Si5351 Xtal Substitute Module

By Terry Mowles VK5TM

The genesis of this project was as a substitute for the increasingly harder to find 10.710MHz xtals in the Picastar project. The original can be seen here: *http://www.vk5tm.com/starlo.php*

As many would be aware, quartz crystals in custom frequencies (and even some that used to be standard frequencies) are now either getting very expensive or even unobtainable. Xtals upwards of 100MHz are also not readily available to the homebrewer.

This module, using the SI5351 can generate a frequency from as low as 8kHz, up to 160MHz. It also has 4 configurable output levels of drive 2,4,6 or 8mA. The PIC also goes to sleep after programming the Si5351.

There are many projects on the internet using the SI5351 with two or three outputs in a VFO + BFO arrangement etc. I am not a fan of this arrangement, as the 3 output Si5351 variant (as used in this project) has a common supply rail for all output stages, resulting in a fair amount of crosstalk between outputs. This project is restricted to using just one output for this reason.

Build information

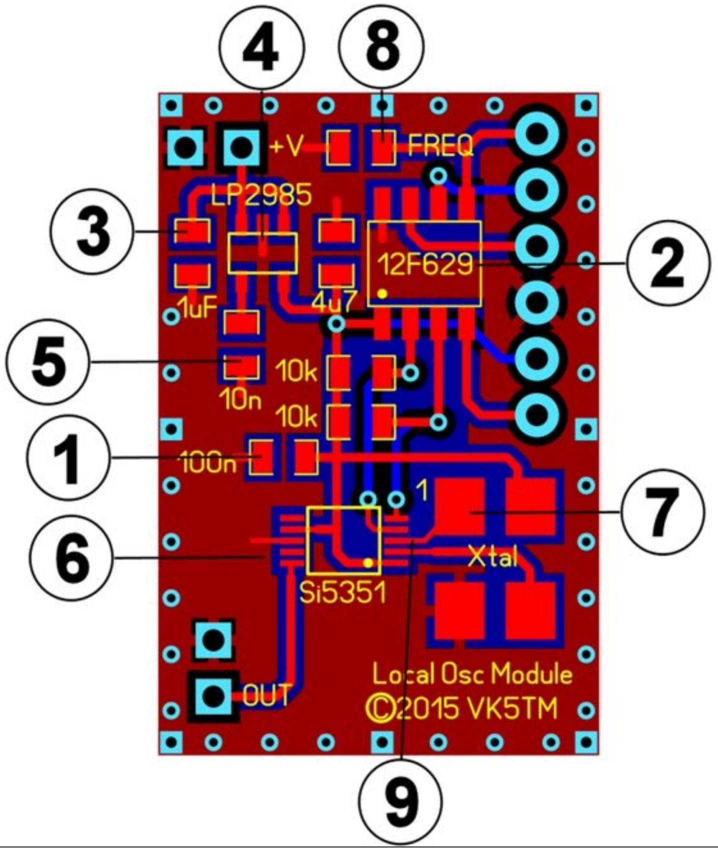
Before describing how to generate the code for the frequency of interest, I will go through the build process. This project should be an easy build for those experienced in using SMD components.

Having said that, those with average to good soldering skills and a steady hand, should also be able to do it with a little patience and perseverance.

The pcb itself is double sided, 20mm x 30mm in size and because of the narrow spacing of the Si5351 pins, is not really suited to etching at home. But if you really want to try, a pdf file of the layout is available on the *CQDATV site*. Otherwise, I have a quantity of pcb's available for sale.

This build description is only a suggestion of how to go about it but there are several important points to note that will be explained through the procedure.

The following numbered drawing corresponds to the numbered steps following. Please read completely through them first before commencing construction. (Please note, this is an earlier version of the pcb, but the components are still in the same relative positions. Also ignore the number 9 step, it no longer applies).



1: FIRST IMPORTANT NOTE: The PIC is static sensitive, so safe antistatic work practices should be used.

A programmed PIC can be erased or otherwise be made non operative from the effects of static electricity (ask me how I know this!).

Install the 100n capacitor FIRST, regardless of whether you are going to program the PIC once installed on the pcb or have already programmed the PIC.

2: Install the PIC. If programming on board, do it now. The header pins on the right hand side are setup to work with PicKit 1, 2 or 3 programmers (see component overlay diagram further on).

For other programmers, you will need to make up an appropriate cable.

Also, see the important smoke alert at the end of the build description.

3: Install the 1uF and 4.7uF capacitors. These are Multi Layer Ceramic Caps (MLCC) and are not polarity sensitive.

SECOND IMPORTANT NOTE: DO NOT use Electrolytic or Tantalum caps, they do not have the right characteristics for use in this circuit.

4: Install the LP2985 3.3v regulator.

5: Install the 10n cap and 2 x 10k resistors.

6: This is the part that needs the most care in mounting the Si5351 the pins are small and very close together.

A good magnifier and plenty of flux are your friends. I soldered this using the "flood soldering method" i.e. apply flux, position chip (the right way round) and solder across all the pins on both sides (don't worry about the solder bridges).

Next, with more flux and desoldering braid, remove the excess solder.

Although it may appear that all the solder has been removed, there will be more than sufficient under the pins for a good connection to the pads.

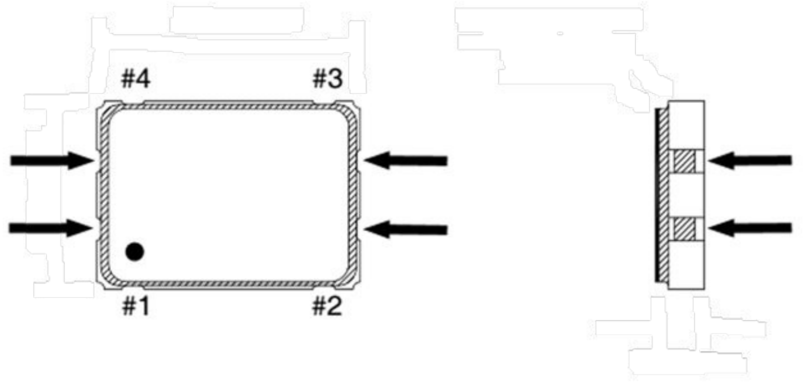
Don't be tempted to "touch up" the pins with a bit of extra solder, you are only asking for a world of trouble.

