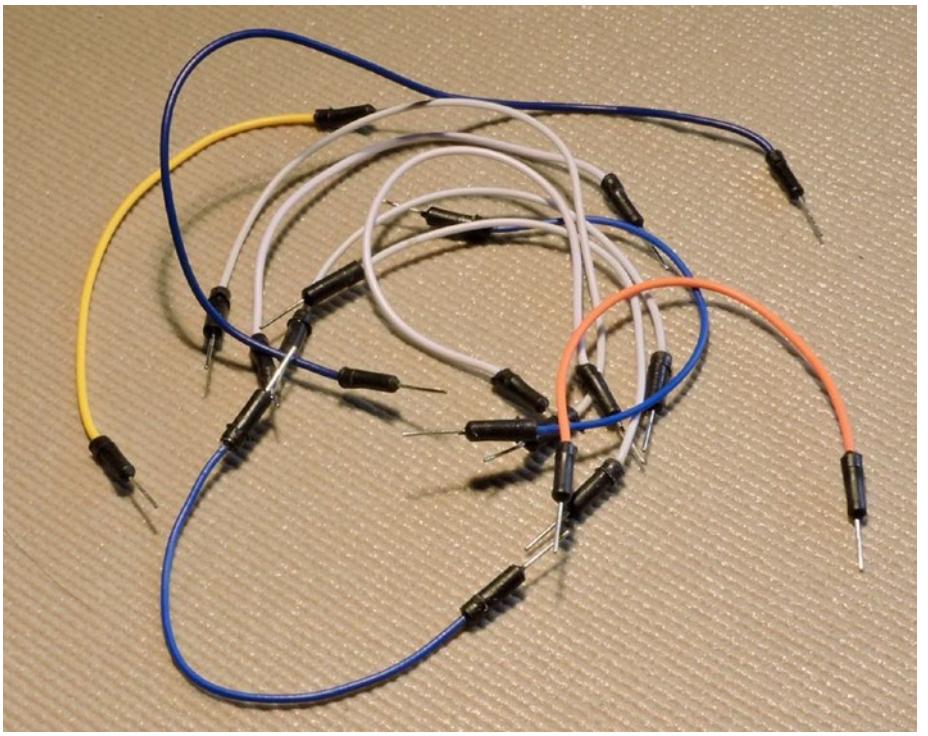


Figure 1-4. *A random collection of DuPont wires*

Male-to-male DuPont wires can be purchased in assorted sets on eBay for about the Buy-it-Now price of $2.00 US with free shipping. They might have auction titles like "65Pcs Male to Male Solderless Flexible Breadboard DuPont Jumper Cable Wires." I recommend that you get the *assorted* sets so that you get different colors and lengths. A search like "DuPont wires male -female" should yield good results. The "-female" keyword will eliminate any ads that feature female connectors.

0.1 uF Bypass Capacitors

You might find that you can get by without bypass caps (capacitors), but they are recommended (shown in Figure 1-5 as yellow blobs on the power rails). These can be purchased in quantity from various sources, including eBay.

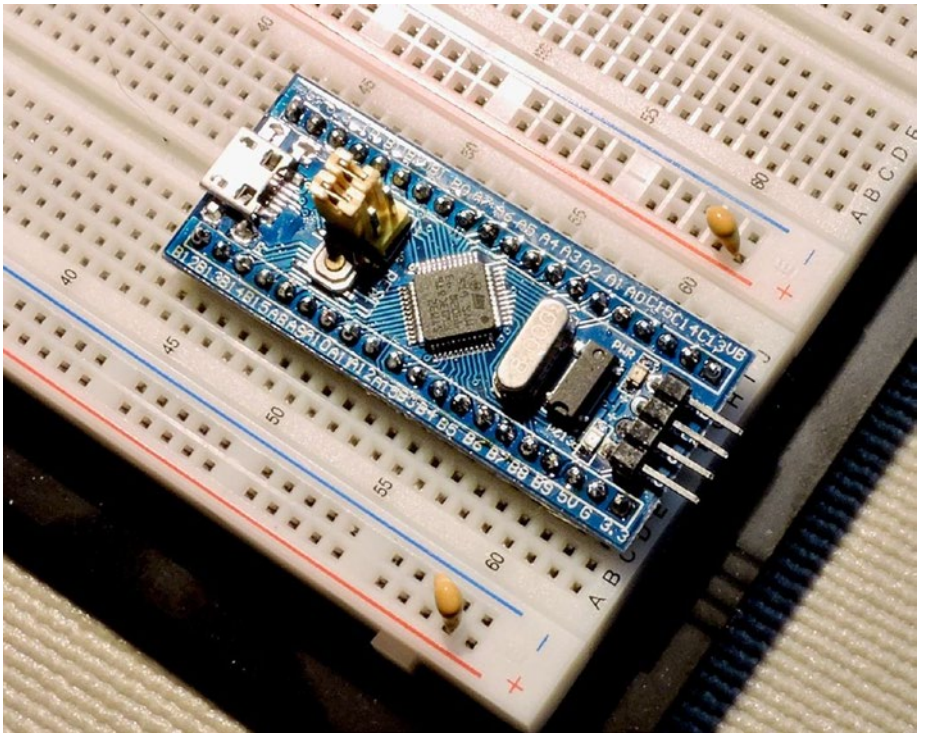


Figure 1-5. *Breadboard with STM32F103C8T6 and 0.1 uF bypass capacitors installed on the rails*

Try to buy quality capacitors like Metalized Polyester Film units if possible. The voltage rating can be as low as 16 volts. A few of these should be plugged into your supply rails on the breadboard, between the positive and negative rails, to filter out any voltage transients and noise.

USB TTL Serial Adapter

This device is essential for some of the projects in this book. Figure 1-6 illustrates the unit that I used. This serial adapter is used to communicate data to your desktop/laptop. Without a display, this allows you to communicate through a virtual serial link (via USB) to a terminal program.

There are several types of these available on eBay and elsewhere, but be careful to get a unit with *hardware flow control signals*. The cheapest units will lack these additional signals (look for RTS and CTS). Without hardware flow control signals, you will not be able to communicate at high speeds, such as 115200 baud, without losing data.

If you're running Windows, also be careful of buying FTDI (FTDI Chip) fakes. There were reports of FTDI software drivers bricking the fake devices at one time. Your choice doesn't have to include FTDI, but if the device claims FTDI compatibility, be aware and check your driver support.

You'll notice in Figure 1-6 that I have a tag tied to the end of the cable. That tag reminds me which colored wire is which so that I can hook it up correctly. You might want to do something similar.

