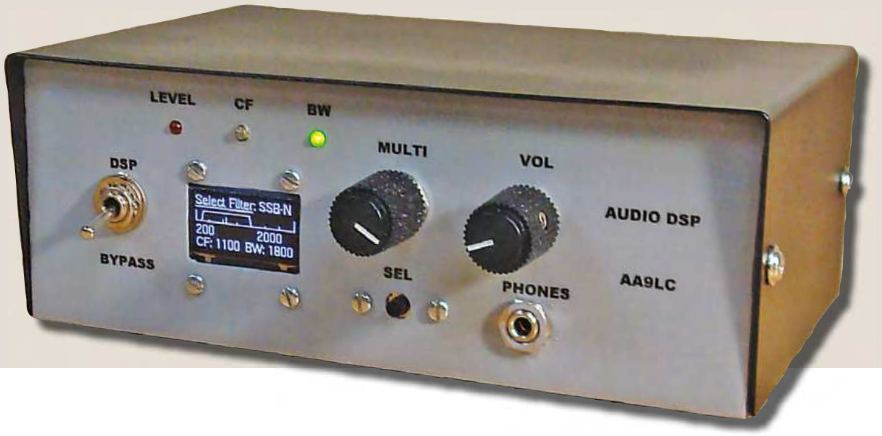
An Improved DSP Audio Filter

This add-on audio filter stores adjustable parameters for seven different profiles.



Grant Zehr, AA9LC, and Scott Zehr, K9GKC

The DSP filter described here is based on the SOTABEAMS LASERBEAM-VARI module (available from DX Engineering) and offers performance like that described by Steve Allen, KZ4TN, in his article, "Build a DSP Audio Filter," from the November 2020 issue of *QST*. However, this DSP filter adds a more flexible user interface. With an Arduino Nano micro-controller and a 0.96-inch × 128-pixel × 64-pixel OLED, this unit can show the bandwidth (BW), center frequency (CF), high and low filter cutoff frequencies, and a graphical representation of the filter profile (see the lead photo). Parameters for seven filters are stored in memory and can be recalled during operation. One memory is user programmable and will save your favorite settings. The existing filter parameters can be modified by editing the Arduino sketch that controls this unit.

This design is based on the Arduino controlled filter designed by Dennis Zabawa, KG4RUL, and available on the SOTABEAMS website at **www.sotabeams.co.uk/variable-bandwidth-filter-modules-ssb-cw**. We borrowed heavily from the code included in his sketch, adding a graphic display and the ability to store filters.

Construction

Construction is simplified because many of the needed parts are included with the purchase of the DSP module. R1 – R6, D1 – D3, C1 – C3, and the rotary encoder (with switch) are provided (see Figure 1). We purchased a 5 × 7-centimeter proto board and fitted it with sockets to install the Arduino Nano and the LASERBEAM-VARI. The small board functions as a motherboard for the project (see Figure 2). We used point-to-point wiring on the motherboard (see Figure 3) to connect the Arduino Nano and the DSP module. R1 – R6 and C1 – C3 are mounted on the motherboard. Instead of including SW-3, as shown in Figure 1, we installed a header on the motherboard to allow access to DSP Pin RB-12, if needed. Normally, this filter operates with the DSP gain set to one (DSP Pin RB-12 not grounded). The OLED, LEDs, rotary encoder, audio gain control, headphone jack, and DSP/bypass switch are mounted on the front panel. Although the rotary encoder provided with the LASERBEAM-VARI includes a pushbutton switch, we also included a light-touch pushbutton switch (SW5) to keep the small case from sliding away as the **DSP MODE SELECT** (**SEL**) switch is pushed. The power switch, power connector, audio in and audio out jacks are mounted on the rear panel. The motherboard is mounted vertically inside the case using small right-angle brackets (see Figure 4).

We recommend using 0.1-inch headers and connectors to simplify removal of the motherboard, Arduino, and DSP module when needed. We used an audio amplifier kit featuring an LM386 chip (U2 in Figure 1) and mounted it inside the case. A simple 7 V regulator circuit provides a stable supply for the unit.

"Parameters for seven filters are stored in memory and can be recalled during operation."

