Si5351a with quadrature output

VFO controller for Arduino

by WA5BDU

Overview

V1.9

This document describes my Arduino Si5351a controller program for the Arduino. I started with some simple demo code from Hans Summers G0UPL which took a frequency, calculated registers, and sent them to the chip. I later added a feature Hans described at Dayton/FDIM which puts the outputs of CLK0 and CLK1 on the same frequency but 90 degrees apart. This quadrature output local oscillator is a key component in phasing type radios.

After getting the above working I began to expand my software to add features necessary to meet my needs for a VFO to be used in a transceiver project. Those features include:

- A keyed line input, so the controller can swap between TX and RX frequency outputs as required as the user keys and un-keys the transmitter.
- A TX_Out output pin echoes the keyed line input, but doesn't change state until the TX frequency registers have been sent, and changes back to key up before the RX frequency registers are sent.
- An LCD display to show the frequency I'm on, menu options and so on.
- A user settable CW Pitch value which is the amount that the VFO shifts between TX and RX states if in CW mode.
- CW/Phone mode selection which functions to enable or disable the CW offset shift when the key line is closed.
- LSB/USB selection. This chooses the direction of CW offset in CW mode, so the user can listen to a station on either side of zero beat. In CW it's sometimes called NORMAL and REVERSE operation. There is also an output pin SB_Relay than cab be used control a sideband select relay in the receiver. Also, the phase difference between the outputs alternates between 90° and 270° when this selection is made.
- RIT control. This provides for separate TX and RX frequencies, with both displayed. The receiver can be tuned without disturbing the transmit frequency. RX and TX frequencies can be swapped and the offset can be zeroed without turning off RIT.
- Frequency step selection. There are both a menu for selecting from seven step sizes and a quick pushbutton selected step change between fine (10 Hz) and fast (100 Hz).
- Band selection with all ham bands from 3.5 to 144 MHz selectable.
- A Save State menu option which writes most of the current parameters into EEPROM so the VFO will start in the same state the next time it is powered on.

- Three miniature pushbuttons allow quick selection of often used functions while a menu controlled by the rotary encoder is used to access other functions. The pushbuttons operate with the familiar TAP and HOLD logic, giving two uses for each.
- A sidetone which sounds when the key is closed. A menu option can enable or defeat this function.

Details of operation

The pushbuttons offer six actions, one of which opens a menu with additional actions. A brief press of a pushbutton is called a TAP and a long press (> 0.5 second) is a HOLD operation.

There is audio feedback in the form of a short beep immediately as a switch is closed, followed by two short beeps after it has been held long enough for the HOLD function.

Pushbutton actions:

PB1 Tap	RIT ON/Off (toggle)
PB1Tap*2	Clear RIT if ON
PB1 Hold	TX/RX Swap (toggle)
PB2 Tap	Step 10/100 toggle
PB2 Hold	Band select
PB3 Tap	USB/LSB toggle
PB3 Hold	Main menu

RIT Operation

RIT is turned On and Off with a TAP of PB1. When enabled, the bottom line of the display shows the transmit frequency (marked with T) and the upper line shows the receive frequency (marked with R). The encoder now adjusts the receive frequency only. RIT can be used to allow fine tuning of the station you are working without moving your transmit frequency.

If RIT is turned off and then back on, the previous offset is restored. RIT is cleared if the band is changed.

RIT is turned off with a TAP of PB1, but it can be cleared to zero offset while remaining enabled with a double TAP (two TAPs in quick succession).

Finally, when RIT is enabled, a HOLD of PB1 will **swap** the transmit and receive frequencies. This function primarily works to set up split operation, when working DX for example. It works only if the RIT is on. As an example, you hear a DX station on 7010 kHz said to be listening up 2. Turn on the RIT and tune the receiver up 2, or possibly to the offset where you hear other stations working the DX. Now your TX frequency is 7010 kHz and your RX frequency is approximately 7012 kHz. Activate the RX/TX Swap function you will be transmitting on 7012 kHz and listening on 7010 kHz.

