Pocket Tutor

Build a handheld version Of the Morse Code Tutor

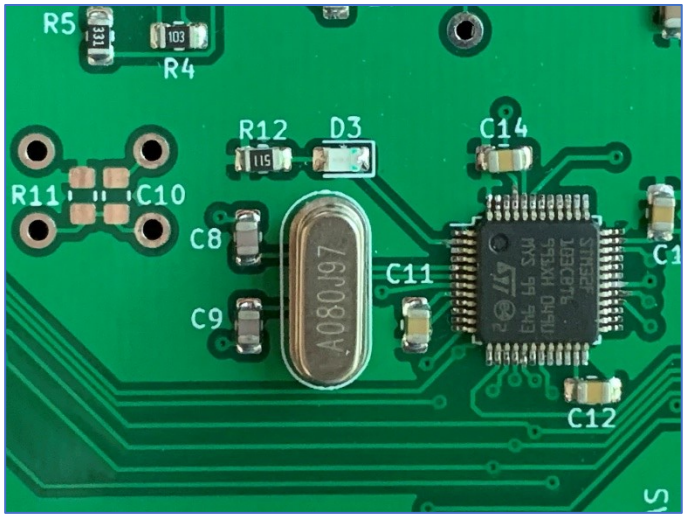
Part 3: The Microcontroller

Bruce E. Hall, W8BH

The original Morse Code Tutor uses the inexpensive and yet powerful “Blue Pill” microcontroller module. At a cost of only a few dollars, it has out-of-the-box functionality far superior to the Arduino and its clones. It has remarkable bang-for-the-buck.

And yet, the Blue Pill suffers a few disadvantages. First and foremost, quality control is lacking. Many units have poorly soldered components and non-functional USB ports. Second, the microcontroller is almost always a clone, and does not always have the exact functionality of the STM32 it is trying to emulate. The memory capacity of the device varies from lot to lot. And most importantly, emulated flash memory, used by Morse Code Tutor to store its settings, works on some modules but not others.

In this section we replace the Blue Pill with a genuine STM32 microcontroller. Yes, it costs a few dollars more and does not come assembled. But we overcome many of the disadvantages noted above. And, in the process of designing our own circuit, demystify the device. If you have ever thought about building your own microcontroller project, read on. *It is easier than you think.*