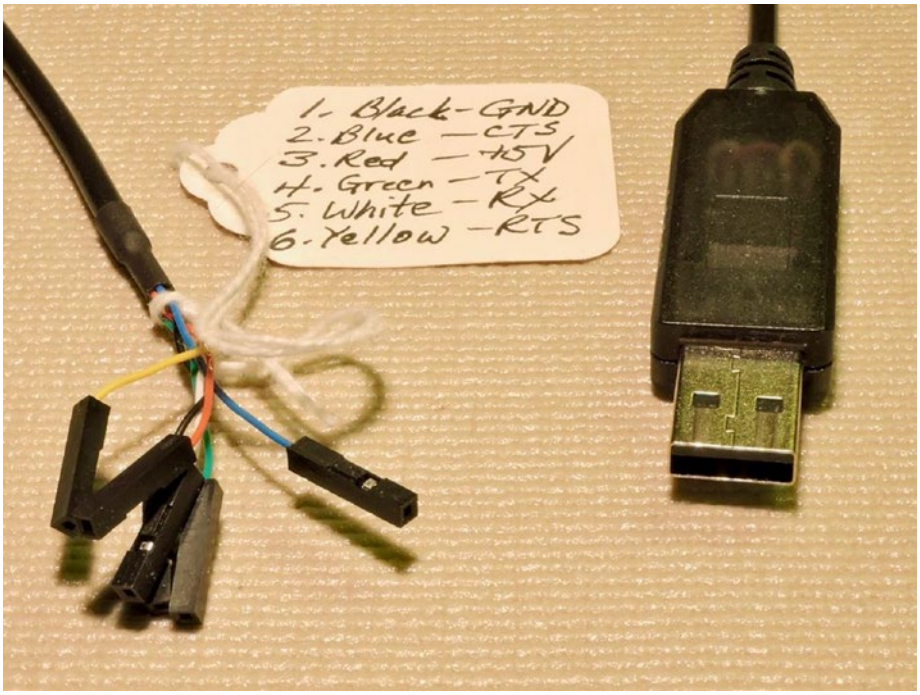


Figure 1-6. *A USB-to-TTL serial (5V) adapter cable*

These are normally 5-volt devices and are hence TTL compatible. Note, however, that one of the features of the STM32F103 family of devices is that many of the GPIO pins are 5-volt tolerant, even though the MCU operates from a +3.3-volt supply. This permits the use of these TTL adapters without causing harm. More will be said about this later. Other units can be purchased that operate at the 3.3-volt level or that can switch between 5 and 3.3 volts.

Power Supply

Most of the projects presented will run just fine off of the USB or TTL adapter power output. But if your project draws more than the usual amount of current, then you may need a power adapter. Figure 1-7 illustrates a good adapter to fit the breadboard power rails. It can be purchased from eBay for about $1.00 US with free shipping. Mine was advertised as "MB102 Solderless Breadboard Power Supply Module, 3.3V 5V for Arduino PCB Board." If your breadboard lacks power rails, you may need to shop for a different type of breadboard.

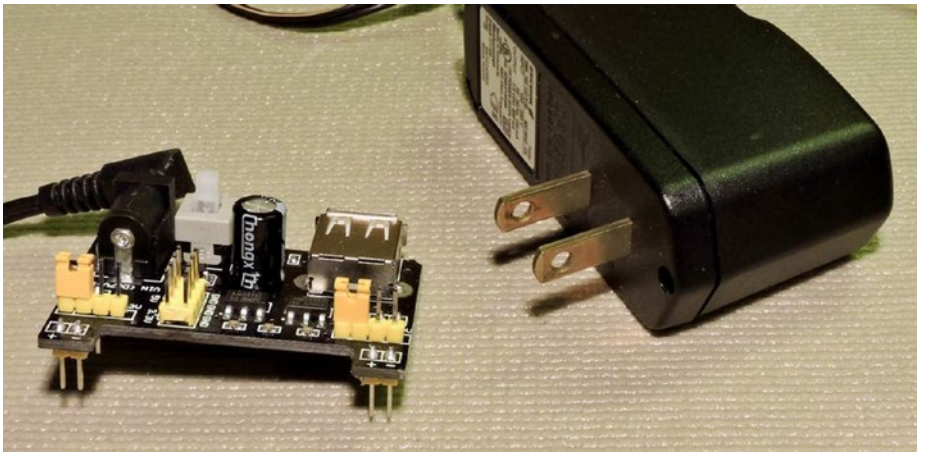


Figure 1-7. *A small breadboard power supply and 7.5 VDC adapter*

The MB102 is convenient because it can be jumpered to supply 3.3 or 5 volts. Additionally, it includes a power on/off button.

The other consideration is the *wall adapter* to supply the *input* power (this is not included). While the MB102 accepts up to 12 volts of input, I found that most 9 VDC wall adapters had an open circuit voltage near 13 volts or more. I feel that those are risky because if the cheap MB102 fails for any reason, the over-voltage might leak through and damage your MCU unit(s) as well.

Foraging through my junk box of "wall warts," I eventually found an old Ericsson phone charger rated at 7.5 VDC at 600 mA. It measured an unloaded voltage of 7.940 volts. This is much closer to the 5 and 3.3 volt outputs that the MB102 will regulate to. If you have to purchase a power adapter, I recommend a similar unit.

Small Stuff

There are some small items that you may already have. Otherwise, you will need to get some LEDs and resistors for project use. Figure 1-8 shows a random set of LEDs and a SIP-9 resistor.

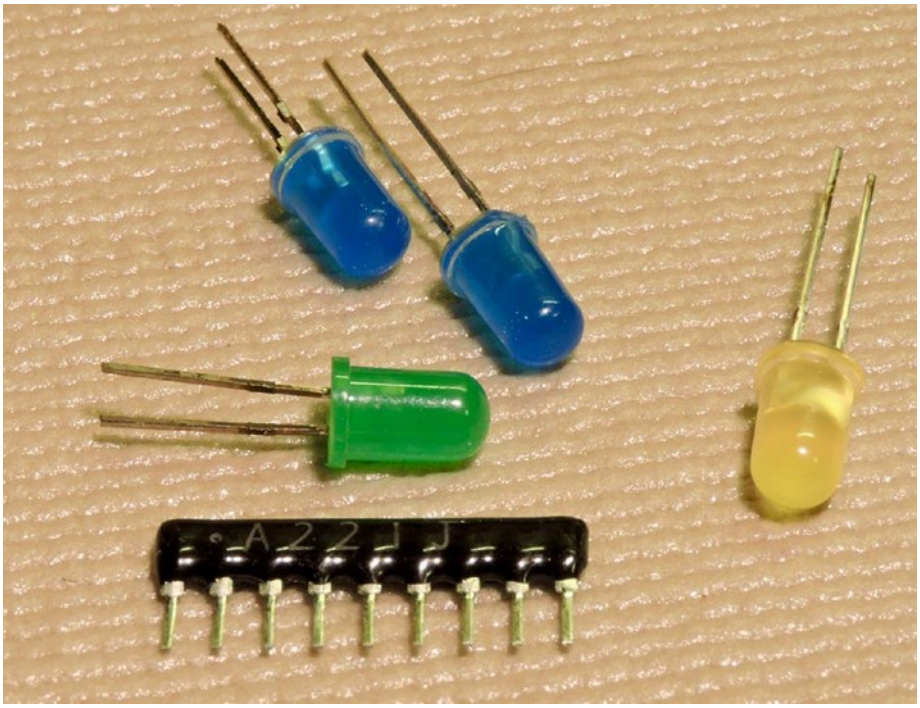


Figure 1-8. *A random collection of 5 mm LEDs and one SIP-9 resistor at the bottom*

Usually an LED is targeted for about 10 mA of current for normal brightness. Smaller LEDs only require maybe 2 to 5 mA. With a supply voltage near 3.3 volts, you'll want a resistor of about 220 Ω to limit the current (220 ohms limits the current to a maximum of approximately 7 mA). So, get a few 220 Ω resistors (1/8th watt will be sufficient).

Another part you may want to consider stocking is the SIP-9 resistor. Figure 1-9 illustrates the internal schematic for this part. If, for example, you want to drive eight LEDs, you would need eight current-limiting resistors. Individual resistors work but require extra wiring and take up breadboard space. The SIP-9 resistor, on the other hand, has one connection common to the eight resistors. The other eight connections are the other end of the internal resistors. Using this type of package, you can reduce the parts count and wiring required.