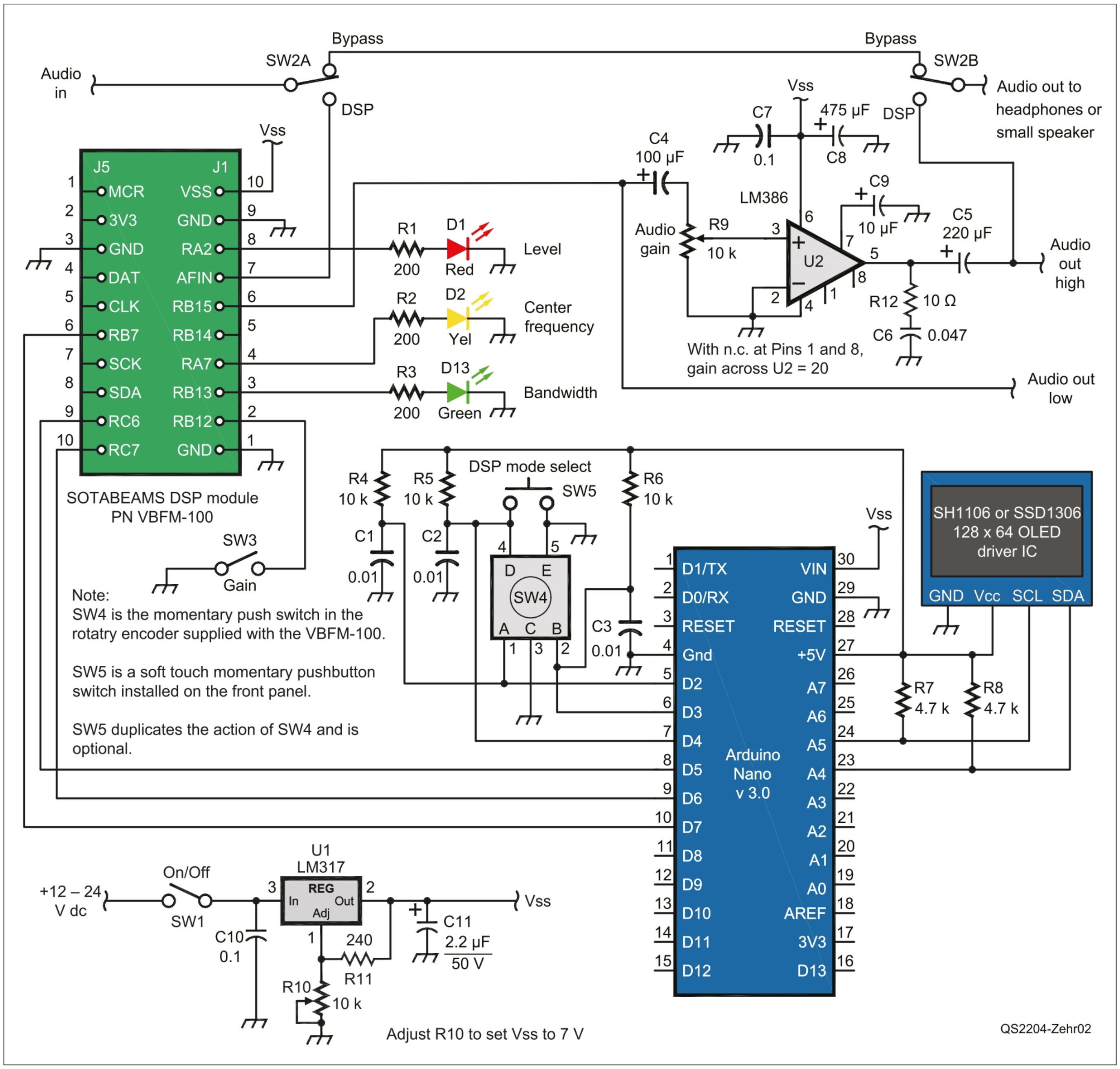


Figure 1 — A schematic diagram of DSP audio filter.

This filter uses an Arduino Nano version 3.0 or compatible. Check the pin numbers on your board carefully. Some versions of the Nano place the pins at different locations on the board. If there is a different arrangement on your Nano, rely on the function label of the pin (e.g. "D5") rather the pin number. We had good luck with Arduino-compatible microcontroller boards from eBay, but to guarantee compatibility, you may wish to purchase an official Arduino Nano V 3.0 board.

There are at least two 0.96-inch × 128-pixel × 64-pixel OLED devices available at the time of this writing. We used the OLED with the SSD1306 driver integrated circuit (IC). However, the sketch can be easily modified to allow use of an OLED with an SH1106 driver IC. With the SH1106 OLED, you must edit the sketch to select the **DEFINE** statement for the SH1106 OLED and disable the **DEFINE** statement for the SSD1306. Be sure the display you select uses the I2C interface. The OLED should have four pins (GND, VCC, SCL, SDA) and be able to operate at VCC of 5 V. We used a discarded computer switch box to house this project.