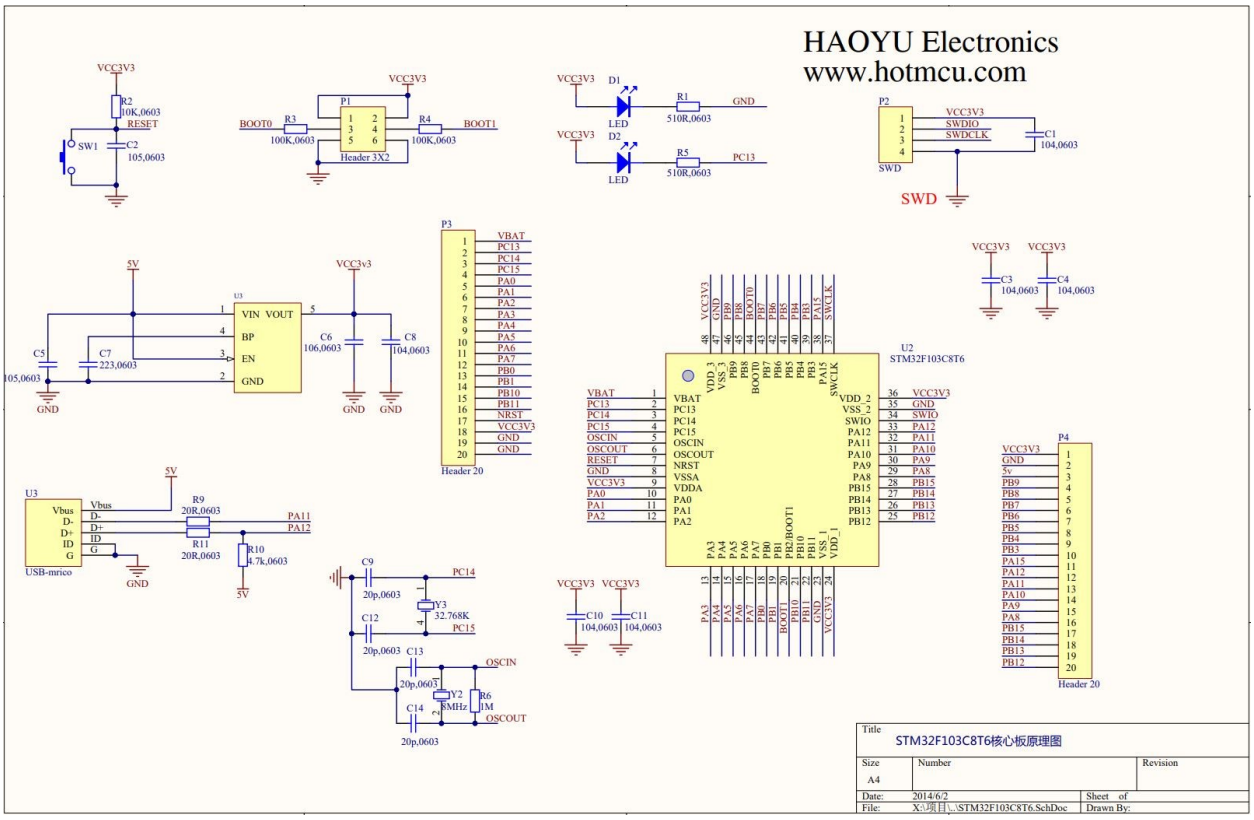


Here is a photograph of the microcontroller circuit on the Pocket Tutor. Note the oval crystal in the center. To the left of the crystal is the reset circuit, (R11, C10) which is not installed. Above the crystal is a diagnostic LED (D3) and its current limiting resistor (R12). To the right of the crystal is the microcontroller, surrounded by four bypass caps. No other components are required.

Easy-peasy!

The Blue Pill.

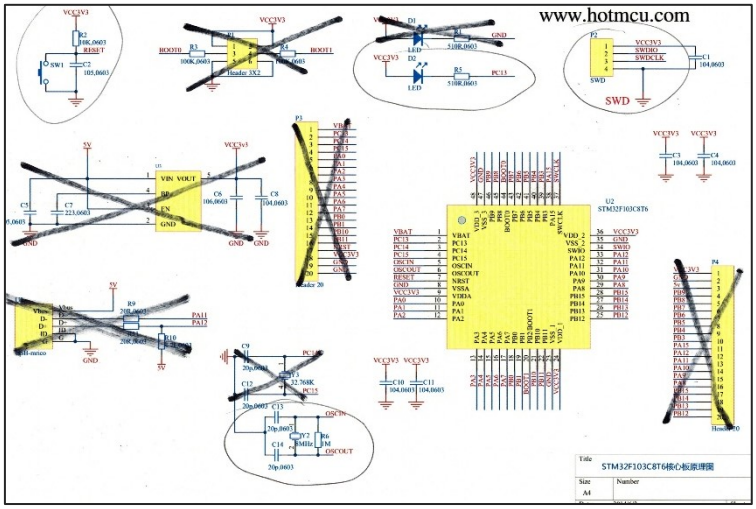
The “Blue Pill” became popular in 2017 as a clone of the Maple Mini, a microcontroller board designed by LeafLabs in 2011. The centerpiece of the Blue Pill module is the STM32F103C8T6 microcontroller by STMicroelectronics. In addition to the microcontroller, the Blue Pill board contains an 8 MHz crystal, a 32 kHz crystal, a voltage regulator, reset switch, and assorted resistors and caps. Here is the schematic.



Starting clockwise from top-left, there is a reset circuit, a programming header, two LEDS, some more headers, the microcontroller, two crystals with caps, the USB connector, and finally the voltage regulator with caps. A few months ago, I printed this schematic, circled what I needed, and crossed out what I did not need. Try it yourself!

Reset:	keep, but optional. See below.
Headers:	no need for headers, except for the SWD, which is used for programming.
LEDs:	keep one of them as a diagnostic tool.
uController:	keep, obviously.
Crystals:	keep only the 8 MHz one.
USB:	cross out.
Power supply:	cross out. We have our own circuit for that.

OK, what is left?