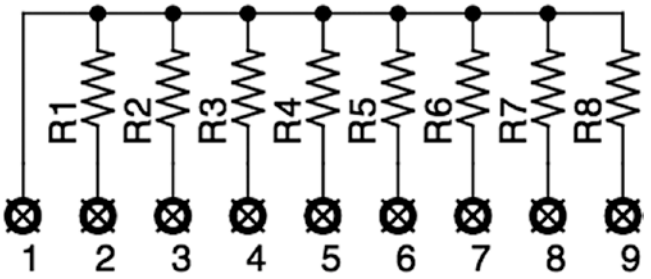


Figure 1-9. *The internal schematic view of a SIP-9 resistor*

Summary

This chapter has introduced the main actors that will appear in this book. It also itemized most of the things that you might need to acquire. The next chapter will guide you through the necessary steps of software installation. Once that is out of the way, the real fun can begin.

CHAPTER 3

Power Up and Blink

The unit that you purchased has likely already been preprogrammed to blink when it is powered up (perhaps you've checked this already). This makes it easy to test that it is a working unit. There are a few other important details regarding power, reset, and LEDs that need to be discussed in this chapter. Finally, the use of the ST-Link V2 programmer and a device probe will be covered.

Power

The STM32F103C8T6 PCB, otherwise known as the "Blue Pill" board, has a number of connections, including a few for power. It is not necessary to use all of the power connections at once. In fact, it is best to use only one set of connections. To clarify this point, let's begin with an examination of your power options. Figure 3-1 illustrates the connections around the edges of the PCB, including power.

W. Gay, *Beginning STM32*, https://doi.org/10.1007/978-1-4842-3624-6_3

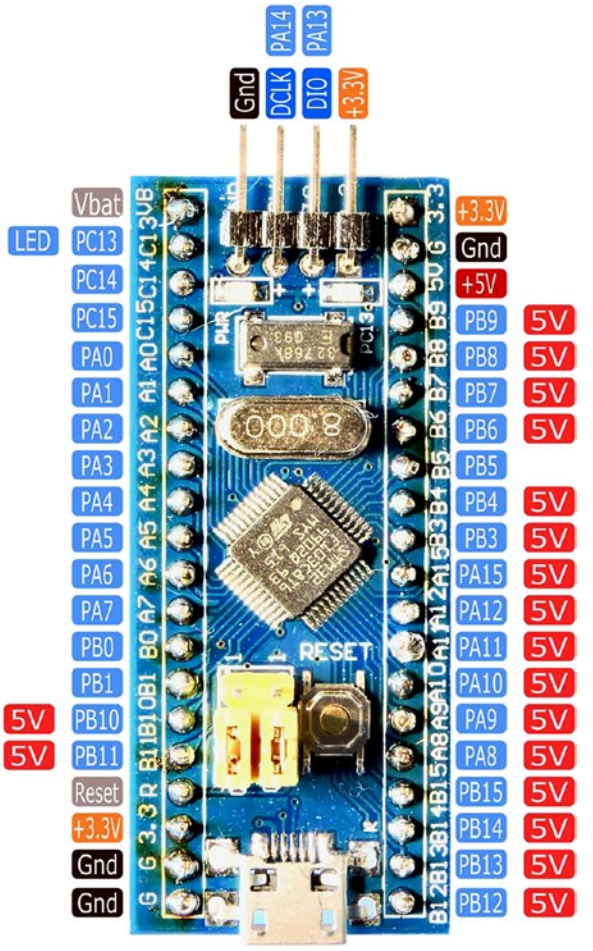


Figure 3-1. *Power and GPIO connections to the STM32F103C8T6 "Blue Pill" PCB. Power can be supplied to a +5V, +3.3V, or USB port, with the matching voltage. Pins marked as "5V" (with no plus sign) are 5-volt tolerant inputs. Pins marked with a plus sign are for power input.*

The four pins at the top end of the board (darker blue) are used for programming the device. Notice that the programming connection labeled DIO is also capable of being a GPIO PA13. Likewise, DCLK is capable of being a GPIO PA14. You'll discover how configurable the STM32 can be as we go through this book.

At the programming connector, note that the input supply voltage is +3.3 volts. This connection is electrically the same as any of the others that are labeled "+3.3V" around the PCB. These are shown in a light orange.

+3.3V Regulator

The STM32F103C8T6 chip is designed to operate from any voltage from 2 to 3.3 volts. The Blue Pill PCB provides a tiny +3.3-volt regulator labeled "U1" on the underside (see Figure 3-2). My unit used a regulator with an SMD code of 4A2D, which is an XC6204 series part. Yours may vary.

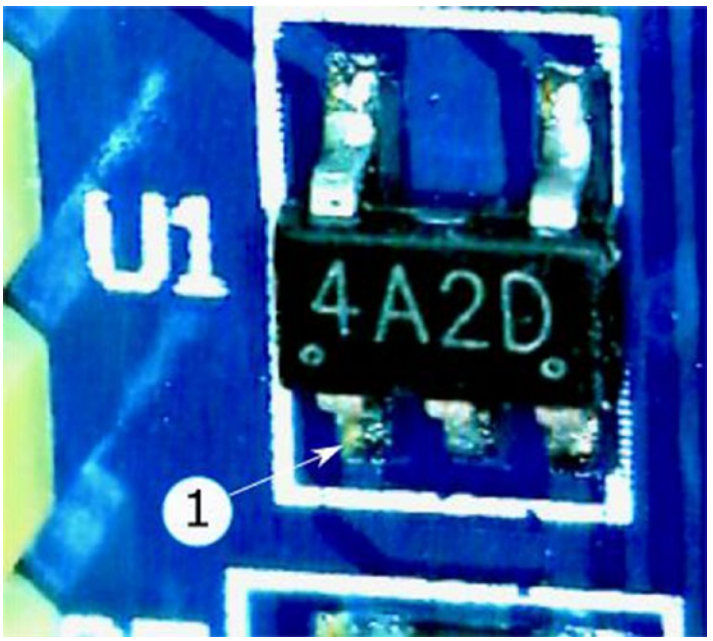


Figure 3-2. *The +3.3-volt regulator on the underside of the PCB. Pin 1 of the 4A2D (XC6204 series) regulator chip is marked.*

The official schematic for this board specifies the regulator as being the RT9193-33, which supports 300 mA.[1] It is possible that my PCB is a clone using a cheaper regulator chip. My XC6204 series regulator chip is limited to 150 mA. Unless you know the specifics of your unit, it is safest to assume 150 mA is the current limit.

The power performance of the MCU will be examined in a later chapter. But as a starting reference point, the blink program in the device as supplied uses about 30 mA (measured with the +5-volt input supply at 4.97 volts). This measurement includes the small additional current used by the regulator itself.

The datasheet for the STM32F103C8T6 documents the maximum current draw at about 50 mA. This document measurement was obtained with the external clock and

all peripherals enabled, while operating in "run mode" at 72 MHz. Subtracting 50 from your regulator max of 150 leaves you a current budget of about 100 mA from the +3.3-volt regulator. It's always good to know what the limits are!