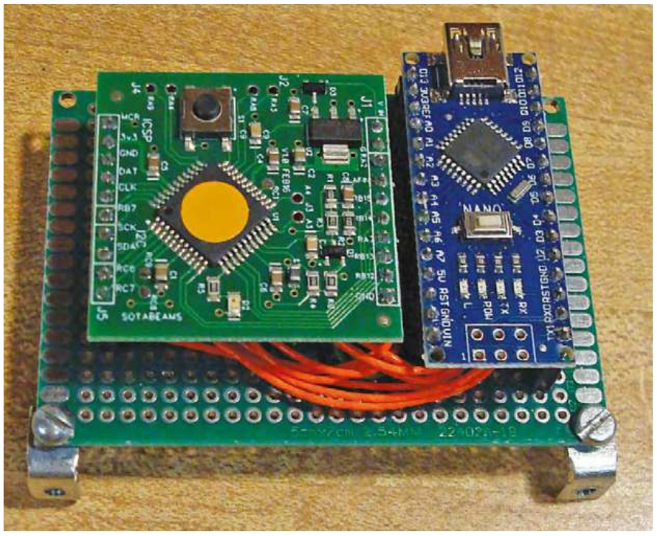


Figure 2 — A VBFM-100 module and Arduino Nano are mounted on a small motherboard.

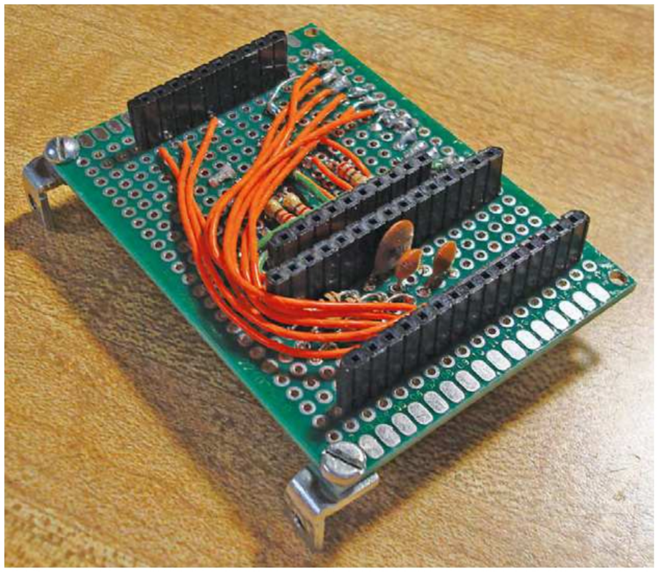


Figure 3 — Point-to-point wiring connects the Nano and the DSP module.

Programming the Arduino Nano

The sketch for this project is on the *QST* in Depth web page at **www.arrl.org/qst-in-depth**. The installation to your Nano should be routine. You will also need the U8glib.h library for text and graphics and the EEPROM.h library to store and retrieve data from memory. The EEPROM.h library is included in your Arduino IDE download, but it must be installed before you can use it. The U8glib.h library is available at the Arduino site at **www.arduino.cc/reference/en/libraries/u8glib**. For help downloading and installing libraries, go to **https://www.arduino.cc/en/Guide/Libraries**. There are many books that explain Arduino programming in more detail.[1, 2, 3, 4]

Table 1 — Seven filters are available.

Filter	BW	CF	Low cutoff	High cutoff
Factory Default	2400	1500	300	2700
CW – Narrow	300	800	650	950
CW – Medium	500	800	550	1050
Memory 1	(BW)*	(CF)*	–	–
SSB – Narrow	1800	1250	350	2150
SSB – Medium	2100	1300	250	2350
SSB – Wide	2400	1400	200	2600

*bandwidth and center frequency are user programmable.

Operation

When first powered up, the OLED will display a splash screen for a few seconds, then you should see a graphic similar to the one in the lead photo. The last filter used will be displayed along with any changes to the BW or CF that you made the last time that you used the unit. You will be in Select Filter mode. At this point, turning the multifunction knob (rotary encoder) will select a new filter. There are seven choices (see Table 1).

Any changes you make with the Memory 1 filter will be saved and restored at the next power-up. The other filters revert to their original values when you select another filter. The filter name will be displayed along with the high and low cutoff frequencies, CF, and BW. Pressing the DSP mode select switch (**SW4** or **SW5**) will change the filter to Adjust Filter – Bandwidth mode. In this mode, turning the multifunction knob changes the BW of the filter you have selected. The change that you make is symmetrical, changing both high and low cutoff frequencies until you approach the filter limits (200 Hz at the low end and 3,500 Hz at the high end). Pressing the DSP mode select switch again advances you to the Adjust Filter – Center Frequency mode. Turning the multifunction knob now changes the filter's center frequency. The high and low cutoff frequencies and the center frequency will again be displayed. Press the DSP mode select switch again, and you return to the Select Filter mode with the options described above. As the DSP mode select switch is pressed, the front panel LEDs also indicate the active (DSP) mode.

