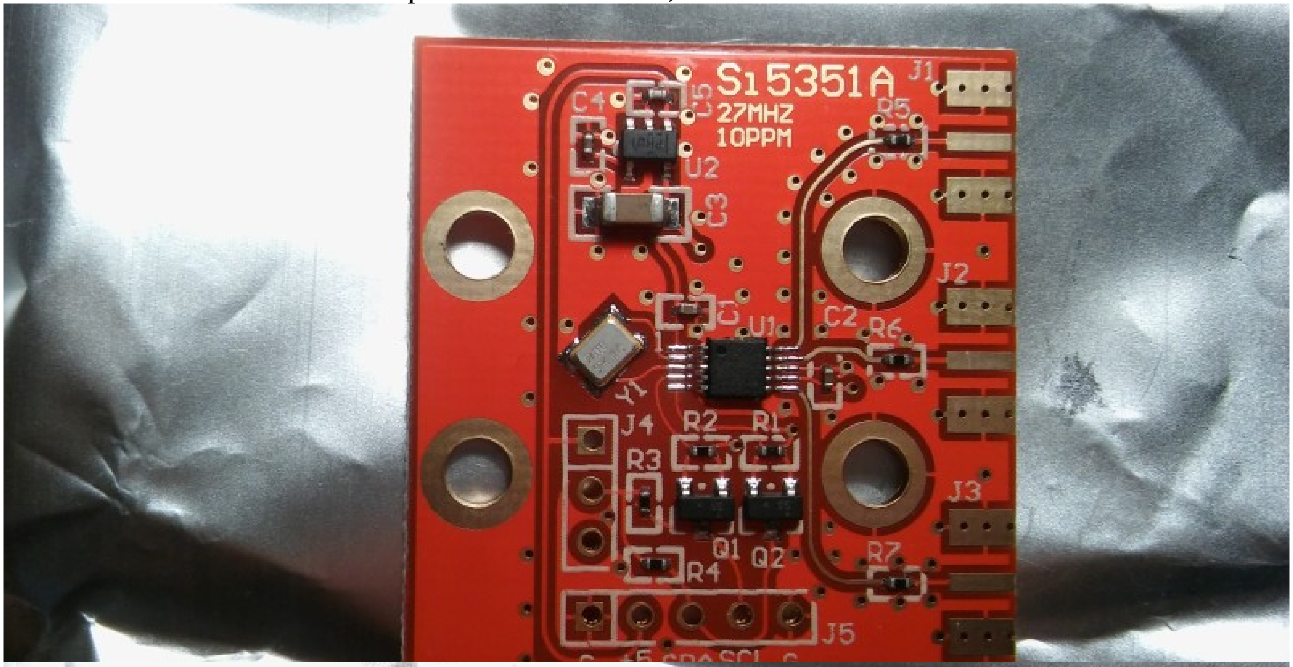
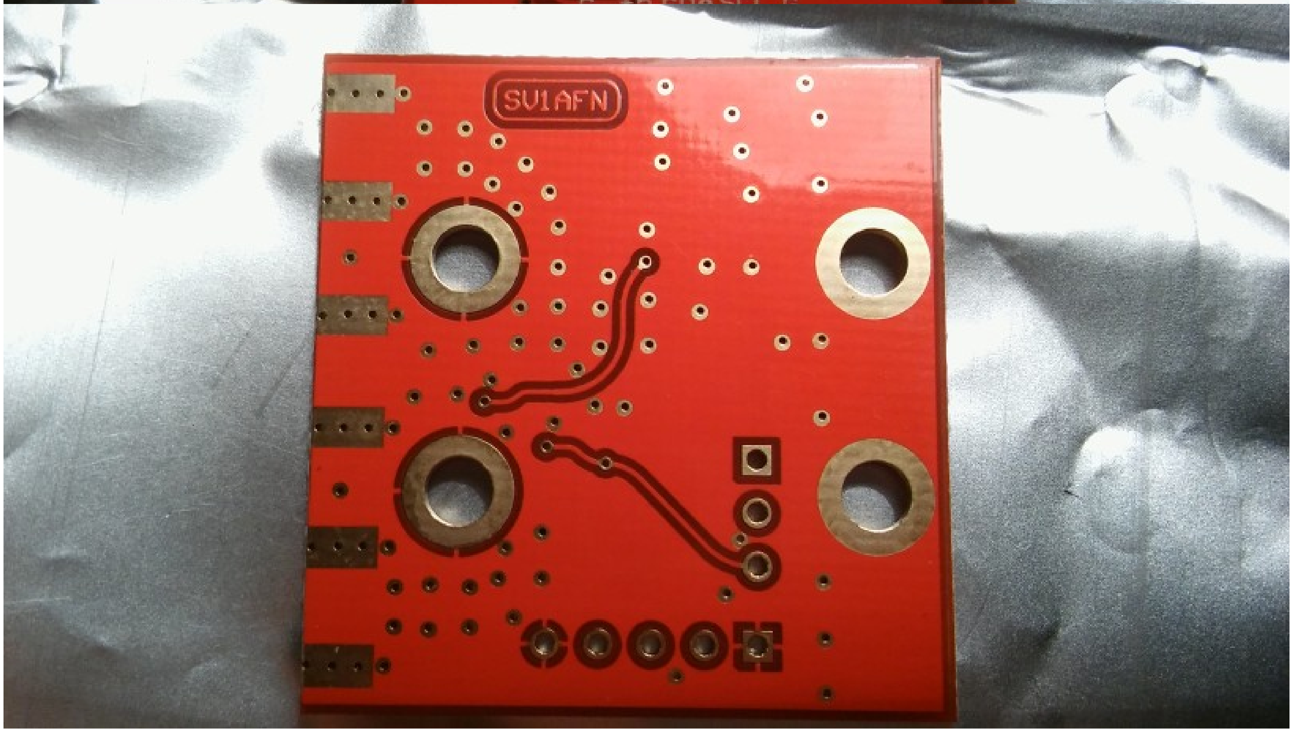
A Frequency Generator with Si5351A Synthesizer Module