USB Power/+5V

When powered by a USB cable, the power arrives by the Micro-USB B connector. This +5-volt supply is regulated to the +3.3 volts needed by the MCU. Similarly, at the top right of Figure 3-1, there is a pin labelled "+5V" (with a plus sign), which can be used as a power input. This goes to the same regulator input that the USB connector supplies.

Because of the low current requirements of your MCU, you can also power the unit from a TTL serial adapter. Many USB serial adapters will have a +5-volt line available that can supply your MCU. Check your serial adapter for specifications to be certain.

Be careful *not* to connect a USB cable and supply +5 volts *simultaneously*. Doing so could cause damage to your desktop/laptop through the USB cable. For example, if your +5-volt supply is slightly higher in voltage, you will be injecting current into your desktop USB circuit.

+3.3V Supply

If you have a +3.3-volt power supply, you can leave the +5V inputs *unconnected*. Connect your +3.3-volt power supply directly to the +3.3V input (make sure that the USB cable is *unplugged*). This works because the regulator disables itself when there is no input provided on the 5-volt input.

When supplying power to the +3.3-volt input, you are connecting your power to the VOUT terminal of the regulator shown in Figure 3-3. In this case, there is no 5-volt power flowing into VIN of the regulator. The CE pin is also connected to VIN, but when VIN is unconnected, the CE pin becomes grounded by a capacitor. A low level on CE causes the regulator to shut down its internal subsystems.

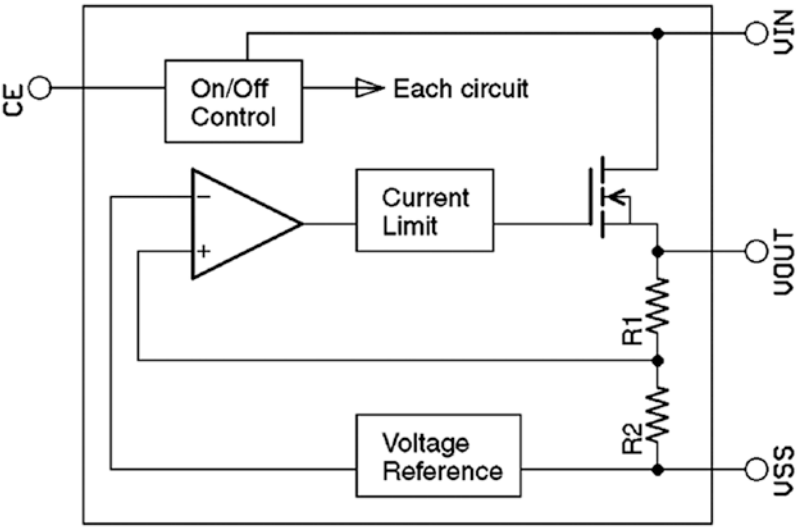


Figure 3-3. *Block diagram of the 5 to 3.3 volt regulator*

There is, however, a small amount of current flow into the regulator's voltage divider. This current will flow from your +3.3 volts to ground, through *internal* resistances R1 and R2 of the regulator. These resistances are high, and the current involved is negligible. But be aware of this when measuring current for ultra-low-power battery applications.

Caution Do not supply both +5 volts and +3.3 volts at the same time. This could cause damage to the regulator or your desktop when the USB cable is plugged in. Use a single power source.

One Power Source Rule

What I've been leading up to is the general advice to use just *one power source.* I can't stress enough that supplying your PCB with more than one power source can cause damage.

This tends to be obvious with the +3.3-volt and +5-volt supply inputs. What can *easily* be forgotten, however, is the *USB cable.* Consider that you could have power arriving from a USB serial adapter, the ST-Link V2 programmer, or the USB cable. Move slowly when changing your power arrangement, especially when switching from programming the device to your normal power configuration.

Certain applications may require you to use additional supplies; for example, when powering motors or relays. In those cases, you would supply the *external* circuits with the power they need but not the MCU PCB. Only the signals and the ground need to share connections. If this isn't clear, then assume the one power source rule.

Ground

The return side of the power circuit, or the negative side, is known as the ground connection. It is labeled in Figure 3-1 in black. All of these ground connections are electrically connected together. These pins can be used interchangeably.

Reset

The PCB also supplies a button labeled "RESET" and a connection on one side labeled "R." This connection permits an external circuit to reset the MCU if required. Figure 3-4 illustrates the push-button circuit, including the connection going to the MCU.

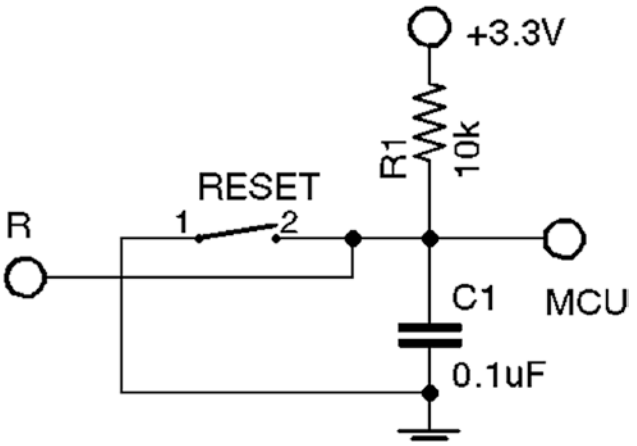


Figure 3-4. *The STM32F103C8T6 Reset circuit. Connection "R" is found on the edge of the PCB.*

Showtime

You've probably already tested your unit, but if you haven't yet then do so now. The safest and easiest way to do this is to use a USB cable with a Micro-USB B connector. Plug your cable into a USB power source, which doesn't have to be a computer.

Once powered, your unit should blink away. If not, then try pressing the Reset button. Also make sure that your boot-0 and boot-1 jumpers are positioned as shown in Figure 3-1 (both jumpers should be positioned to the side labeled "0").

There are two built-in LEDs. The LED on the left indicates that power has been applied (mine was yellow, but yours may differ). The LED at right is activated by GPIO port PC13 under program control (mine was red; again, yours may differ).

Caution Some have reported having their USB connector break off of the PCB. Be gentle inserting the Micro-USB B cable end.

If you are currently lacking a suitable USB cable, you can try the unit out if you can supply either +5 volts or +3.3 volts to the appropriate connection as discussed. Even a pair of dry cells in series for +3 volts will do (recall that this MCU will function on 2 to 3.3 volts).

Figure 3-5 illustrates the unit's being powered from the +3.3-volt connection at the top of the PCB where the programmer connects. Be careful when using alligator clips, ensuring they don't short to other pins. DuPont wires can be used with greater safety.