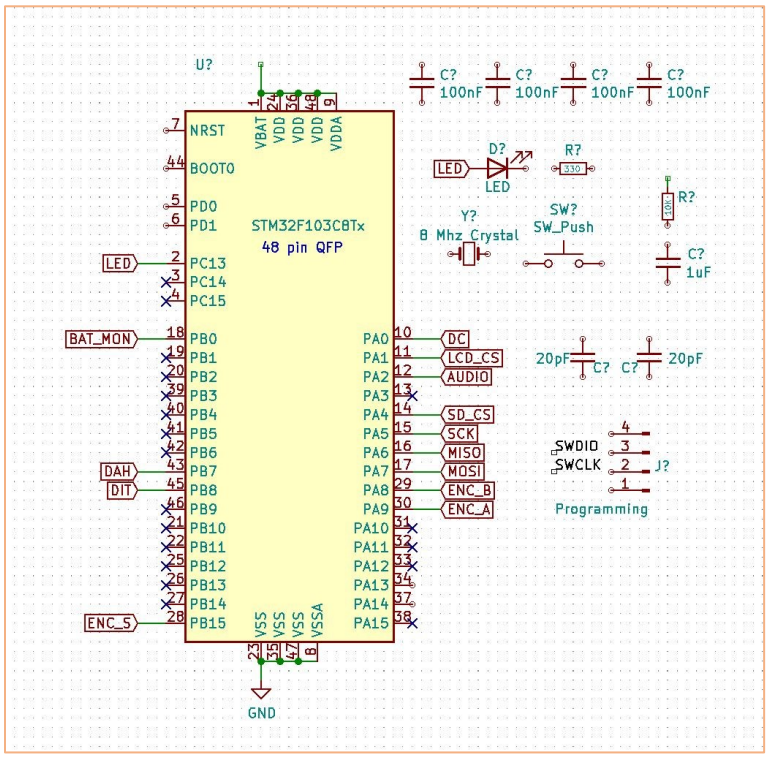
The remaining circuitry doesn't look as complicated.

The reset circuit, top-left, is just a switch and RC debouncer for resetting the microcontroller. If we accidentally put the controller in an infinite-loop, reset brings us back to the start of the application.

In our application, reset isn't necessary. If the device stops working, just cycle the power. I did not solder mine in. However, reset is helpful in the development phase of any project, so I kept it in the design.

Similarly, a diagnostic LED is not needed for our application, but useful for testing purposes. Keep.

And finally, the crystal. Strictly speaking, this is not needed either, since the STM32 microcontroller can run on its internal clock instead. However, the current Arduino environment assumes that you are using a Maple/Blue Pill with an external 8 MHz crystal. We need to include the crystal if we want to program in the Arduino IDE. The datasheet recommends using the crystal with 2 capacitors and a high-value resistor. Usually only the crystal is needed, but I am including both loading capacitors for better reliability. More on that below.

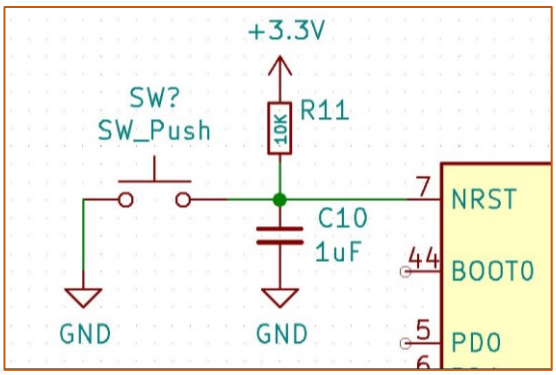


Microcontroller Components

We are left with 14 components: 3 components for reset, 2 for the LED, 1 programming header, 3 for the crystal, 4 bypass capacitors and the microcontroller itself.