This sketch displays three modes using two DSP LEDs. In the bandwidth mode, the BW LED (green) is illuminated. In center frequency mode, the CF LED (yellow) is illuminated. In the Select Filter mode, the BW LED (green) is illuminated while the BW parameter of the filter is being loaded and during normal operation. The CF LED (yellow) is illuminated while the CF parameter of the filter is being loaded, which makes the CF LED appear to flash each time a new filter is selected. There are examples available at **www.arrl.org/qst-in-depth**.

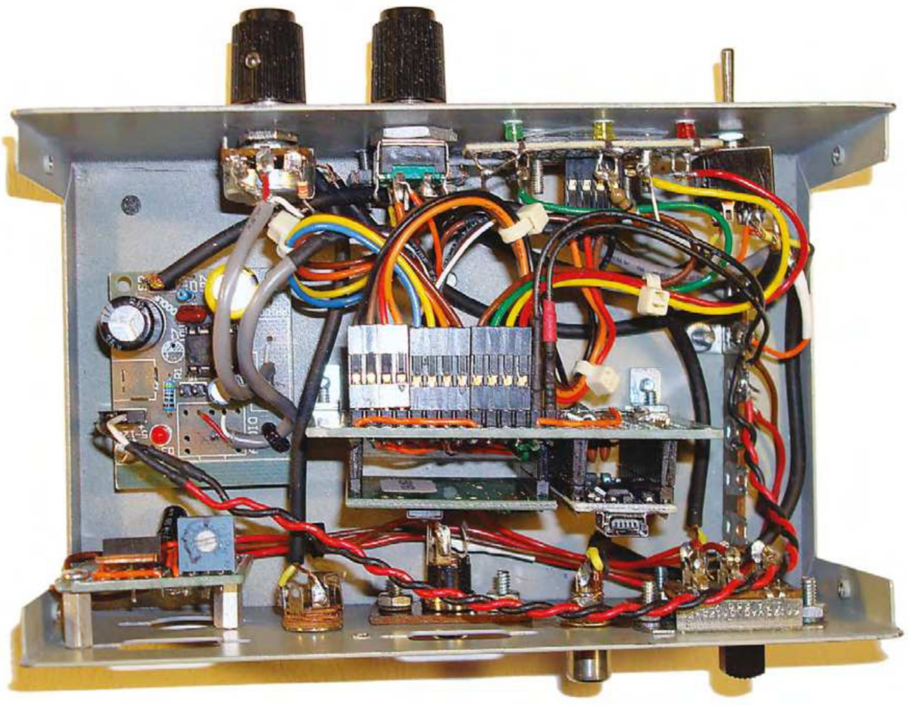


Figure 4 — The motherboard is mounted vertically inside a small chassis.

Storing Power-Up Settings

One feature of the VBFM-100 DSP module deserves special mention. The DSP board includes a pushbutton switch that allows the user to save a set of power up filter parameters (CF, BW, and filter mode). Any time the module is powered up, pressing the switch will save the values currently in use. With this design, however, the Arduino assumes that the factory default settings are stored in the DSP. If alternate values are stored in the DSP, then erroneous data will be displayed. In short, avoid pressing the black button on the DSP module while power is applied. There is a recovery procedure that can be used to return your DSP to factory default settings, but it is not simple. There is no problem pressing the button while power is off, but you should be careful during testing and operation of the unit.

Summary

This high-performance filter works well with almost any amateur transceiver because it plugs directly into the speaker or headphone jack. It may not add much to the performance of a modern top-end DSP receiver, but it can improve the performance of many homebrew rigs, middle-aged commercial transceivers, vintage radios, and less-expensive commercial gear currently available.

Notes

[1]G. Popiel, KW5GP, *More Arduino Projects for Ham Radio*, *ARRL*, 2017.
[2]S. Monk, *30 Arduino Projects for the Evil Genius*, McGraw-Hill, 2010.
[3]S. Monk, *Programming Arduino, Next Steps,* McGraw-Hill, 2014.
[4]J. Purdum, W8TEE, D. Kidder, W6DQ, *Arduino Projects for Amateur Radio*, McGraw-Hill, 2015.

Grant Zehr, AA9LC, is a retired family physician living in Central Illinois. He has been licensed since 1966 and enjoys HF DXing, satellite operating, and antenna design. along with construction of equipment for the home station. You can reach Grant at **aa9lc@arrl.net**.

Scott Zehr, K9GKC, was first licensed in 1956 as a Novice, KN9GKC, while in high school. He worked in the electronics industry for 43 years, and in retirement continues to enjoy the operation, design, and bench work of ham radio. You can reach Scott at **k9gkc@arrl.net**.

For updates to this article, see the *QST* Feedback page at www.arrl.org/feedback.