This article describes a simple Frequency Generator that can be used in place of a Variable Frequency Oscillator (VFO) or to replace a Crystal Oscillator. My basic goal was to build a project that could be duplicated easily with components that could be easily sourced through mail order with low cost.

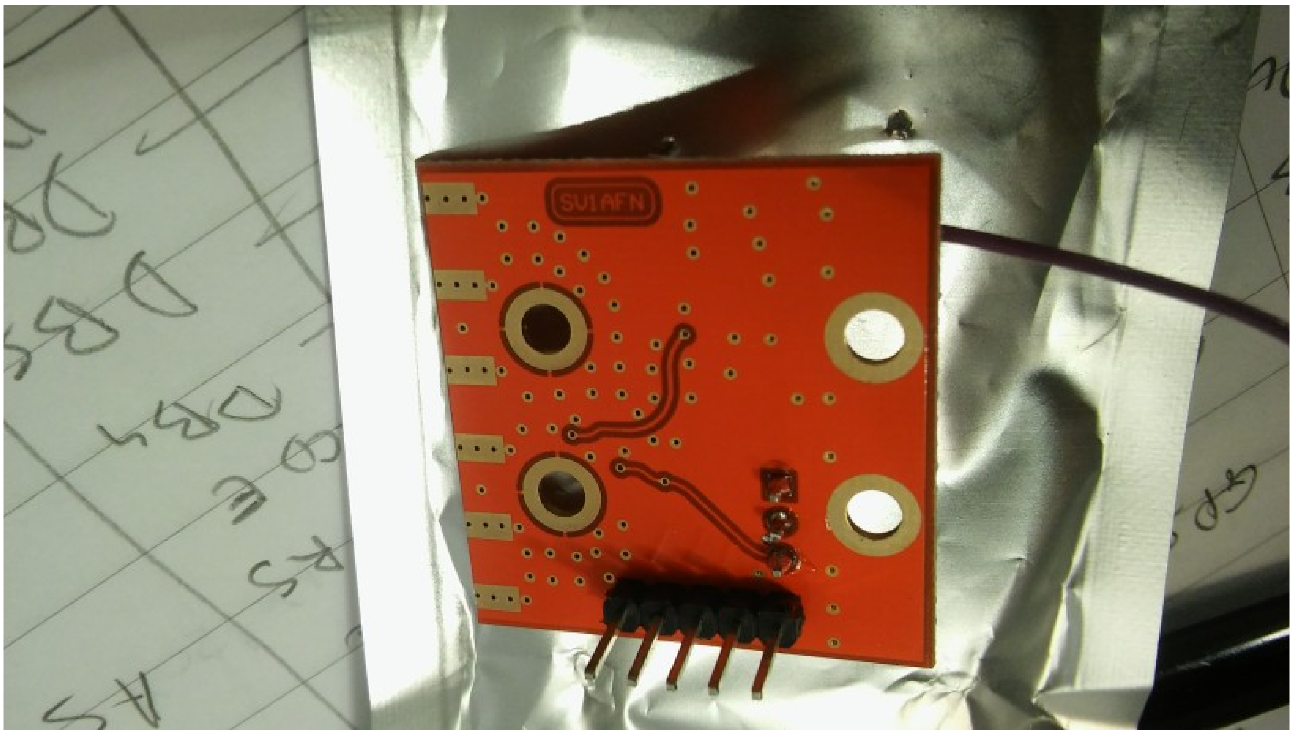
The project initially used the AD9850 DDS (Direct Digital Synthesizer) from Analog Devices that outputs both sine and square waves from 1 Hz to 42 MHz. As I wanted a Generator capable of going higher than that in frequency I decided to proceed with Si5351A controlled from an Arduino board or even better from a barebones