7: Fit the oscillator module.

The BOM (bill of materials) lists a 7 x 5mm oscillator module. You may want to source a 5 x 3mm module (BOM available as part of the download on the CQDATV site).

THIRD IMPORTANT NOTE: (and one that caught me out) some oscillator modules have small pads on the ends (see drawing below), which for no logical or sensible reason, are not connected to their adjacent pins.

In the oscillator module I used, that pad adjacent to pin 2, was connected to pin 1! DO NOT get solder on these pads!!!



8: This is where you decide which frequencies to generate. If only generating a single frequency, you can skip this step. (The original software was capable of generating one of two frequencies).

An alternative programmed frequency can be generated and this is done by placing a zero ohm 0805 resistor or linking the two pads with a piece of wire at the position marked "FREQ".

You could also wire a switch across these pads if desired.

Note that this is an either/or function, you cannot switch between the two frequencies while the module is powered on.

That is basically the build information, so the next step is check and double check that everything is in the correct place, correctly oriented and soldered and that there are no solder bridges.

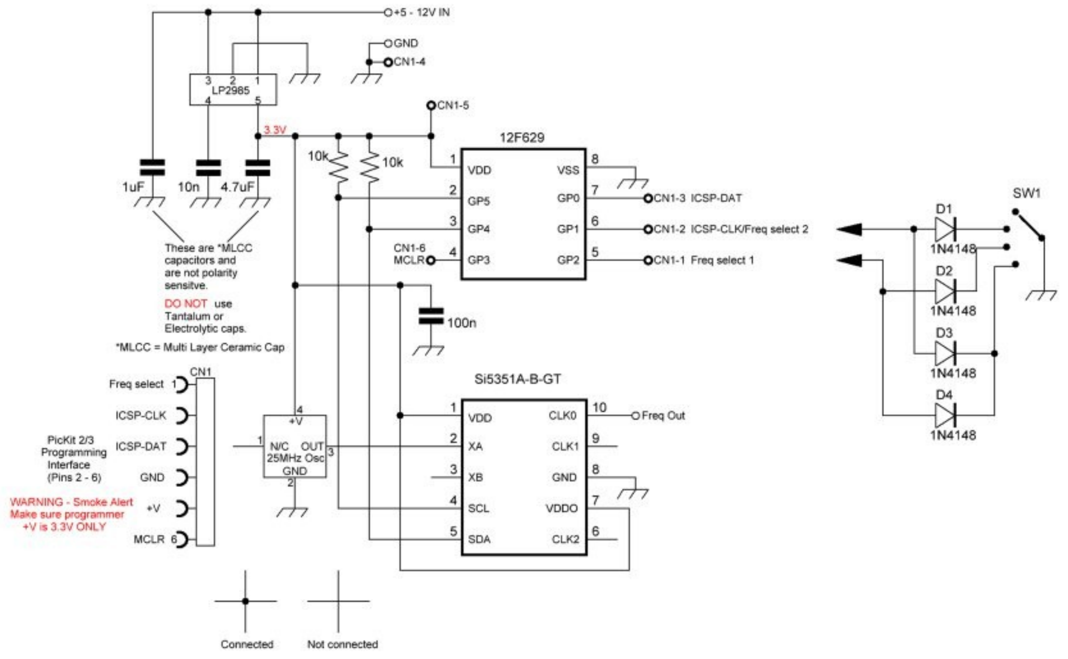
NOTE > SMOKE ALERT: If you have a fully loaded module and are experimenting with changing the program (or forgot to program the PIC in a previous step), it is VITAL that your programmer does not output more than 3.3V on it's V+ pin.

If it does, you will let out the all important smoke from the Si5351 and the module will never work again.

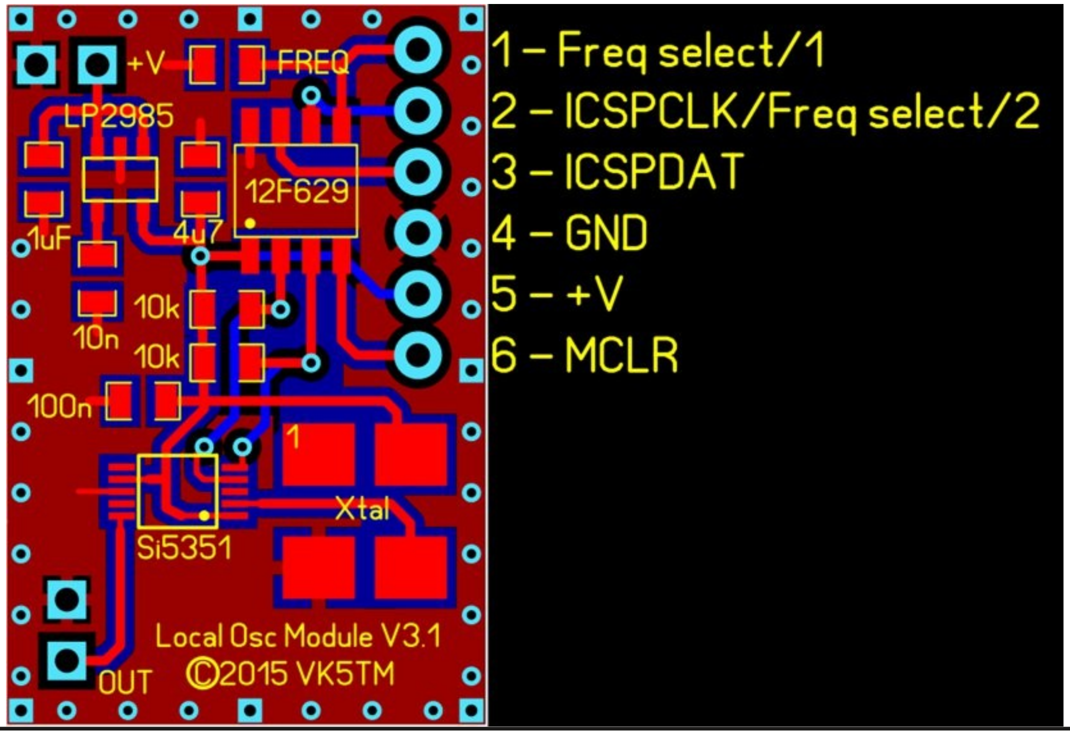
All that remains is to connect a source of power between 5V and 12V (the regulator is a low dropout type and will work satisfactorily from 5V) and check the output.