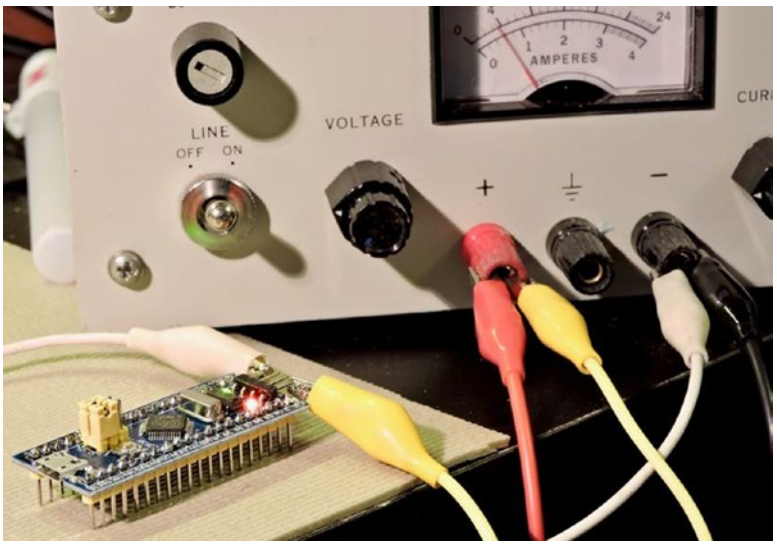


Figure 3-5. *The STM32F108C8T6 blinking and powered by a HP 6284A power supply using the top header strip (+3.3 volts)*

ST-Link V2

The next item to check off our list in this chapter is to hook up and run the `st-info` utility. When you get your programmer, you will likely just get four DuPont wires with female ends. This isn't real convenient but does work if you wire it correctly. If you switch devices to be programmed frequently, you'll want to make a custom cable for the purpose. The programmer hookup diagram is shown in Figure 3-6. It has been reported that different models of the programmer are available using different connections and wiring.

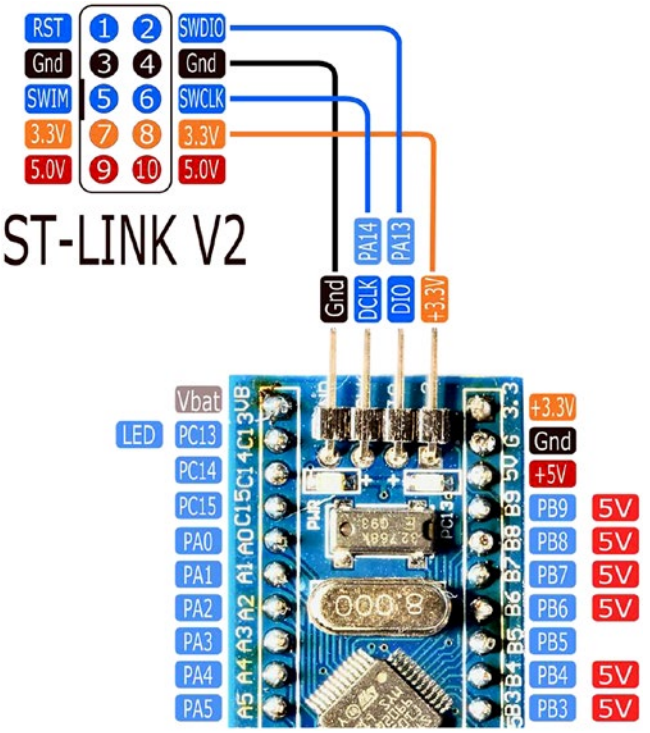


Figure 3-6. *ST-LINK V2 programmer hookup to STM32F103C8T6 device. Check the connections on the device you have, assome ST-Link programmers are different.*

With the programmer hooked up according to Figure 3-6, check your boot-0 and boot-1 jumpers located beside the Reset button. These should appear as they do in Figure 3-1 (with both jumpers close to the side marked "0").

Plug your ST-Link V2 programmer into a USB port or use a USB extension cable. Once you do this, the power LED should immediately light. Also, the PC13 LED should also blink if your unit still has the blink program in it. Figure 3-7 illustrates the setup.

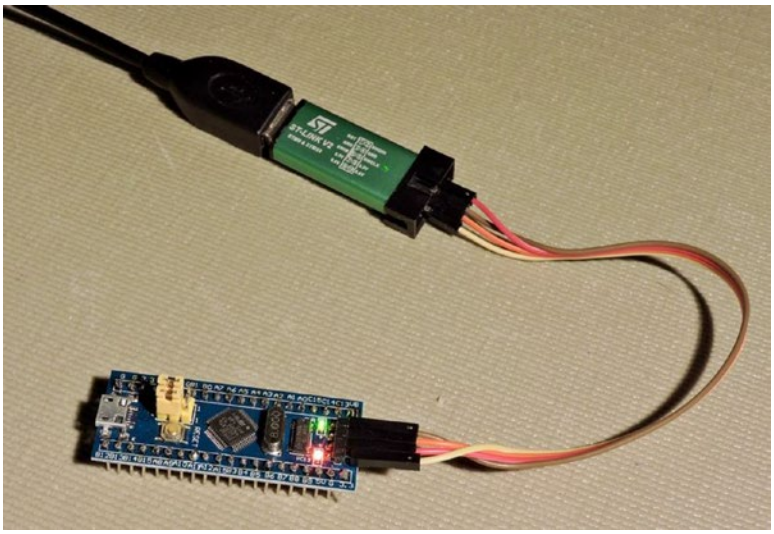


Figure 3-7. *ST-Link V2 programmer using a USB extension cable, hooked up to the STM32F103C8T6 using DuPont wires*

CHAPTER 6

USART

The Blue Pill PCB provides one GPIO-controlled LED to communicate by. Needless to say, this would be limiting if it were all you had. Perhaps the best early-development peripheral to pursue for communication is the USART (Universal Synchronous/Asynchronous Receiver/Transmitter).

This chapter will examine how to coax a STM32 USART to speak to your desktop through a USB serial adapter cable. A second project will demonstrate the same USART using two FreeRTOS tasks and a message queue.

USART/UART Peripheral

Within this book and technical literature at large, you will see the terms USART and UART used almost interchangeably. The difference between the two is in capability: USART is short for Universal *Synchronous*/Asynchronous Receiver/Transmitter. The UART moniker drops the synchronous function from the designation.

USART/UART peripherals send data serially over a wire. One wire is used for sending (TX) and another for receiving (RX) data. There is implied a common-ground connection between the two endpoints. Synchronous communication sometimes requires one end to act as the master and provide a clock signal. Asynchronous communication does not use a separate clock signal but does require both ends to agree precisely on a clock rate—known as the baud rate. Asynchronous communication begins with a start bit and ends with a stop bit for each character.

The USART peripherals provided by the STM32F103 are quite flexible. These can indeed function as USART or UART, depending upon configuration. This chapter will focus on the asynchronous mode for simplicity, and thus the name UART applies to the remainder of this chapter.

W. Gay, *Beginning STM32*, https://doi.org/10.1007/978-1-4842-3624-6_6

Asynchronous Data

Figure 6-1 provides an annotated scope trace of an asynchronous byte 0x65 being transmitted. This TTL (Transistor Transistor Logic) signal starts at the left with the line idle high (near 5 volts). The beginning of the character is marked by a low bit (near zero volts), known as the *start bit*. This alerts the receiver that data bits are following, with the least significant bits first (little endian). This example was an 8-bit value. The end of the character is marked by a *stop bit*. If the stop bit is not seen by the receiver, then an error is flagged.