Occasionally while trying to work the DX, you may activate the Swap function again, tune around to find where the DX seems to be listening, then activate the Swap again to return to operation with the adjusted split.

Quick tuning rate change

A TAP of PB2 causes the VFO to alternate between slow (10 Hz / step) and fast (100 Hz / step) tuning rates. Other rates are available from the menu. If you select another rate and tap PB2, you will be back in the 10 Hz and 100 Hz step realm.

Band select

HOLD PB2 to enter the band selection screen. Rotate the encoder to cycle through eleven ham bands 80 through 2 meters. Tap PB2 when the desired band is displayed. The band starting frequencies are hard coded in the software. However, if you change bands, the stored frequency for the band you are leaving becomes the current stored frequency. That way when you return to that band, you are on the frequency you left.

Bands are a convenient way to navigate among ham bands. Use of the appropriate step size and the rotary encoder can be an alternative method to go quickly to a desired frequency area.

USB/LSB toggle

A TAP of PB3 causes the VFO to alternate between the USB and LSB modes. The change changes the direction of offset while in CW mode. In CW mode, LSB and USB are often referred to as Normal and Reverse and in the display are tagged as CWL and CWU. The benefit in CW is the ability to listen to either side of zero beat.

Also, some phasing receivers use a DPDT relay two switch sidebands. The SB_Relay output pin can be used to control this relay. It is configured to be HIGH on LSB and LOW on USB.

MENU:

A HOLD of PB3 enters the menu mode. After a brief message "STEP SIZE" is displayed. Rotate the encoder to choose **STEP SIZE, CW PITCH, CW/PHONE, SAVE STATE, SIDETONE** or **EXIT**. In general, TAP PB3 again to select the desired item. Some of them have sub-menu choices as described below.

STEP SIZE: Rotate encoder to cycle through steps from 1 Hz through 1 MHz in decade intervals. TAP PB3 when the desired step value is shown. The step size is shown at the right of the top line on the LCD. Step sizes are displayed as: 1, 10, 100, 1k, 10k, .1M, 1M

CW PITCH: This setting determines the amount of shift from the displayed frequency being used while in CW mode. For example, if the displayed frequency is 7025.000 kHz and the shift is 500 Hz while in CWL mode, the actual output frequency will be 7025.500 kHz.

The pitch is displayed and varied with the rotary encoder. Note that the VFO output frequency also moves so the effect can be heard if the VFO is in use in a receiver. TAP PB3 again to accept the new pitch. Note that limits of 150 Hz to 1000 Hz are enforced.

CW/Phone: This function is a "toggle". Pressing PB3 swaps to the alternate state. Indication to the user is via the LSB/USB tags in Phone mode which change to CWL and CWU in CW mode. The functional effect is that no CW pitch offset is used in the Phone mode while the CW pitch is used as an offset in CW mode.

SAVE STATE: This menu option saves your current VFO frequency, step size, band selection, CW pitch, CW/Phone mode, sidetone On/Off to EEPROM. The next time you power up, it will start with the settings that existed when you did your last SAVE. This is what modern transceivers do when you press the power off button.

After writing the current parameters to EEPROM, the program sounds two tones, low and then high, to indicate the writing has taken place.

When writing the data to EEPROM, a flag byte is written with a specific value. On startup, if the value is correct it is assumed that the EEPROM fields contain good data and they are loaded.

Also on startup, if the program finds that the flag byte doesn't find the expected value, it performs a write of the current (startup) parameters. This would normally happen only once – when the program has been installed and executed for the first time.

SIDETONE: The program has the option of generating a sidetone any time the keyed line input is closed. The pitch is the same as that set in CW pitch for the CW offset value. The SIDETONE menu option toggles the sidetone function between the On and Off states. TAP PB3 when SIDETONE is displayed. The display will briefly show ON or OFF to indicate the new state. If the state is ON, there will also be a brief beep sound.

EXIT: If it is desired to leave the menu without making any changes, TAP PB3 while the EXIT option is displayed.

A times four (X4) frequency output option:

Some receivers have a divide-by-four logic arrangement in front of a circuit that develops I & Q signals. Since this VFO provides I & Q outputs already, there's generally no need for an output of four times the indicated frequency.