The via's around the edge of the module are all grounded, so you can use these to solder a shield to. That's all there really is to it.

Here is the schematic of the module. You will notice a diode matrix and switch to the right hand side. This enables multi frequency switching and is explained later in this article.



And component overlay and programming connector details.



A larger version of the schematic and overlay is included in the download available from the *CQDATV site*.

Coding the Si5351 xtal substitute frequency

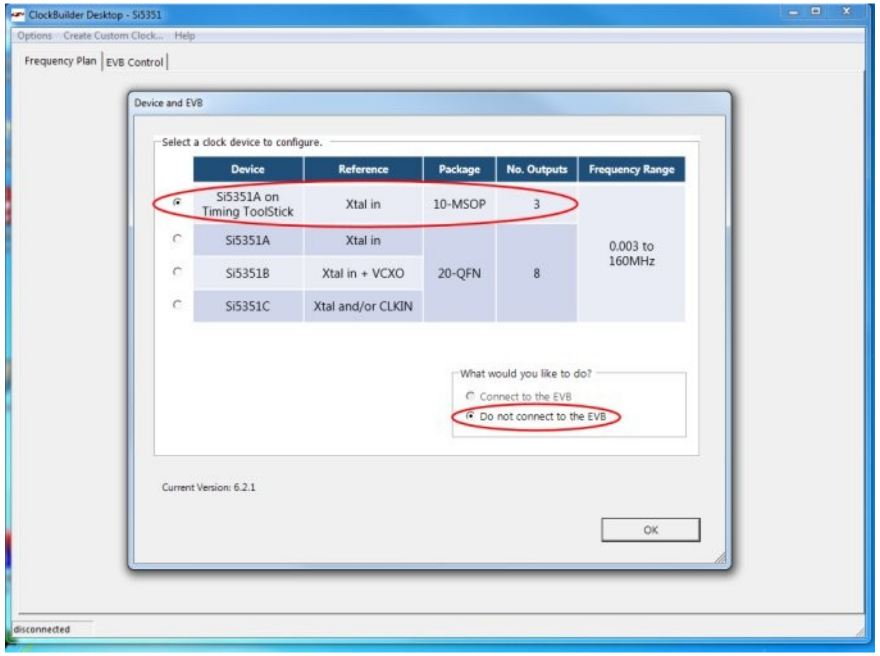
Coding the frequency of the xtal substitute module is a relatively easy task, however, you will need PIC programming skills (or somebody that can do it for you). The asm file, VK5TM_Si5351_LO.asm, is available in the download from the *CQDATV site*.

To start, you will need the Clockbuilder software for the Si5351. At the time of writing, it was available here:

http://www.silabs.com/products/clocksoscillators/pages/timingsoftwaredevelopmenttools.aspx

Look for"Si5351" in the list.

Once you have downloaded and installed it, start the software and you should see the following (without the red ellipses):


ClockBuilder Desktop - Si5351
Options Create Custom Clock... Help
Frequency Plan
EVB Control
Device and EVB

Select a clock device to configure.

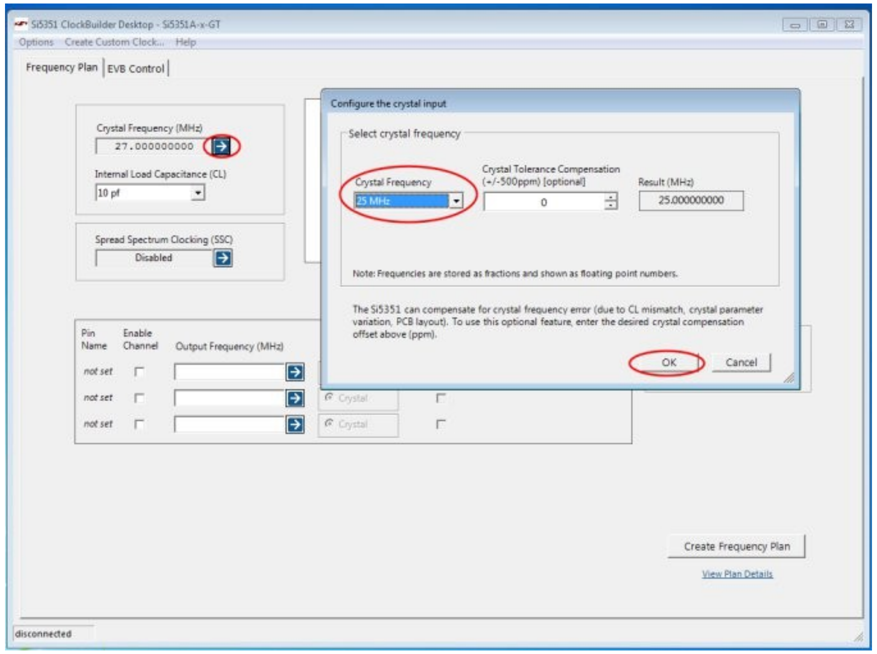
	Device	Reference	Package	No. Outputs	Frequency Range
●	Si5351A on Timing ToolStick	Xtal in	10-MSOP	3	0.003 to 160MHz
○	Si5351A	Xtal in	20-QFN	8	
○	Si5351B	Xtal in + VCXO			
○	Si5351C	Xtal and/or CLKIN			

What would you like to do?
Connect to the EVB [✓ Do not connect to the EVB]

Current Version: 6.2.1

Select the options shown circled in red and click OK. This will take you to the main page.