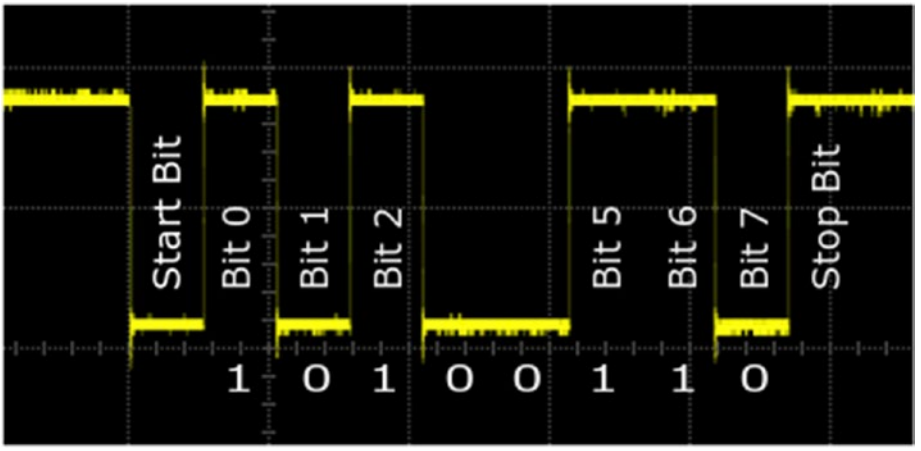


Figure 6-1. *Annotated scope trace of the UART signal for the value 0x65. Note how the least significant bits are sent first.*

Values being sent can be configured to be 8 or 9 bits in length. The last bit is the parity bit when enabled. The stop bit(s) end the transmission of a character and are configured as 0.5, 1, 1.5, or 2 bits in length.

USB Serial Adapters

A USB TTL serial adapter is an extremely helpful thing to own when working with microcontrollers. With very little hookup, you can use a terminal program on your desktop to communicate with your STM32. This eliminates the need for a costly LCD screen and keyboard.

If you haven't acquired one yet, here are some guidelines for what to look for:

- It must be a "TTL" adapter (signals at +5 volts or +3.3 volts).
- The USB device is supported by your operating system.
- The unit supports hardware flow control (RTS and CTS).

The TTL distinction is important. Normal RS-232 adapters operate at plus and minus about 3 volts or more. These *cannot* be wired directly to your STM32.

The *TTL* serial adapters, on the other hand, signal between zero and +5 volts and can be used with any of the *5-volt-tolerant inputs*. Fortunately, ST Microelectronics arranged that the receive line (RX) for UART 1 and 3 has 5-volt-tolerant inputs. Sending from the 3.3-volt STM32 works fine because the high signal is well above the threshold needed to be received as a 1-bit by the adapter.

Figure 6-2 illustrates one that is used by the author. These can be purchased for around $3 US on eBay. Be sure to get a unit that supports hardware flow control. These will include connections for RTS and CTS. Without hardware flow control, you won't be able to support higher rates like 115,200 baud without losing data. Be careful about FTDI units. In the past there have been reports of FTDI (FTDI Chip) drivers bricking FTDI clones. It is best to get a genuine FTDI unit or to avoid units claiming FTDI compatibility.

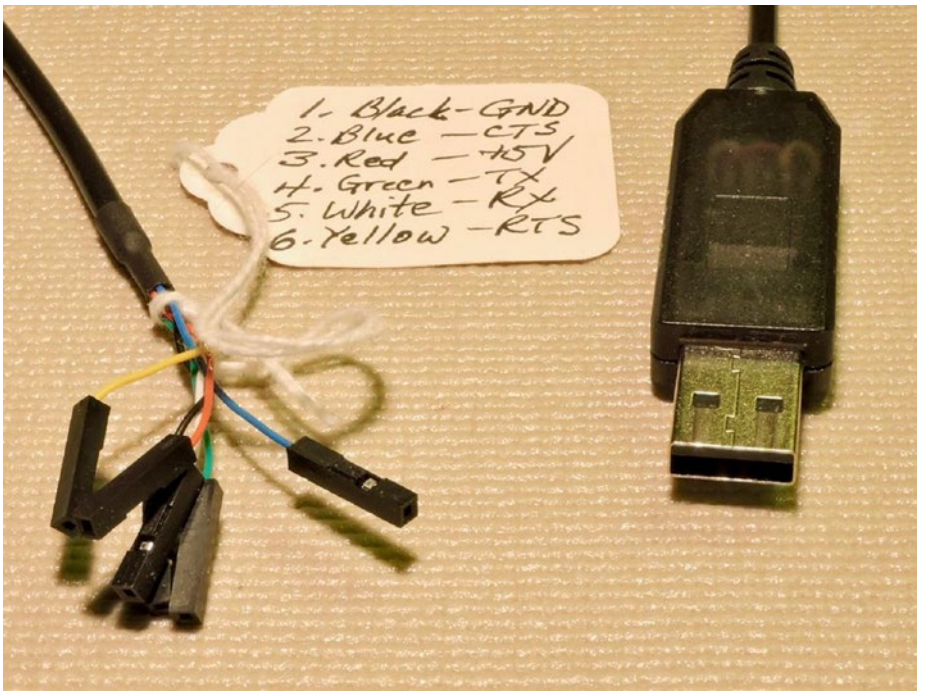


Figure 6-2. *Example USB TTL serial adapter cable. A tag was added as a colored-wires legend.*

Hookup

The two projects featured in this chapter require you to attach a USB TTL serial adapter so that your desktop can view the output. Figure 6-3 shows the hookup required.

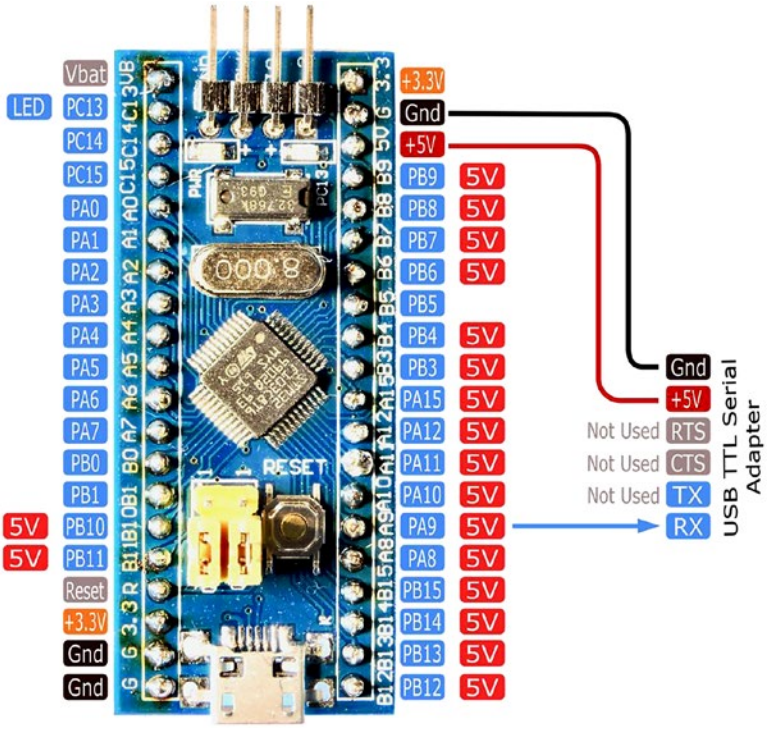


Figure 6-3. *USB TTL serial hookup for outgoing communication. No flow control is used in this example.*

The baud rate used in this chapter is 38,400, which is a relatively low speed. This allows us to avoid flow control for simplicity for our first demo. You should be able to power your device from the serial adapter's +5 volt line, as shown. Connect that supply to the +5 volt input on your Blue Pill so that the onboard regulator will provide the MCU 3.3 volts.

If this is not possible, then power the device separately. Be sure to make a common ground connection between the power source, the MCU, and the serial adapter.

Finally, note that only one data connection is required for these particular demos. This is because these demonstration programs only transmit and do not receive data from the desktop.