However, one user wanted to use this VFO with his existing hardware which needed the X4 signal. So I've added it as an option in V1.2.

There is a logical flag that tells the software on startup whether to run in the normal (X1) or four times (X4) modes. The user can edit the source code to change that flag as desired. The default is for X1 mode. The line to edit is this one:

```
boolean X4 = false; // Output freq is 4 times indicated. Normally FALSE
```

To switch to X4 mode, change 'false' to 'true'.

But there's another way to get there if you don't want to make the change permanent. Just hold in any button while starting up and the software will swap to the opposite mode for the current session only.

At startup, the LCD will remind you of what mode you are in with a brief message.

Output level:

The Si5351a allows selection of four different output levels. This software selects the highest level. With my oscilloscope, I measured these levels into 50 ohms:

0.77 Vpp, 1.35 Vpp, 1.83 Vpp, 2.09 Vpp

If you would like to measure or otherwise experiment with the different levels, there's a disabled routine that can be turned on by removing the comment marks:

```
// do_levels(); // 9/13/2020 - experiment with levels settings
```

Find this line and remove the two forward slashes at the beginning to enable the routine. Then the VFO will start up at the lowest level and with each press of a button advance to the next level and display its value (2 mA, 4 mA, 6, mA, 8 mA). After the highest level is shown, the next press returns to normal operation.

Hardware details:

The Si5351a is a 3.3 V IC controlled by two data lines. Rather than use a 3.3 V Arduino, I used the 5 V version, and my first version of the hardware used a TXS0102 logic level converter.

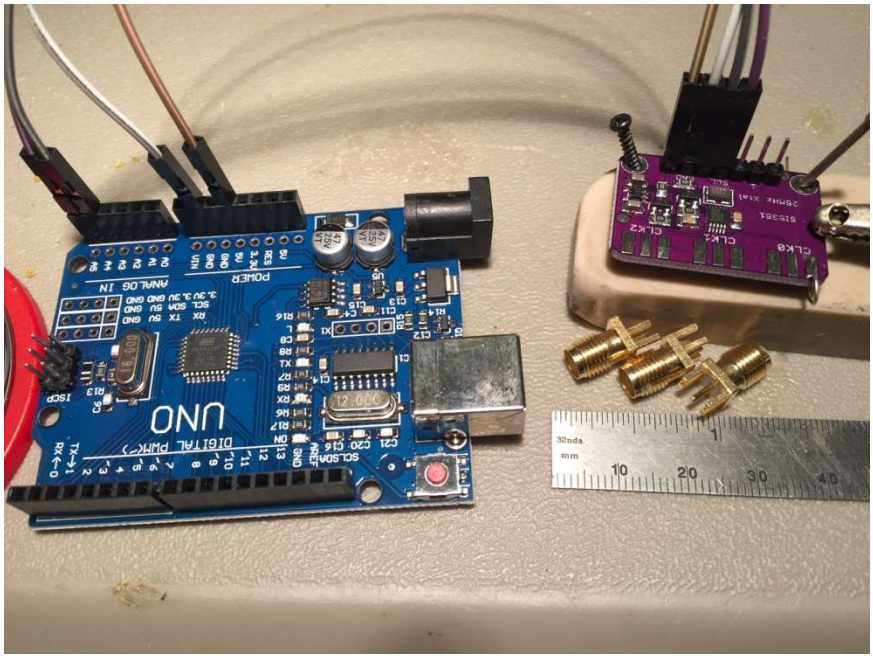
However, since then complete little Si5351a modules have appeared which aren't much bigger than a postage stamp and include a 3.3 V regulator plus 5V to 3.3V logic level converters and the tiny chips are all soldered in. They're also quite inexpensive.

I tried one with my Arduino program and was able to demo the system with just four interconnecting wires between the Arduino and the synthesizer board:

- A4 to SDA
- A5 to SCL
- GND to GND
- 5 V to 5 V in

It worked. Of course I have no rotary encoder, pushbuttons or display but it came up on its starting frequency of 7025000 kHz and I see CLK0 and CLK1 90 degrees apart on my scope. The USB port powers it fine without the added load of the display, etc.

