

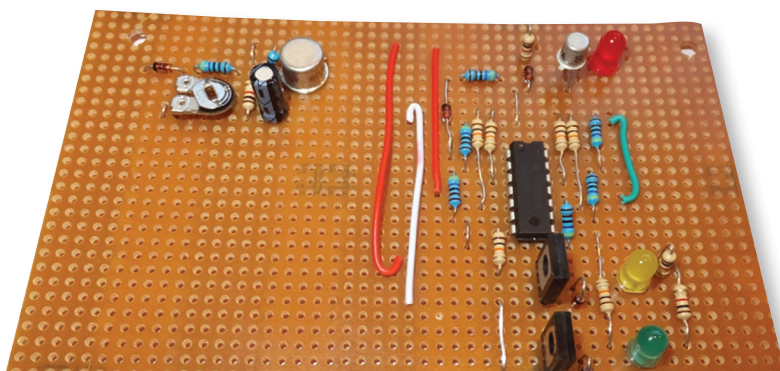


FIGURE 3: My 'bodger's build' analogue sequencer.

potential divider comprising the resistors R6 to R10, thus creating a different switch-on voltage for each one. The operational amplifiers are used as voltage comparators, and they turn their outputs on as the ramping voltage across C1 passes their switch-on voltages. The result is a sequenced turn-on of events 1 to 4. I have omitted the event-4 NPN transistor and associated components for clarity in Figure 1.

Each voltage comparator output uses an external NPN transistor which pulls that event output to ground when the event triggers. To switch a DC or other external voltage, you can simply add a FET-based switch which can be constructed using a device that will handle as much current and voltage as needed.

In my example, the FET-switch circuit diagram shown in Figure 2 will handle up to 100A with only a small voltage drop across the device. It's configured to switch 13.8V DC, but could be set up easily for other voltages. You turn on the FET switch by grounding the 'Enable' line. There is also a separate 'Disable' line you can use in your projects for an override to turn the switched rail off by other means (ie, a VSWR trip or other alarm condition). I found a nice supply of suitable FETs at the PiHut [2] when ordering some other bits and bobs.

My bodger's build of this kind of sequencer is shown in Figure 3. The operational amplifier is a quad device in a single package, and is a TL084 that I happened to have in my component store. My build only has three events to handle as that was all I needed, and I've used some beefier NPN transistors for the Events 2 and 3 outputs because that was all I had.

The ramping voltage on the comparator inputs is shown in Figure 4 which displays the voltage across C1 after the PTT line is grounded. If we also add in the Event 1 output, as shown in Figure 5 (cyan trace), you can see Event 1 output going high once the ramping voltage passes the switch-on value, and then off again as the voltage across the capacitor collapses once the PTT ground is removed. This is the signal applied to the base of the following NPN transistor, and will make the actual event output line go to ground.

A digital approach

The sequencing can also be performed by software running in a microcontroller. This would give very accurate control over the event timing, and allow configuration changes easily without modifying hardware. I've drawn a microcontroller-based sequencer (MCU) in Figure 6. The hardware doesn't actually change all that much: we are replacing the ramping voltage and op-amps with a MCU, but the external switching components remain the same. Of course, I used vibe coding to write the software from an English language specification to save all that mucking about writing and debugging code. I created the high-level and outline requirements written in Markdown format. You will find the document on the project GitHub [3] called 'PTT Sequencer.md'. I uploaded this document to ChatGPT, and issued the following prompt:

Acting as an arduino coding expert, analyse the attached outline requirements document and help me develop this into a detailed requirements specification. Ask any questions necessary for clarification before producing a detailed requirements specification. This project is for the Arduino Nano Every board and will be compiled using the Arduino IDE.

Chat GPT then asked me for clarification on ten specific points:

- 1. Pin allocation:** Are D2 for PTT and D4 to D7 for outputs acceptable, or do you need specific pins?
- 2. Reverse timing:** Should the reverse sequence use the same 45 ms delays, or do you want separate configurable release delays?
- 3. Immediate release action:** On PTT release, should Output 4 go HIGH immediately, followed by delays, as drafted above?
- 4. Mid-sequence behaviour:** Do you want the sequencer to reverse from its current state if PTT is released before the full sequence completes?
- 5. Rekey behaviour:** If PTT is pressed again while the reverse sequence is running, should the sequencer resume



FIGURE 4: The sequencer voltage ramp.