CHAPTER 7

USB Serial

One of the nice things about the STM32 MCU is the availability of the USB (Universal Serial Bus) peripheral. With USB, it is possible to communicate directly with a desktop platform in various modes. One of these flexible modes is USB's emulation of a serial link between the MCU and the desktop.

This chapter will explore the use of libopencm3 and FreeRTOS working together to provide a convenient means of communication. You will use the USB CDC class of operation (USB communication device class). This provides a very convenient means for interacting with your Blue Pill.

Blue Pill USB Issue

First, let's clear the air about the Blue Pill USB issue. What is this issue you may have read about in the Internet forums?

It turns out that the PCB is manufactured with a 10 kohm resistor (R_{10}) pullup resistor to +3.3 volts, which is incorrect. For full-speed USB, this is supposed to be 1.5 kohm. You can test this by measuring resistance with your DMM between the A12 pin on the PCB and the +3.3-volt pin. You will likely read 10 kohms.

This defect does not always prevent it from working, however. For example, I had no difficulty using USB from the STM32 to a MacBook Pro. But your mileage may vary. The hard way to correct this is to replace R_{10} on the PCB, but this is difficult because the resistor is so incredibly small.

Caution Many people have reported in online forums that their Blue Pill USB connector has broken off or become inoperable. Exercise extra-gentle care when inserting the cable.

W. Gay, *Beginning STM32*, https://doi.org/10.1007/978-1-4842-3624-6_7

Correction of the issue is best accomplished by placing another resistor in *parallel* with it. Placing a 1.8 kohm resistor in parallel with the 10 kohm resistor produces a combined resistance of 1.5 kohms. Figure 7-1 illustrates how the author soldered a resistor to one of his units. The 1/8-Watt resistor is simply soldered carefully between pins A12 and the +3.3-volt pin. It's not pretty, but it works!

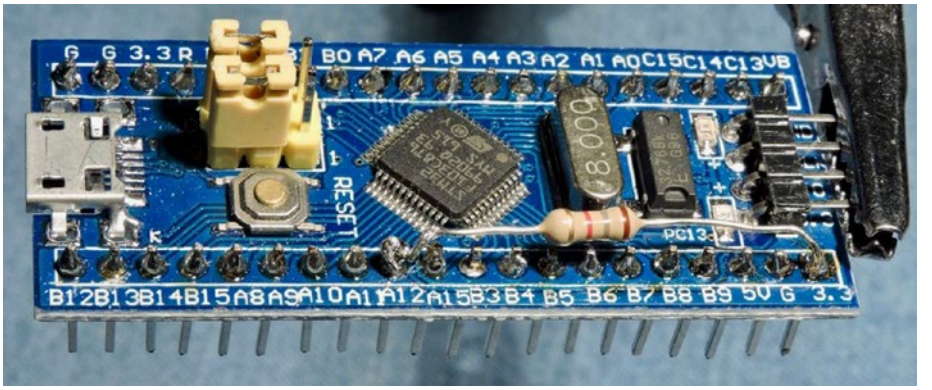


Figure 7-1. *Correcting the USB pullup by addition of a 1.8-kohm resistor*

To see how pullup resistance makes a difference, look at the scope trace in Figure 7-2. This is what the D+ line looked like with the default 10-kohm resistor.

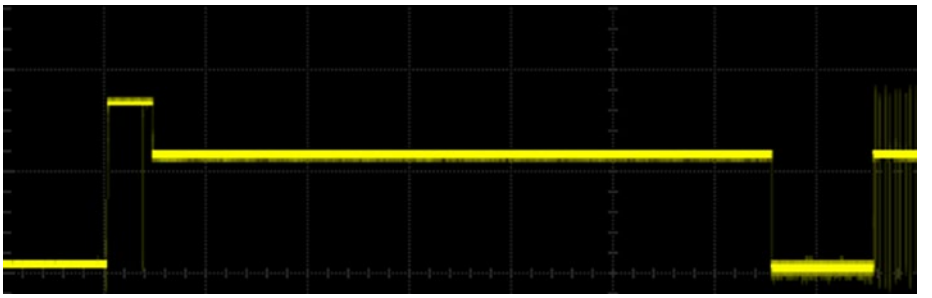


Figure 7-2. *D+ line scope trace using 10-kohm pullup resistance*

In the figure, you can see a rise at the start followed by a slump to perhaps the 70 percent level. To the right where the high-frequency signals begin, you can see that the signal rests at about the 70 percent level in between excursions. Attach this device to a different PC USB port or hub and the degradation might be worse.

Compare this to Figure 7-3, which is a scope trace after the 1.5-kohm pullup resistance was in effect.

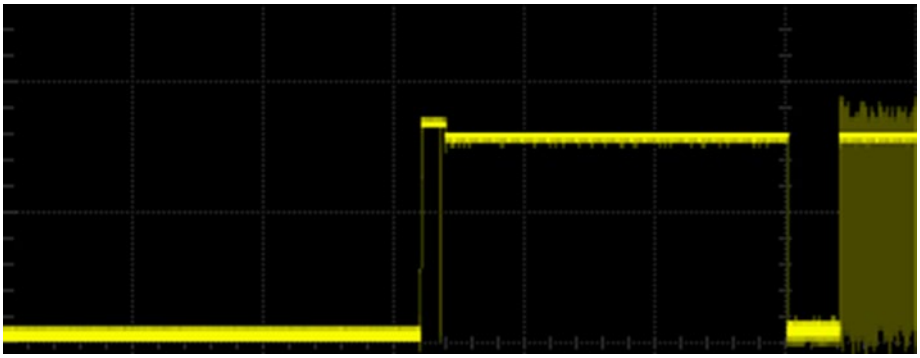


Figure 7-3. *D+ line scope trace with 1.5-kohm pullup resistance*

Ignoring capture-timing differences, you can see that the signal rests much higher, perhaps at the 90 percent level. This helps to assure improved signal thresholds.

Introduction to USB

USB is a popular means of communication from a personal computer to various peripherals, such as printers, scanners, keyboards, and a mouse. Part of its success is due to its standardization and low cost. The standard also includes USB hubs allowing the cost-effective extension of the network to accommodate additional devices.

In USB communication, the host directs all traffic. Each device is polled on a regular basis based upon its configuration and requirements. A keyboard infrequently needs to send data, for example, while a sound-recording device needs to send bulk recording data in real time. These differences are accommodated by the USB standard and are part of the device configuration.

Pipes and Endpoints

USB uses the concept of endpoints with connecting pipes to carry the data. The pipe carries the information, while the endpoints send or receive. Every USB device has at least one endpoint known as endpoint 0. This is a default and *control* endpoint, which allows host and device to configure device-specific operations and parameters. This occurs during device *enumeration*.

Figure 7-4 provides a high-level view of endpoints 0, 1, and 2 that we will be using in the example program. Technically, endpoint 0 is just one pipe. It is drawn here as two pipes because the control endpoint permits a response back to the host. All other endpoints have data travelling in one direction only. Note that the "In" and "Out" in Figure 7-4 are labeled according to the *host* controller's viewpoint.