Click the arrow next to "Crystal Frequency" which will pop up the box where you select "25MHz" from the drop down list. Note that there are only two options, 27 or 25MHz. Again, click OK.


Si5351 ClockBuilder Desktop - Si5351A-x-GT
Options Create Custom Clock... Help
Frequency Plan
EVB Control

Crystal Frequency (MHz): 27.000000000
Internal Load Capacitance (CL): 10 pf
Spread Spectrum Clocking (SSC): Disabled

Pin Name	Enable Channel	Output Frequency (MHz)
not set		
not set		
not set		

Configure the crystal input

Select crystal frequency

Crystal Frequency: 25 MHz
Crystal Tolerance Compensation (+/-500ppm) [optional]: 0
Result (MHz): 25.000000000

Note: Frequencies are stored as fractions and shown as floating point numbers.

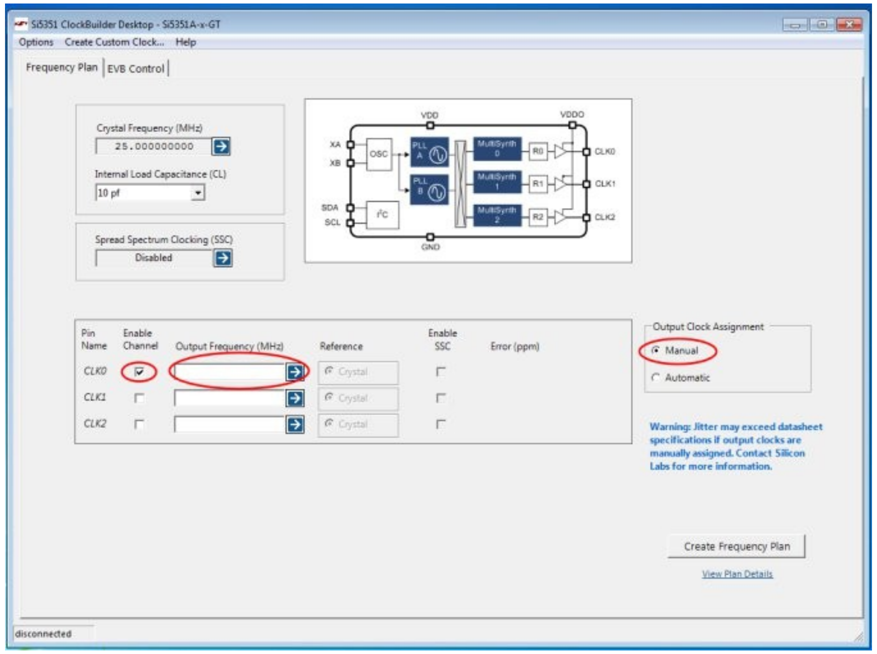
The Si5351 can compensate for crystal frequency error (due to CL mismatch, crystal parameter variation, PCB layout). To use this optional feature, enter the desired crystal compensation offset above (ppm).

On this next page, click "Manual" first and then select the checkbox next to "CLK0". Ignore the warning that comes up, you are only using one output so there can be no jitter problems.

See image top of next page.

Clicking the arrow next to the blank "Output Frequency (MHz)" box will pop up a box where you can enter the frequency required and the "Drive Strength".

You can select a drive strength of 2,4,6 or 8mA.


Si5351 ClockBuilder Desktop - Si5351A-x-GT

Options Create Custom Clock... Help

Frequency Plan | EVB Control

Crystal Frequency (MHz): 25.000000000

Internal Load Capacitance (CL): 10 pf

Spread Spectrum Clocking (SSC): Disabled

Pin Name	Enable Channel	Output Frequency (MHz)	Reference	Enable SSC	Error (ppm)
CLK0	☑		Crystal	☐	
CLK1	☐		Crystal	☐	
CLK2	☐		Crystal	☐	

Output Clock Assignment: [✓ Manual] Automatic

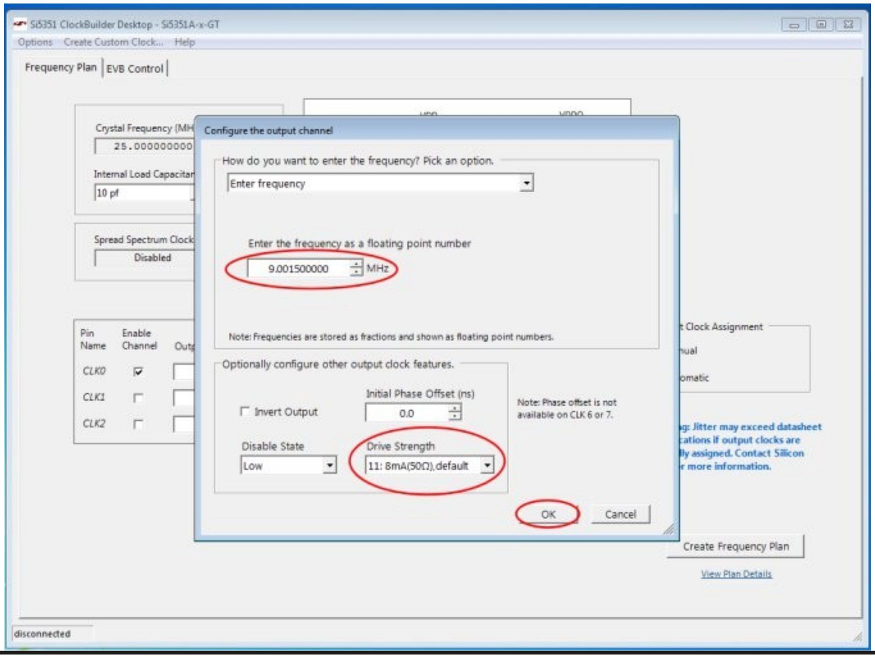
Warning: Jitter may exceed datasheet specifications if output clocks are manually assigned. Contact Silicon Labs for more information.