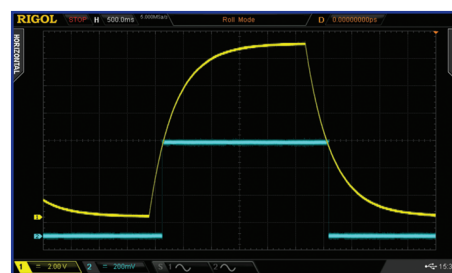


FIGURE 5: Event 1 oscilloscope capture.

forwards from its current state?

- 6. Electrical interface:** What will the four outputs drive: relay drivers, transistor inputs, RF relays, amplifier PTT, preamp power, bias, or something else?
- 7. Output naming:** Should the four outputs have functional names, for example Preamp, Relay, Amplifier, Transmitter, rather than Output 1 to Output 4?
- 8. Watchdog LED rate:** Is one flash per second acceptable?
- 9. Debounce:** Is a 10ms PTT debounce acceptable, or is the PTT line already clean and externally debounced?
- 10. Serial diagnostics:** Do you want optional serial debug output, or should the firmware be silent? My recommendation is silent by default, with a compile-time debug option.

My response was as follows as a single reply:

- Q1:** Pin allocation as you recommend is acceptable.
- Q2:** Reverse timing should be separately configurable.
- Q3:** On PTT release, output 4 should go HIGH immediately followed by delays as per spec.
- Q4:** If the PTT is released before the full sequence has completed it should reverse from its current state.
- Q5:** If PTT is pressed again while the reverse sequence is running, the sequencer should resume forwards from its current state.
- Q6:** The outputs will drive NPN transistors to in turn control FET switches.
- Q7:** Outputs should be called Event1,

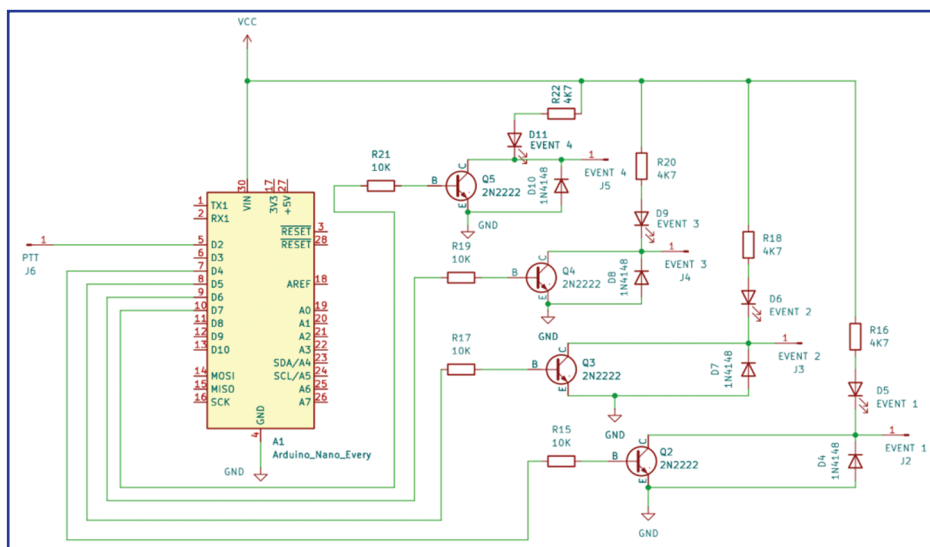


FIGURE 6: MCU-based sequencer circuit diagram.



FIGURE 7: Oscilloscope capture of PTT turning on.



FIGURE 8: Oscilloscope capture of PTT turning off.