Below is a photo of the Arduino and Si5351a module with the minimum wires required for programming a frequency. RF output can be taken from the CLK0 & CLK1 pads or from pins in the 0.1" header. Note that I often use the much smaller Arduino Nano but here I'm using the Uno which is easier to mount and has pre-installed 0.1" female headers and a power connector.



The **rotary encoder** is a Bourns PEC11L-4020F-S0020 encoder with switch from Mouser. No external pull-ups or filtering are used. Internal pull-ups are activated in software. I'm not currently using the switch.

Pushbuttons PB1 and PB2 are wired to logic inputs and ground those inputs when the switches are depressed. PB3 has diodes connected from PB1 and PB2 to itself such that pushing PB3 grounds both the PB1 and PB2 logic lines, a condition defined as "PB3". In other words, I'm getting three pushbutton inputs from two logic lines. No physical pull-ups are used – pull-ups are activated in software on the Arduino.

A miniature **speaker or sounder** is connected to D9 for alerts and an optional sidetone. I used a CEM1201(50) sounder from Digikey. Since it only has a 50 ohm resistance and Arduino pins should be kept below 40 mA output, I connected a 200 Ω resistor in series with the speaker. A capacitor of 0.4 uF to 1.0 uF or so could also be used.

Of course the user could route the audio into a receiver chain instead of using a speaker within the VFO.

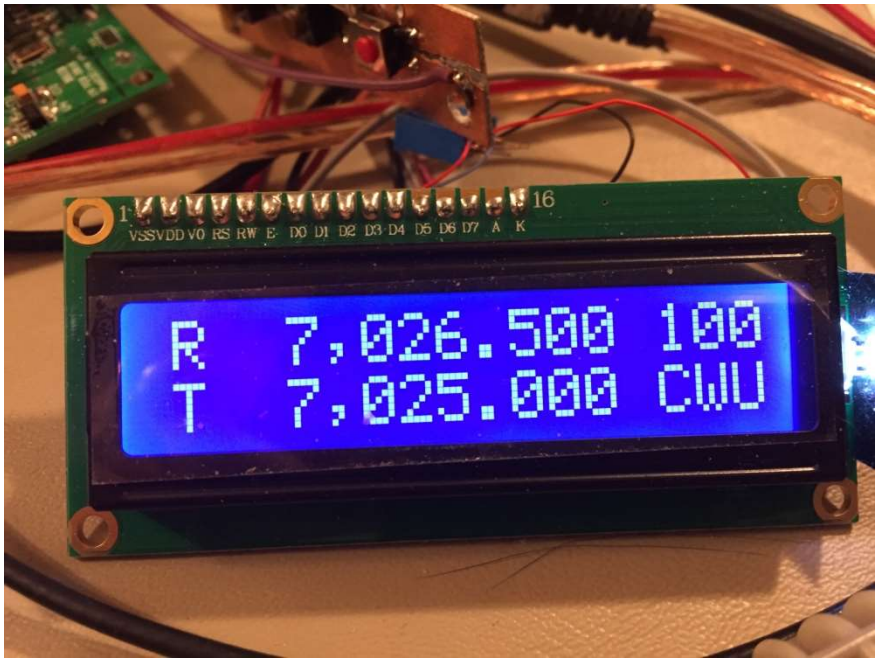
A **reset pushbutton** is optional but handy. I installed one on the rear of the cabinet. You can achieve the same effect by cycling power though. Or press the button on the Arduino, if it is accessible.

Key down or PTT input is provided so the software can shift the frequency by the amount of CW pitch, in the correct direction for USB or LSB receive. It also allows RIT operation. This input uses the D8 line and is active (transmit mode) when LOW.

A **TX_Out** pin echoes the Key Down input but doesn't change state until the TX frequency registers have been sent. Registers can be updated in a couple of milliseconds, but this feature can prevent keying the transmitter while the frequency is changing. It also changes back to its key up state *before* the RX registers are sent.