Create Frequency Plan

View Plan Details

disconnected

Depending on your application, you will need to experiment with this setting. The higher the current level, the higher the drive. If unsure of the level needed, select 8mA, you can always put an attenuator on the output if it is too high.


Configure the output channel

How do you want to enter the frequency? Pick an option.: Enter frequency

Enter the frequency as a floating point number: 9.001500000 MHz

Note: Frequencies are stored as fractions and shown as floating point numbers.

Optionally configure other output clock features.

☐ Invert Output

Initial Phase Offset (ns): 0.0

Note: Phase offset is not available on CLK 6 or 7.

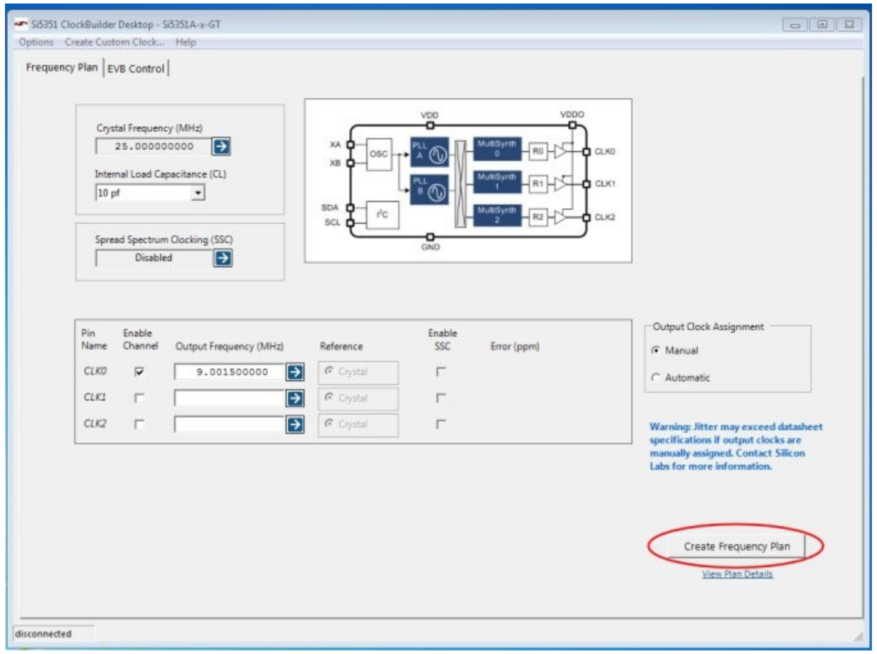
Disable State: Low

Drive Strength: 11: 8mA(50Ω),default

OK Cancel

For this article, I have entered an example frequency of 9.0015MHz for use as the LSB frequency with a 9MHz filter.

Once you have made your choices, click on the "Create Frequency Plan" button. Nothing will appear to happen, but it does generate the needed file.


Si5351 ClockBuilder Desktop - Si5351A-x-GT

Options Create Custom Clock... Help

Frequency Plan | EVB Control

Crystal Frequency (MHz): 25.000000000

Internal Load Capacitance (CL): 10 pf

Spread Spectrum Clocking (SSC): Disabled

Pin Name	Enable Channel	Output Frequency (MHz)	Reference	Enable SSC	Error (ppm)
CLK0	☑	9.001500000	Crystal	☐	
CLK1	☐		Crystal	☐	
CLK2	☐		Crystal	☐	

Output Clock Assignment: [✓ Manual] Automatic

Warning: Jitter may exceed datasheet specifications if output clocks are manually assigned. Contact Silicon Labs for more information.

Create Frequency Plan

View Plan Details

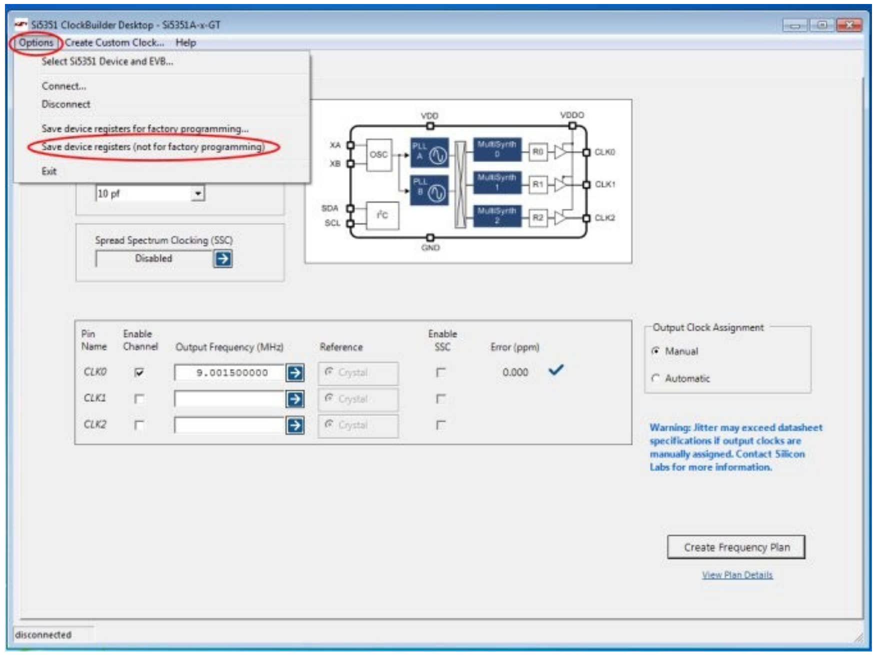
disconnected

You can click the "View Plan Details link if you wish, but it will not tell you anything really useful.