ATmega328P-PU, as these MCUs are highly popular. After all I had published the detailed construction of a Generator with AD9850 in SV-News early this year (January 2015 issue).

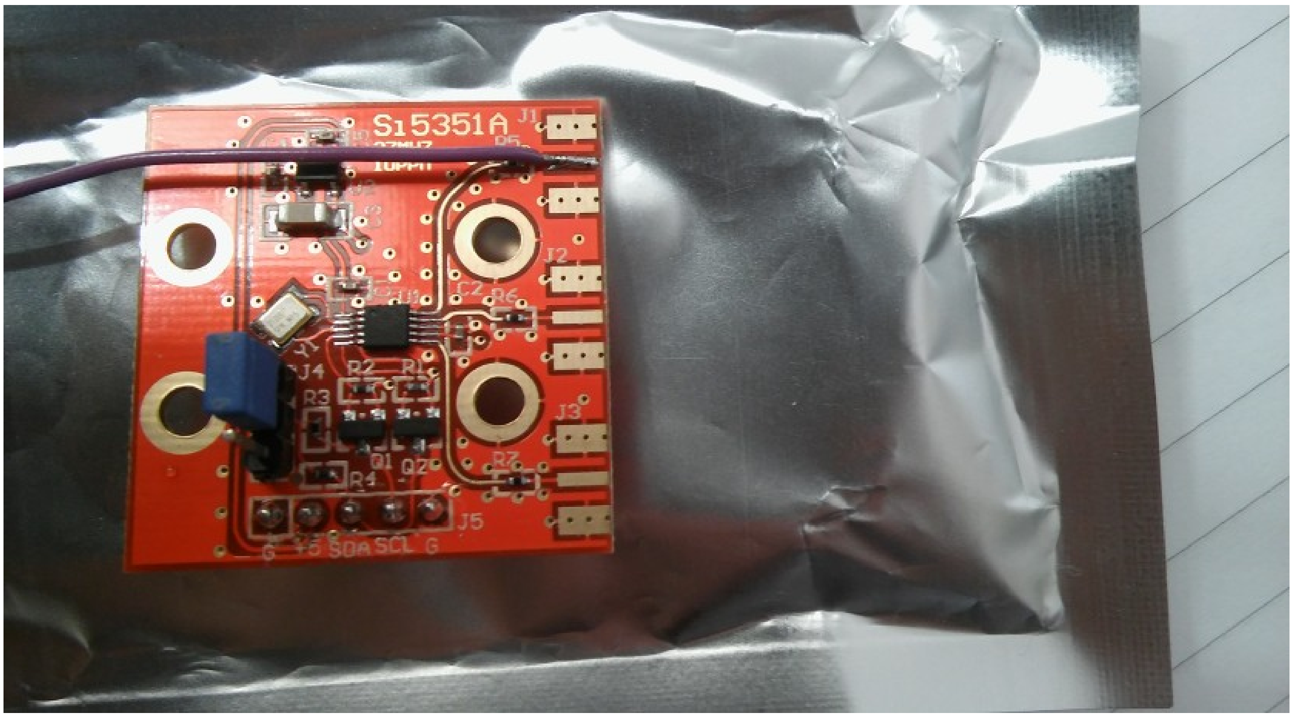
The basis of the project is a small readily available board with a mere surface of 3 by 3 centimeters military grade board that incorporates the Si5153A from Silicon Labs, a special 10 ppm 27 MHz crystal, a 3,3 volt regulator (as Si5351A operates in that level) and two mosfet transistors that operate as level converters for the control ports of the ic (SDA and SCL). An on-board jumper selects the desired level for the ports to either 5 or 3,3 volts.