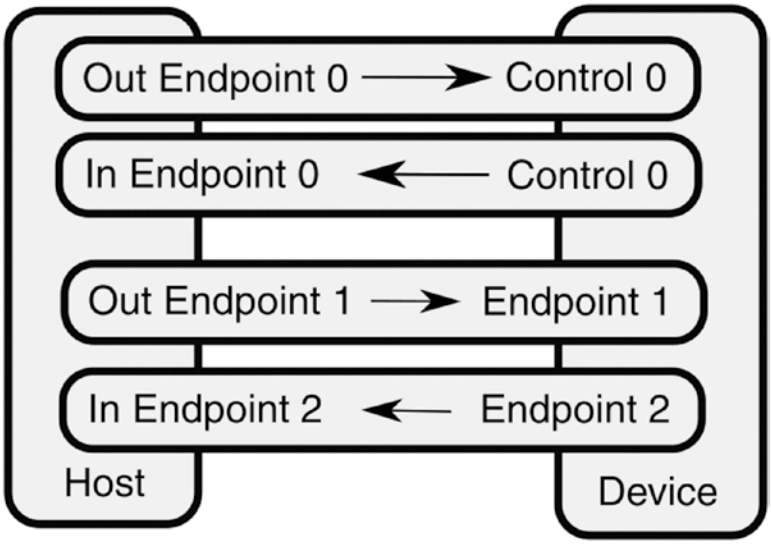


Figure 7-4. *USB pipes and endpoints*

A device may have additional endpoints, but our USB CDC example only needs two in addition to the required control endpoint 0:

- Endpoint 1 is the device's receiving endpoint (host's sending, specified as 0x01)
- Endpoint 2 is the device's sending endpoint (host's receiving, specified as 0x82)

As will be seen in the source code, bit 7 of the endpoint number indicates whether it is an input or output (with respect to the *host* controller). The value 0x82 indicates in hexadecimal that endpoint 2 (with bit 7) is sending (from the device's point of view). Unlike a TCP/IP socket, USB pipes transfer data in *one* direction only.

As you may have realized, one potentially confusing aspect of USB programming is that input and output are specified in the code from the host controller's point of view. For example, endpoint 0x82 is a receiving (input) endpoint from the host's point of view. This tends to be confusing when writing for the *device*. Be aware of that when setting up USB descriptors.

This necessarily has been a brief introduction to USB. Entire books have been written on the subject, and the interested reader is encouraged to seek them out. Our focus will be limited to the successful *use* of the USB peripheral for the benefit of our STM32. Let's get started!

USB Serial Device

With the MCU flashed and plugged into the system, you need to access it on your operating system as a serial device. This practice varies with the operating system, which complicates things slightly. The MCU source code is found in the following directory:

```
$ cd ~/stm32f103c8t6/rtos/usbcdcdemo
$ make clobber
$ make
$ make flash
```

The preceding steps will build and flash the code into your MCU device. The following sections will describe details on the desktop side of the USB conduit.

Linux USB Serial Device

Under Linux, with the STM32 flashed and plugged into a USB port, you can use the `lsusb` command to view the connected devices:

```
$ lsusb
Bus 002 Device 003: ID 0483:5740 STMicroelectronics STM32F407
```

In this example, I only had one device. Don't be worried about the STM32F407 designation. This is just the description given to the device ID 0483:5740 that ST Microelectronics registered. But how do you find out what device path to use? Try the following after plugging in your cable:

```
$ dmesg | grep 'USB ACM device'
[  709.468447] cdc_acm 2-7:1.0: ttyACM0: USB ACM device
```

This is obviously not very user friendly, but from this you find that the device name is `/dev/ttyACM0`. Listing it confirms this:

```
$ ls -l /dev/ttyACM0
crw-rw---- 1 root dialout 166, 0 Jan 25 23:38 /dev/ttyACM0
```

The next problem is having permissions to use the device. Notice that the group for the device is `dialout`. Add yourself to the `dialout` group (substitute `fred` with your own user ID):

```
$ sudo usermod -a -G dialout fred
```

Log out and log in again to verify that you have the correct group:

```
$ id
uid=1000(fred) gid=1000(fred) groups=1000(fred),20(dialout),24(cdrom),...
```

Being a member of the `dialout` group saves you from having to use `root` access to access the serial device.

MacOS USB Serial Device

Perhaps the simplest way to find the USB device under MacOS is to simply list the callout devices:

```
$ ls -l /dev/cu.*
crw-rw-rw- 1 root wheel 35,  1  6 Jan 15:14 /dev/cu.Bluetooth-Incoming-Port
crw-rw-rw- 1 root wheel 35,  3  6 Jan 15:14 /dev/cu.FredsiPhone-Wireless
crw-rw-rw- 1 root wheel 35, 45 26 Jan 00:01 /dev/cu.usbmodemFD12411
```

For the USB demo, the new device will appear as something like the path `/dev/cu.usbmodemFD12411`. The device number may vary, so look for `cu.usbmodem` in the pathname. Notice that all permissions are given.

Windows USB Serial Device

Serial devices under Windows show up as COM devices in the Device Manager once the cable is plugged in and the driver is installed. Figure 7-5 is an example screenshot.

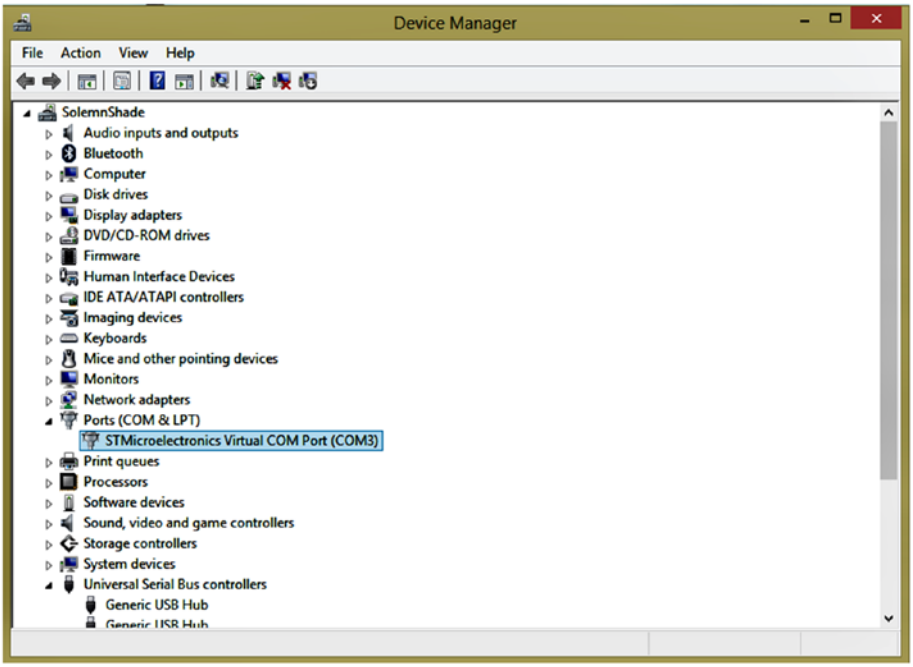


Figure 7-5. *Example Windows Device Manager dialog*

In this example, the USB device is attached as Windows port COM3. If you're using Cygwin under Windows, the device pathname is `/dev/ttyS2` (subtract 1 from the COM port number).

USB GPIO

The STMF103 series only supports USB on GPIO pins PA11 (`USB_DM`) and PA12 (`USB_DP`). There are *no alternate configurations* for USB. Further, there is no need to configure PA11 and PA12, because these are automatically taken over when the USB peripheral is enabled.[1] This is the only peripheral that I am aware of that behaves this way and is a tiny detail hidden in the reference manual RM0008 about alternate configurations. You *do*, however, need to enable the clocks for GPIOA and the USB peripheral.