The **LCD display** I'm using is a somewhat generic 2x16 character display with Hitachi interface and a backlight LED. A photo is shown below. The pin ID's seen in the schematic diagram and wire list are visible.

In the photo, RIT is ON, so both receive and transmit frequencies are displayed. The 100 Hz step size is in the upper right and mode CW, upper sideband is at the lower right.



Power source

While I have powered the Arduino plus Si5351a board from a USB port, adding the LCD display and possibly other loads requires that an external power supply be used. The Arduino can accept either a 5 V external power source or an unregulated 8 to 12 VDC source. It has a 5 V regulator on board and can supply the 5 V needed by the display and the Si53551a board.

Coupling RF output

The output of the Si5351a is a square wave going from about 0 V to 3.3 V. As such, it has a DC level of 1.65 VDC. If you are using the output to drive a logic chip, this may be appropriate. In other instances, you may want to add 0.1 uF ceramic capacitors in series with the outputs to block the DC level.

Calibration

The Si5351a module will typically have a 25 MHz or 27 MHz crystal timebase. The actual frequency will likely be off a bit and you can correct for it in the source code for your specific device.

I determined my actual crystal frequency like this:
Listening on my K3 with the Si5351a set at 7025000 Hz I read 7024816 Hz. So my actual frequency is 25,000,000 * 7,024,816 / 7,025,000 which gives me 24,999,345. I changed the appropriate line in the source code to this:

#define XTAL_FREQ 24999345 // my specific Si5351a board

Please note that while in the CW mode, the output will be shifted by the value of CW_pitch, so you should take your reading in the “phone” mode or have the key closed while reading the frequency.

Of course I could have used a more accurate standard than the K3 if I'd wanted greater accuracy.

Wiring details

Most interconnections are made using 0.1” posts/sockets and DuPont jumpers. A wire list based on the Arduino signal names is presented for this purpose. Although I’ve favored the Arduino Nano in my projects, I’ve actually used a Uno version of the Arduino for my latest version. It’s larger but easier to mount and has 0.1” headers pre-installed.

Arduino pin usage

ID	Name	Usage
D0		
D1		
D2	ENC-A	Rotary encoder (rev 1.9)
D3	ENC-B	Rotary encoder (rev 1.9)

D4	LCD	LCD DB6
D5	LCD	LCD DB7
D6	LCD	LCD 'E'
D7	LCD	LCD 'RS'
D8	KEY IN	Ground for Key Down.
D9	SPKR	TONE, BEEP_PIN, for Spkr use if desired.
D10	TX_Out	Goes LOW *after* registers loaded for TX frequency
D11	SW2	SW2 aka PB2
D12	SW1	SW1 aka PB1
D13		Arduino on-board blue LED
A0	LCD	LCD DB4 (rev 1.8)
A1	LCD	LCD DB5 (rev 1.8)
A2	SB_Relay	Can control a sideband select relay in the radio. HIGH = LSB
A3		
A4	SDA (I2C)	SDA on Si5351a module
A5	SCL (I2C)	SCL on Si5351a module
GND	GROUND	GND from power in, display, encoder, SW1, SW2, SW3, Si5351a

And info on the LCD:

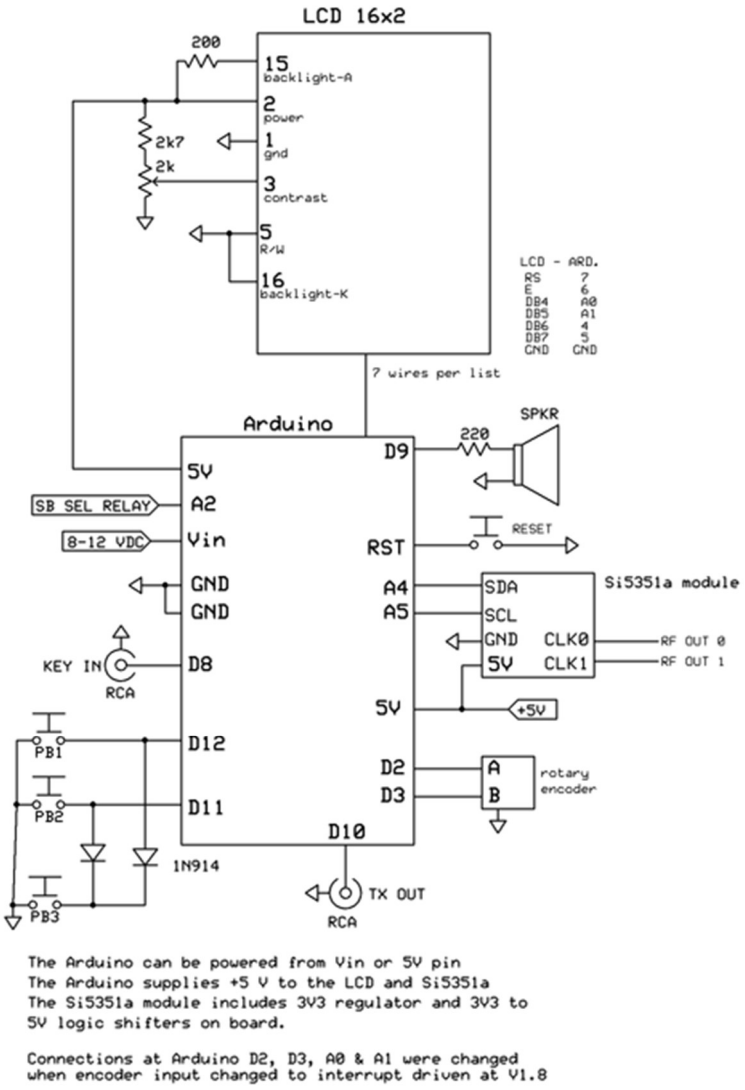
LCD Pins:

Pin #	Name	Notes / connections
1	Vss	GND
2	Vdd	+5V
3	Vo	Contrast, +1.15 V from voltage divider (approximate, adjustable)
4	RS	Arduino D7
5	R/W	Jumper to Vss / GND for "write only" LCD
6	E	Arduino D6
7	D0	N/C
8	D1	N/C
9	D2	N/C
10	D3	N/C

11	D4	Arduino A0 (rev 1.8)
12	D5	Arduino A1 (rev 1.8)
13	D6	Arduino D4
14	D7	Arduino D5
15	A	Backlight LED, 220 Ω then to 5V
16	K	Backlight LED, to GND

Schematic diagram:

(Next page)



Some specifications:

Frequency range: 3.5 MHz to 150.000 MHz

Resolution / step sizes: 1, 10, 100, 1000, 10,000, 100,000, and 1,000,000 Hz

Controls:

Rotary Encoder

Three pushbuttons

Display / indication:

16x2 LCD display with backlight. Hitachi interface.

Output level:

About 3 Vp-p, or 2.09 Vpp into 50 Ω. Use a blocking capacitor if the DC component is not desired.

Output is a square wave.

Load resistance:

Some sources suggest that cross-talk and other issues can be minimized by using a load higher than 50 Ω, perhaps 80 Ω instead. Also, loading problems may be solved by using a buffer, such as a 74LVT04 or another chip in the family.

Current draw:

With an 11 VDC supply, about 60 mA.

Installing the program on your Arduino board:

- If you don't already have it, download the Arduino IDE at https://www.arduino.cc/en/main/software. It's available for Windows, MAC OS X and Linux. There's also a web based version you can use without downloading the program. But I use the local IDE.
- It's an Arduino rule that your source code must be in a folder having the same base name as the source file. In my case, the file is si5351a_quad.ino so it goes in a folder called si5351a_quad under my Arduino folder. If you double-click the source file to start the Arduino and don't have it in the appropriate folder, the IDE will offer to create the folder for you. Either way, go ahead and load the file.
- I suggest running the IDE (integrated development environment) first, before connecting your Arduino, by double-clicking on the source file. Go to the Tools menu and mouse down to "Port" and note the list of COM ports. Then plug in the Arduino, wait a short time and check again. (If your system has to install a new driver for the Arduino, it may take longer than the usual several seconds.) The new port number that appeared is for your Arduino and you should select it.
- Again, go to Tools and mouse down to "Board". Click the item with the name of your board. I have been using both the Nano and the Uno boards in this project.
- Go to Tools again and down to "Processor". I have "ATmega328P (Old Bootloader)" checked. Depending on the source and vintage of your Arduino, it may just be ATmega328P without the Old Bootloader part. If you get a bunch of errors, try the other selecton.
- The Tools menu also has an entry for selecting a Programmer. We aren't using an external programmer here and I understand that the choice is irrelevant, but I have "Arduino as ISP" selected.