Now, and this is the important bit, click on "Options" in the top menu bar and select "Save device registers (not for factory programming)" see image top of next page.

This will bring up the normal Windows Save menu. Give the file a name and select a location to save it to.

Once you have done that, you can now open the generated text file, the picture at the bottom of the next page is a portion of that file. The important bit is from the circled "Register_MAP" tag and below.



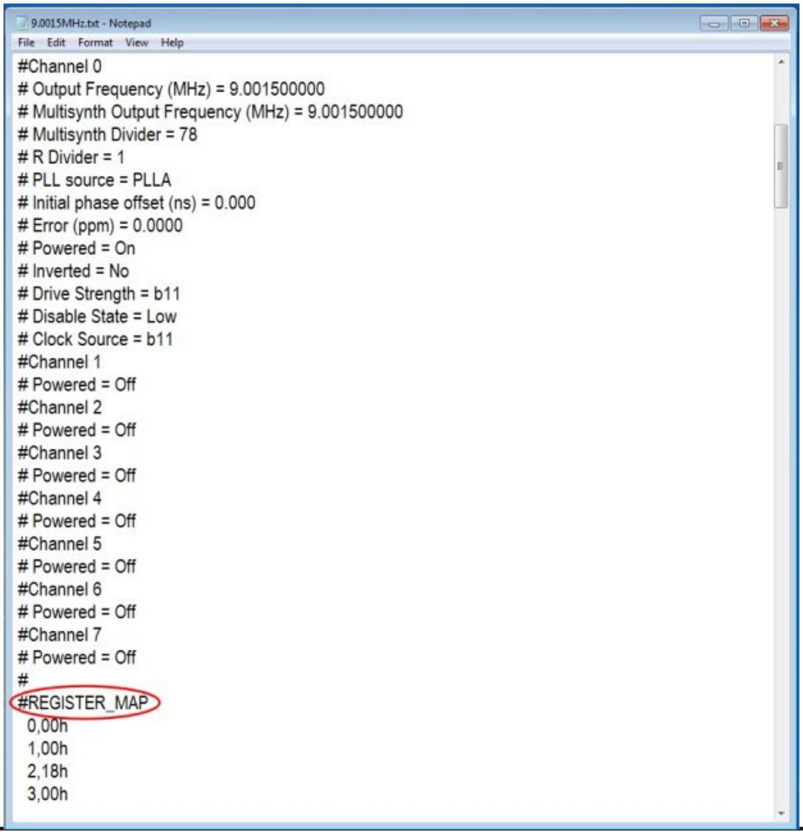
10 pf

Spread Spectrum Clocking (SSC): Disabled

Pin Name	Enable Channel	Output Frequency (MHz)	Reference	Enable SSC	Error (ppm)
CLK0	✓	9.001500000	Crystal		0.000 ✓
CLK1			Crystal		
CLK2			Crystal		

Output Clock Assignment: [✓ Manual] Automatic

Warning: Jitter may exceed datasheet specifications if output clocks are manually assigned. Contact Silicon Labs for more information.



```
#Channel 0
# Output Frequency (MHz) = 9.001500000
# Multisynth Output Frequency (MHz) = 9.001500000
# Multisynth Divider = 78
# R Divider = 1
# PLL source = PLLA
# Initial phase offset (ns) = 0.000
# Error (ppm) = 0.0000
# Powered = On
# Inverted = No
# Drive Strength = b11
# Disable State = Low
# Clock Source = b11
#Channel 1
# Powered = Off
#Channel 2
# Powered = Off
#Channel 3
# Powered = Off
#Channel 4
# Powered = Off
#Channel 5
# Powered = Off
#Channel 6
# Powered = Off
#Channel 7
# Powered = Off
#
#REGISTER_MAP
 0,00h
 1,00h
 2,18h
 3,00h
```

If you only want the one frequency, then you are finished with this step, if you want a choice of frequencies, then you need to do it all again and generate another file/s, corresponding to the number of frequencies required.

Remember to save subsequent files with a different name.

What you need to do with this information is explained in the following steps.

(Larger copies of all images are available in the *download*.)

About now would probably be a good time to go and get that cup of coffee (or other beverage) you have been thinking about.

Changing the PIC program

This will require reasonable PIC programming skills. I have commented the ASM file as much as possible, but if something is unclear, contact me and I will clarify it as best I can.

Firstly, a brief explanation of the text file that is generated by Clockbuilder.

Everything with a "#" in front of it is a comment.

At the top is information on the options selected when generating the file. Then you get to the important bit, the "Register_MAP".

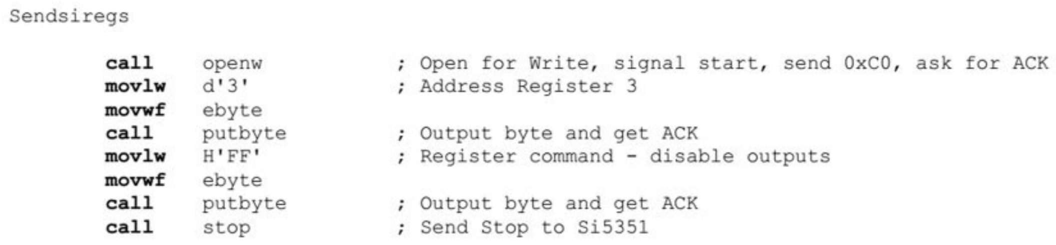
Parts of this are what is needed to be programmed into the PIC. The first digits on a line are the register number in decimal, followed by the command in hex.