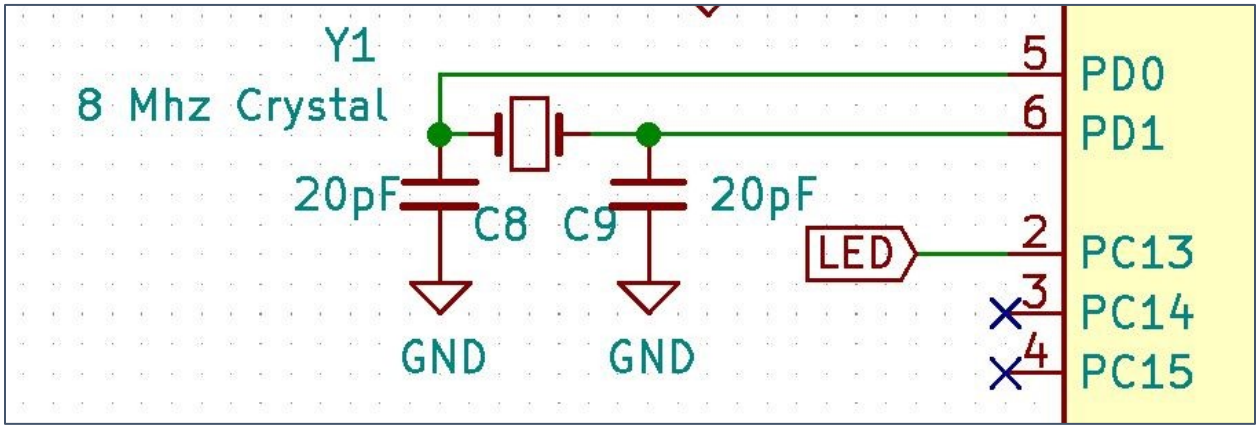
Time to start creating our own schematic. The graphic on the left shows all 14 components. For all of the passive components, I used the same values as found on the Blue Pill schematic. And I labelled the microcontroller pins according to how they are used in the Morse Code Tutor. Everything is ready to be connected.

The reset circuit is just a pushbutton that is debounced with a resistor and capacitor. The microcontrollers reset line, called "NRST", is normally held high by the resistor. When the pushbutton is pressed, it takes the line low. The capacitor keeps the line low until it is sufficiently recharged. The 10K resistor and 1 uF capacitor have a time constant of (10K)(1u) = 10 milliseconds, which is adequate to debounce mechanical switches that typically have bounce times of a few milliseconds or less. Connect the pushbutton, resistor, and capacitor to the NRST line, as shown.



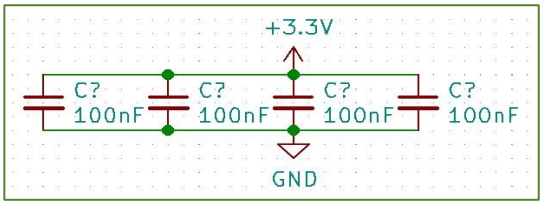
Reset Circuit

Next is the 8MHz crystal. The crystal is connected to our microcontroller on the PD0 and PD1 pins, which by default are the external oscillator pins. The crystal is "loaded" by adding capacitors from each leg of the crystal to ground. These capacitors are unique for each crystal, and determined by the crystal's Load Capacitance. For example, the datasheet for the ABL crystal I use specifies a load capacitance of 18pF. The actual load capacitance is given by the formula: CL = (C8*C9)/(C8+C9) + Cstray, where Cstray is estimated at 5 pF. Adafruit has a nice discussion about crystal loading here. By using 22pF capacitors, our estimated load capacitance is = (22*22)/(22+22)+5 = 16pF, which is acceptably



close. 27pF caps would work, too, and be even closer to the desired loading. Why did I use 20pF? I have a bunch of 20pF caps. Loading capacitance is not critical for our application. If timing must be precise and accurate, however, pay closer attention to this detail.