- Now you're ready to cross your fingers and see if it works. In the Arduino IDE below the menu line is a dark blue bar with some icons at the left. The left-most one is a check mark in a circle called "verify" which tells the IDE to compile the program and check for errors. Next to it is another circle with an arrow to the right called "upload" which tells the IDE to compile the program and if there are no errors, send it to the Arduino. You can try one and then the other, or just go straight to the upload button.
- Any errors will be displayed in red in the bottom window. They can be a bit cryptic.
- The one problem I can foresee as possibly occurring would be "library not found". Most commonly used libraries are installed with the IDE installation. But I don't know if the ones used in this program are all in that category. If you search the source code for "#include", you'll find the list of libraries used. Installing a missing library is fairly well automated, but there a couple of ways to do it.
  Go to Tools>Manage Libraries, and a box with a search field will enter. Type in the name of the missing library and it will be found. Then follow the prompts to install it.
  Another, earlier method that still exists is under Sketch>Include Library. It includes a list of libraries and you can scan it for your missing library.

User customization of the source code

A person with Arduino programming skills could customize the software to whatever extent desired. But there are also changes a non-programmer could make to customize the program as desired. These include the starting frequency for each band, the available step sizes and the crystal oscillator reference frequency. You can also change the response of the rotary encoder, giving 1, 2, or 4 encoder transitions per frequency step. And as of Rev 1.9, you can go all the out to 128 transitions per step, for those really fast encoders. See below.

I've labeled some potential areas for change with "USERS:". You can search the source code for this term to find sections you might wish to edit.

Revisions

Revision 1.8: interrupt driven rotary encoder input:

At least one user had a rotary encoder with a large number of pulses per rotation plus a speedy ball bearing mounted shaft, and the encoder could outrun the software. This could cause not only missing steps, but also spurious changes in direction.

So I decided to implement a change from polled encoder monitoring to use of an interrupt routine. It's still possible for the incoming encoder pulses to outrun the software, but in this case the interrupt routine simply increments a queue of pending up or down steps and the main code will catch up quickly if it gets behind momentarily.

I made another change with revision 1.8 to deal with the encoder going too fast: If the mainline code gets 10 steps behind, it takes a step 10 times as large as the one currently selected and reduces the demand count by 10. [See Rev 1.9 below for another change giving more choices.]

There are hardware changes required to go from the previous revision to the new revision. The only two pins which are capable of generating interrupts were in use elsewhere, so pin usages had to be swapped around.

Things to change:

- Move ENC_B input from A1 to D3 (wire from encoder - B)
- Move ENC_A input from A0 to D2 (wire from encoder - A)
- Move LCD DB4 pin 11 to Arduino A0 (was D2)
- Move LCD DB5 pin 12 to Arduino A1 (was D3)

Note that if your frequency goes down when you turn the knob CW and vice-versa, swap the ENC-A and ENC_B connections.

V1.9: Programmable rotary encoder speed selection

There is a lot of variation in the number of pulses or transitions per revolution you get with rotary encoders. My software produces one frequency step per transition. If your encoder seems too fast, you can compile the software to divide by values from 1 to 255. You simply make a minor edit to the source code and then upload it to your Arduino.

The are a number of places in the sketch where the user can customize things. They are marked with the word: USERS: So search for that term to find those items. In this case you will find this information:

```
#define DIV_FACTOR 4 // edit this line see below for DIV_FACTOR to
                     // actual division ratio
```

The number after DIV_FACTOR specifies the number of encoder events that will produce one frequency step.