While the list runs from 0 to 232, you do not need to add all of these.

The general sequence of programming the registers is shown in the code snippet below. First you send the address of the chip. The address of the chip is 0x60 as per the datasheet and, just to be confusing, in the program it shows as 0xC0. Just trust me that it is correct. If you really need to know why, study the I2C protocol and how the addressing works. HINT 7bits and MSB (Most Significant Bit) first.

Following the sending of the IC address, you send the register address you want to program, 3 in the snippet below and then the command, H'FF'. At the end, you send the stop command.

This is for individual, seperated registers.



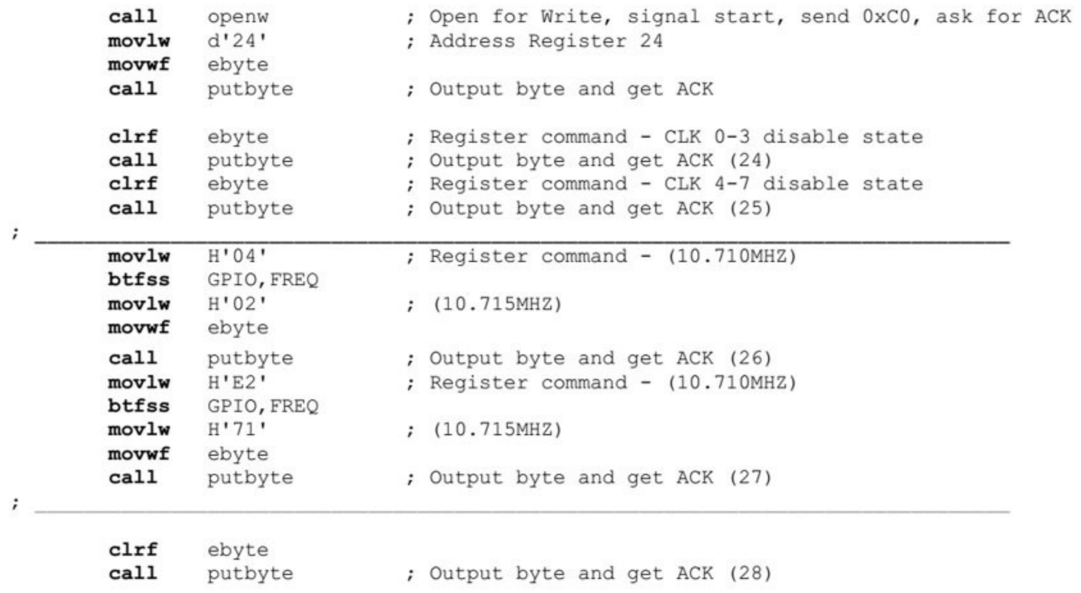
```
Sendsiregs

        call    openw           ; Open for Write, signal start, send 0xC0, ask for ACK
        movlw   d'3'            ; Address Register 3
        movwf   ebyte
        call    putbyte         ; Output byte and get ACK
        movlw   H'FF'           ; Register command - disable outputs
        movwf   ebyte
        call    putbyte         ; Output byte and get ACK
        call    stop            ; Send Stop to Si5351
```

Where the register addresses are sequential, this will get very tedious, so as per the I2C specs, you can send the chip address, the address of the first register in the list to program and then sequentially send a list of commands.

The Si5351 automatically steps to the next register in turn as you send the commands. Once you have reached the end of the list, then you send the stop command.

The following short code snippet (see image top right) is part of the sequence that programs registers 24 to 92 and the stop command is called at the end (not shown).

Now to the registers to be programmed, firstly for a single frequency. According to a post on the SiLabs forum it goes like this:



```
        call    openw           ; Open for Write, signal start, send 0xC0, ask for ACK
        movlw   d'24'           ; Address Register 24
        movwf   ebyte
        call    putbyte         ; Output byte and get ACK

        clrf    ebyte           ; Register command - CLK 0-3 disable state
        call    putbyte         ; Output byte and get ACK (24)
        clrf    ebyte           ; Register command - CLK 4-7 disable state
        call    putbyte         ; Output byte and get ACK (25)
;  ________________________________________________________________________________
        movlw   H'04'           ; Register command - (10.710MHZ)
        btfss   GPIO,FREQ
        movlw   H'02'           ; (10.715MHZ)
        movwf   ebyte
        call    putbyte         ; Output byte and get ACK (26)
        movlw   H'E2'           ; Register command - (10.710MHZ)
        btfss   GPIO,FREQ
        movlw   H'71'           ; (10.715MHZ)
        movwf   ebyte
        call    putbyte         ; Output byte and get ACK (27)
;  ________________________________________________________________________________

        clrf    ebyte
        call    putbyte         ; Output byte and get ACK (28)
```

1. Disable all outputs. reg3 = 0xFF

2. Write reg187 = 0xC0

3. Power down all output drivers

    reg 16 = 0x80
    reg 17 = 0x80
    reg 18 = 0x80
    reg 19 = 0x80
    reg 20 = 0x80
    reg 21 = 0x80
    reg 22 = 0x80
    reg 23 = 0x80

(Note that I haven't checked whether all 8 output registers are present in the 3outout version of the Si5351, but it accepts the code, so they may be there.)

4. Set interrupt masks register (see Register 2 description in datasheet). I didn't do anything with this one.

5. Set crystal load capacitance, XTAL_CL in reg183[7:6]. See datasheet for register description.

6. Write registers 1592 and 149170 using the contents of the register map generated by ClockBuilder Desktop.

7. Apply PLL A and PLL B soft reset reg177 = 0xAC

8. Enable output with OEB control in register 3. (OEB = Output Enable Bits).

So basically, you need to extract from the Clockbuilder file, all the values of registers 15 92 and 149 170 and transpose them into the appropriate places in the asm file.