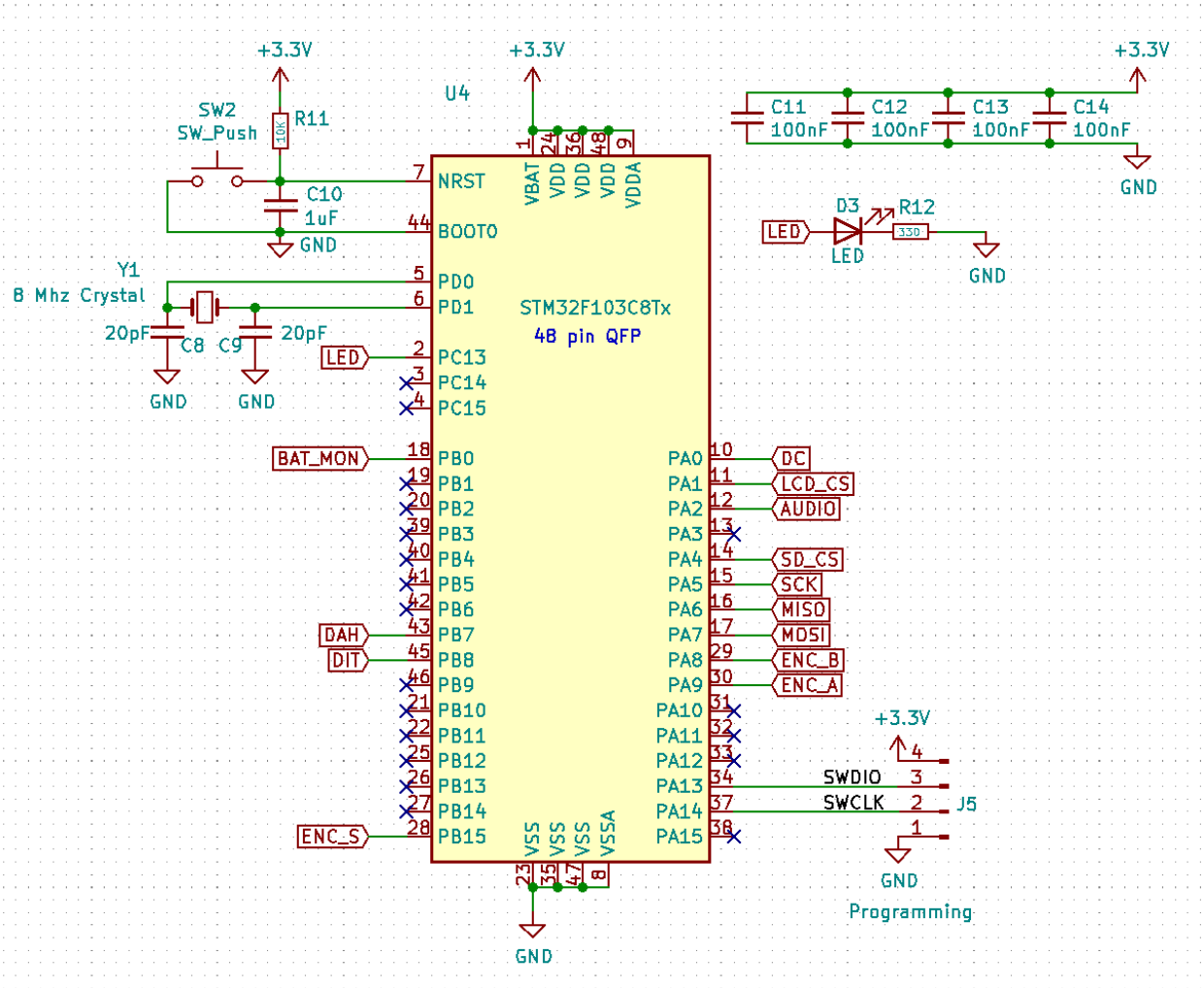
Next are the bypass capacitors. The datasheet for the STM32 recommends a bypass capacitor for every set of voltage pins on the microcontroller. These capacitors serve two useful functions. First, they function as energy reservoirs for the IC they serve, temporarily supplying power in the event of a transient voltage drop. Second, they filter high-frequency noise



Bypass Caps

generated by the IC and/or surrounding circuitry. Often these capacitors will be shown in some unused corner of the schematic sheet. And they might be shown lumped together in one circuit, as above. In reality, they should be physically placed as close as possible to the power pins. If you look carefully, you will see that there are 4 sets of power pins (VDD and VSS) on this controller. Each located on one side of the 4-sided device. Look back at the photograph one page 1, and notice that there is one bypass cap on each side of the microcontroller.

Finally, let's add the diagnostic LED and a programming header, which connects to the SWDIO/PA13 and SWCLK/PA14 pins of the microcontroller. Here is the final schematic:



The next part of the project is the audio circuit. Stay tuned.

73, Bruce.