The ic which is an SMD type can simultaneously produce up to three different square wave frequencies with a range from 8 Khz to 160 MHz and a 50 ohm output impedance. The pcb comes with three gold plated SMA connectors that can be soldered on the board.
At the bottom part of the board we must solder a 5 pin SIL type (single in line) header for the interface of the board with the MCU and its powering needs. The signals available at the header are +5 volts, two grounds and the control ports SDA (serial data) and SCL (serial clock). These are clearly labeled on the top side of the board. On the top side I also soldered a 3 pin SIL header so that the appropriate power level for the level translators can be jumper selected (for this specific project 5 volts as all the Arduinos used operate at 5 volts). The Si5351A ic is being controlled through a MCU (i.e. the Arduino in our case) through the 2-wire interface I2C serial protocol.

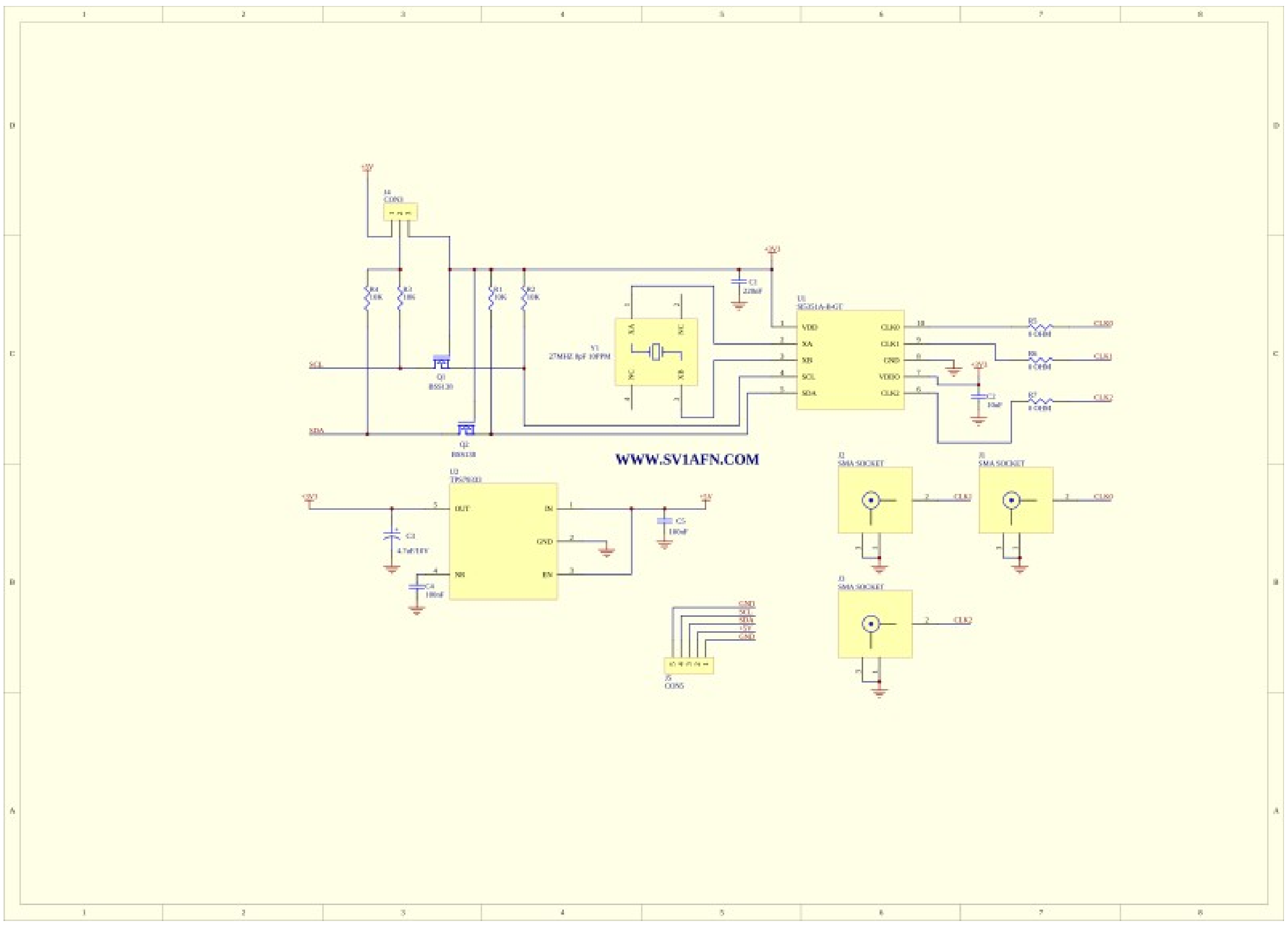




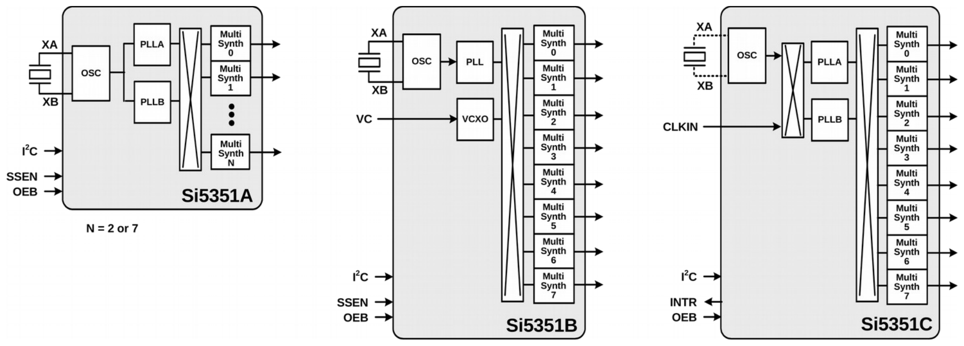
The cost of this excellent board that has been designed and sold from our fellow ham Makis, SV1AFN (www.sv1afn.com) is quite low as it can be mail ordered at 8,90 Euro plus postage, including the three gold plated SMA connectors!
This is the schematic of the board:

Theory of Operation

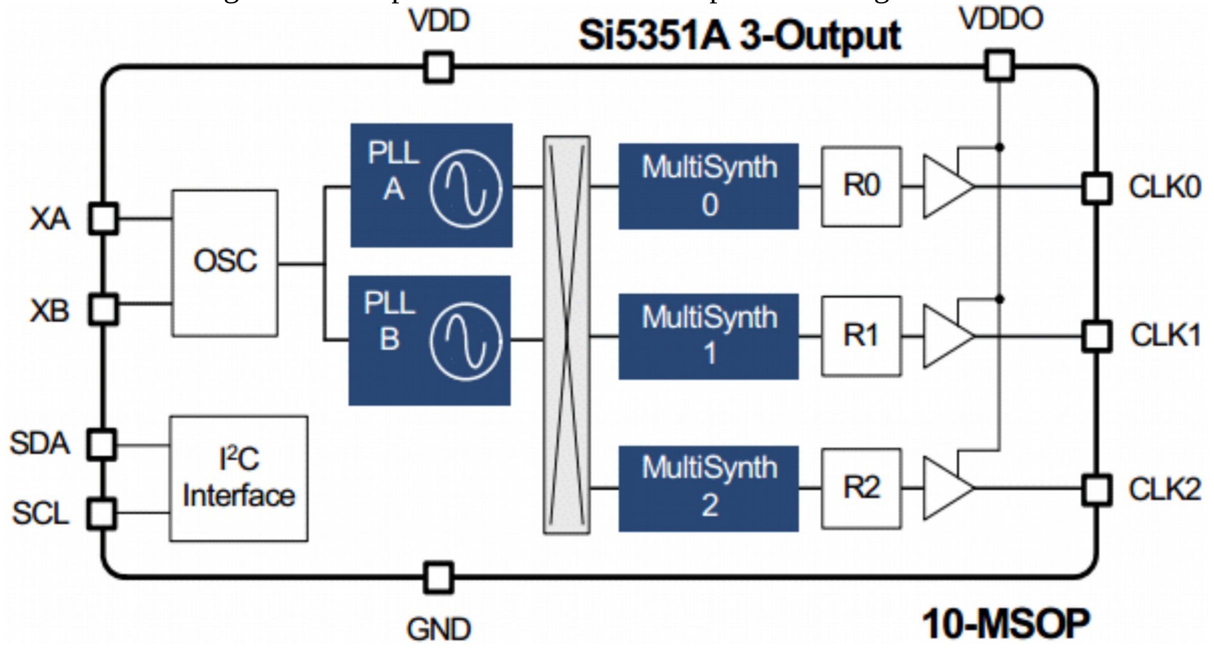
Si5351A from SiLabs can be considered as a “cousin” of the very well known Si570, but at a much lower cost. Contrary to Si570 the basic difference of Si5351A is that it does not have an internal oscillator. Therefore ian external reference oscillator or crystal or crystal oscillator having a frequency of 25 or 27 MHz needs to be connected to the ic.
This specific “little” board from SV1AFN uses, as it can be seen on the schematic, a special crystal on 27 MHz that has an accuracy of 10 ppm and a loading capacitance of 8 pf. That frequency compared to 25 MHz enables us to program the ic on the exact tone spac of 1.46 Hz which is



needed for the WSPR digital mode in every amateur band (should we wish to deal with that mode!).
The Si5351 family has three different versions of integrated circuits with a different number of outputs as SiLabs designed it for crystal substitution in designs where more than 3 crystals were required.