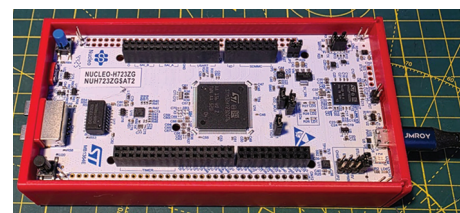


FIGURE 9: Nucleo 144 board in a 3D-printed case.

Event2, Event3 and Event4.

Q8: 1hz flash is perfect.

Q9: 10ms debounce is acceptable but should be configurable.

Q10: Optional serial diagnostics as recommended.

I now have a very detailed requirements document produced by the large language model in markdown format which is on the GitHub for your reference called 'Requirements Specification Full.md'. From my initial document, which contains seven sentences giving an outline description of my conceptual software-based sequencer, I progressed to a full and detailed requirements specification, including test cases, in under 15 minutes. I then issued the prompt:

Produce full code from the final requirements.

And, with a quick copy-and-paste, I now have the software compiled and running on my Nano Every. This took a total of 25 minutes. I have uploaded the code for your reference to the GitHub.

With the Nano Every hooked up to my oscilloscope, I can clearly see the sequencing happening to my specification. The outputs from Event 1 (yellow), Event 2 (cyan), Event 3 (purple) and Event 4 (blue) are shown in Figure 7. The events are occurring exactly to specification at 45ms intervals, and the digital outputs from the MCU are going high to 5V (because the Nano Every is a 5V device). When I remove the grounding of the PTT the reverse occurs as shown in Figure 8. Given that this sequencing is all done in software, there are some simple constants defined in the code that dictate the time between each event in both the PTT-on and PTT-off sequence. It's fully flexible and configurable, and it allows asymmetrical, per-event timing. I don't know how I would create that kind of flexibility in an analogue solution.

I have executed my test cases from the detailed specification document and found no issues with my software teams' initial deliverable; you really can't fault it. This vibe-coded project, with suitable input and guidance, is another excellent example of how modern tools and techniques can drastically accelerate home-brewing for the radio amateur.

Final project musings

It is always essential to ensure that the necessary timing for any hardware you use is adhered to so that hot-switching is avoided. That can be catastrophic; please don't ask how I know this! If you are using coaxial relays, make sure you check the switching and settlement time for *your* hardware. Also, with a project like this we need to be sure we don't let our microcontroller ruin our radio reception. When building the MCU sequencer, make sure the MCU is screened, use decoupling capacitors close to the power pins, and pass your external PTT and line connections through feed-through capacitors or ferrite beads to choke off MCU clock noise. You could also consider optocouplers (like the classic 4N25) instead of standard NPN transistors for the PTT input and external event trigger lines, as this would add a bulletproof layer of protection against ground loops and stray RF making its way into the logic board.

A simple vibe-coding example

I am now collecting ST Microelectronics Nucleo boards. I recently obtained a Nucleo-H723ZG board which is one of the largest 144-pin types available. It contains the STM32H723 MCU, an on-board ST Link programmer, and access to the majority of the MCU pins via on-board headers. As a quick test of programming from the Arduino

IDE, I asked ChatGPT to create me a sketch to utilise the three on-board LEDs and user button with the following prompt:

Acting as an arduino coding expert, write a demo sketch for the Nucleo-144 H723ZG board for the Arduino IDE to utilise the 3 on-board LEDs and user push switch. Sequence the LEDs and at each press of the user button, alter the sequencing speed.

The LLM produced a clean Arduino demo sketch for the STM32 Arduino core targeting the stock NUCLEO-H723ZG, and explained its assumptions and sources for the pin mapping used in the code for the LEDs and Push Button. Less than five minutes elapsed from taking the board out of its box to having a demo sketch running. Pretty good! I've put the code on the column GitHub at [4] for you if you want to take a look. I also found a really neat 3D print design for a slide-in tray for the board, shown in Figure 9. Excellent!

References

- [1] ssb.de/shop/en/radio-electronics/sequencer
- [2] thepihut.com/
- [3] github.com/RSGB-EC/Vibe-Sequencer
- [4] github.com/RSGB-EC/Simple-Vibe-Coding-Example