A lot of the values are 0x00, so in the asm file you will see the command "clrf ebyte", which is the same as loading it with a zero value.

You may notice two or more consecutive sets of clrf ebyte, call putbyte and another clrf ebyte. This was done to make sure that the file remained at zero, as the STATUS C register is used in moving the bits out to the Si5351.

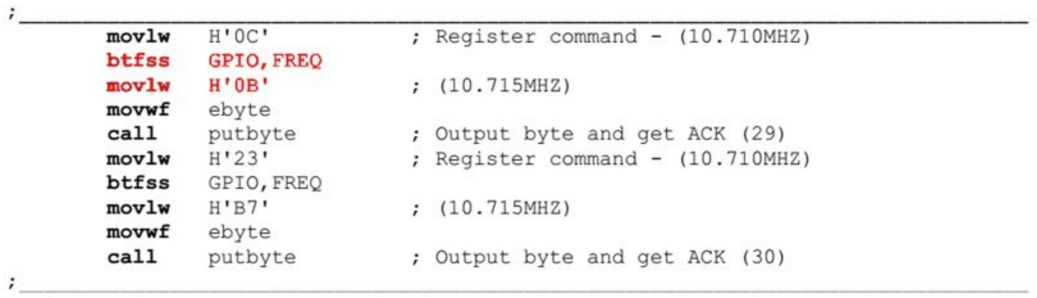
Every section in the asm file corresponding to each register is marked with the register number like so (24).

If you are doing a two frequency program, there is a preliminary step that needs to be done, note that, in the first software version, it will not switch between frequencies while powered up.

(See 'Modifying the Si5351 xtal substitute software Part 2' for multi frequency operation.)

You need to compare the two frequency plans and note which registers differ from each other. There will not be a large number of them.

The code snippet below shows the structure of how the test is done to determine which frequency to generate. It first loads the default frequency value into the 'W' register of the PIC and then tests if the "FREQ" pin is low.
If it is, it loads the new value into the 'W' register and proceeds to load that new value into the Si5351. You will find various sections delineated within the asm file where this happens with the current set of frequencies.



```
;_______________________________________________________________________
        movlw   H'0C'           ; Register command - (10.710MHZ)
        btfss   GPIO,FREQ
        movlw   H'0B'           ; (10.715MHZ)
        movwf   ebyte
        call    putbyte         ; Output byte and get ACK (29)
        movlw   H'23'           ; Register command - (10.710MHZ)
        btfss   GPIO,FREQ
        movlw   H'B7'           ; (10.715MHZ)
        movwf   ebyte
        call    putbyte         ; Output byte and get ACK (30)
;_______________________________________________________________________
```

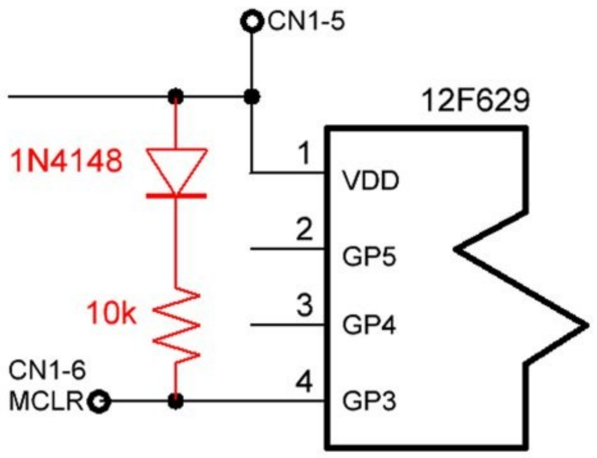
Modifying the Si5351 xtal substitute software Part 2

This is the second part of modifying the SI5351 LO module software such that it can switch between various frequencies and doesn't need to be powered down to do so.

The only hardware difference is a switch matrix using the "Frequency select" pin and one of the programming pins and modified software.

As it stands, I have only used two pins for the frequency select function out of the four available, so this will give you the choice of 4 frequencies.

The more ambitious among you may want to use the additional two available pins (ICSPDAT & MCLR) and get up to 16 frequencies. Note that internal pullups are used on the frequency select pins, but this function is not available on pin 4 of the 12F629 (GP3/MCLR), so you will need to add a pull up resistor (10k) and an isolating diode to this pin.



I have not drawn a pcb for the diode switching, these can easily be mounted on the back of the frequency change switch.

Multi frequency Software

An asm file, VK5TM_Si5351_Multi_LO.asm, written at the request of another ham, can be used as an example of how to accomplish multiple, live frequency switching. It is in the download available from the *CQDATV site*.

It is much the same as the original software as far as programming the Si5351 is concerned, but has had major changes to accommodate this modification. It now uses interrupts. The PIC still goes to sleep when not actively reprogramming the Si5351.

In this example, the four frequencies generated are: 54MHz, 53.9985MHz, 53.0015MHz and 53.9993MHz.

Hopefully, it is commented enough for you to work out what is going on, contact me if you need clarification on anything regarding this software.

TV-AMATEUR
www.agaf.de
Nr. 177
47. Jahrgang
2. Quartal 2015
Zeitschrift für Bild- und Schrift-Übertragungsverfahren
AGAF auf der HAM RADIO
Halle A1 - Stand 246
Aus dem Inhalt:
CMOS-Kameras im Fokus • TV-Testbild-Generator mit RaspberryPi • Zweiweg-Schmalband-DATV bei 145 MHz Ultra HD aus dem Weltall (ISS) • HAM RADIO-WLAN kostenlos • Fernseh-DX-Hobby – gestern und heute • Rückblick: der DATV-Zeppelin • Neue AGAF-Homepage

TV Amateur is a German Language ATV Magazine It is published 4 times a year and if you would like to subscribe go to http://www.agaf.de/