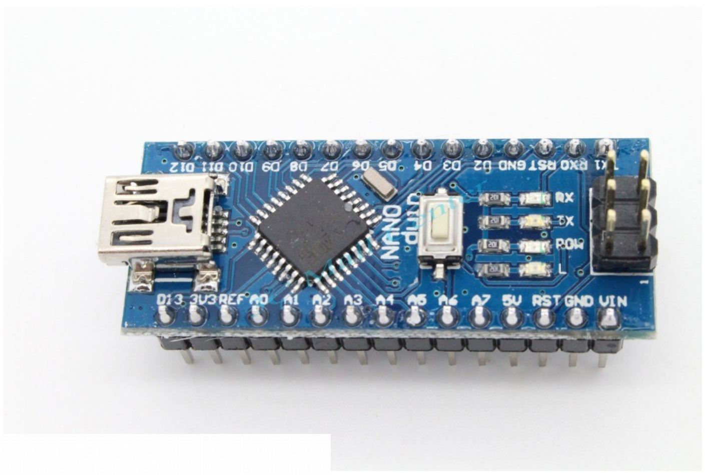


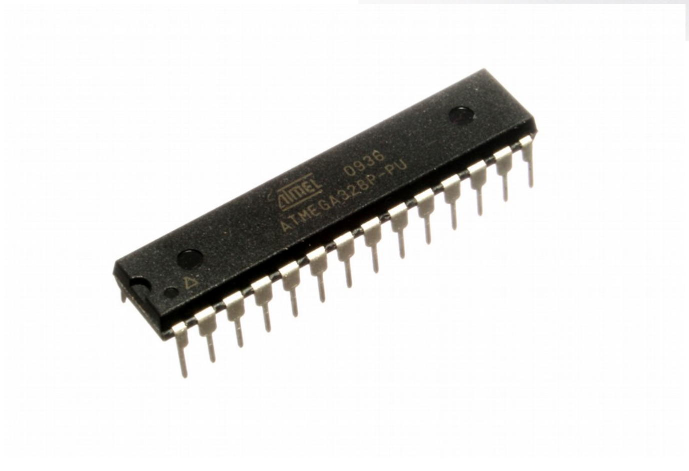
This is the block diagram of the specific version of the chip that is being used in SV1AFNs board:



The packaging of this ic is 10-MSOP with 10 pins (as the name denotes) and is only 3 by 3 millimeters. My eyes felt very relaxed, as the chip and the rest of the SMD components were already soldered on the board that is provided by SV1AFN. So I only had to solder the 5-pin and 3-pin male headers.
The 27 MHz crystal is connected on pins XA and XB. Pins SDA and SCL are the ports for the communication of the chip with the MCU and its programming through the I2C interface. The 3 outputs are brought out on the pcb to the relative pads labeled J1, J2, J3, where you can solder the SMA connectors. These correspond to outputs CLK0, CLK1 and CLK2 respectively.
The Si5351A chip needs to be powered with 3,3 volts and not 5 volts but this is the task of the on board regulator chip, so that we can supply the board with 5 volts from the Arduino board.
Briefly, the Si5351A chip synthesizes output frequencies in three stages. First, a crystal reference oscillator is multiplied up to internal frequency in the range of 600-900 MHz. There are two PLL's (Phase Locked Loops) in the chip, each can be selected to choose a different internal frequency. Then on the second stage, one of the PLL frequencies is divided down to each of up to the 3 available outputs CLK0, CLK1, CLK2. These can be selected to be turned on or off through programming. Both the upward multiplication to the internal PLL output frequency, and the down division to the desired output frequency use the method of fractional ratios. That includes an integer and a fractional part that incorporates a 20-bit nominator and denominator. There is an optional third division stage that may be software configured to divide each output by a power of 2, up to a maximum division ratio of 128. That stage can be used if we want to generate low frequency outputs in the range of 8 Hz to 500 Khz.
There are far more other capabilities on this frequency synthesis chip that fall out from the scope of this construction. Should there be interest on additional chip features, then the chip's data sheet and application note AN619 should be downloaded from SiLabs site for further study.
It is possible to produce frequencies up to the level of 155 MHz. However as we only use the method of Multisynth division and avoid the activation of the third stage that is being available with the chip, so that we can have glitch free produced pulses, the limits of the produced frequencies are limited from 1 MHz to 112,5 MHz.
We should not forget that the 3 outputs produce only square wave with even harmonics and that the phase noise is higher compared to other programmable frequency synth chips. Nevertheless it works well with a number of mixers and in particular with those of the SA602/612 active type.
Although the said limitations, due to its low cost it has started to be used by an increasing number of Hams, especially in QRP projects.

As I previously mentioned, in order to program the Si5351A serially through the I2C bus we need a microcontroller (MCU).
Taking into consideration again what can be easily sourced nowadays especially in our local market, I ended up with a microcontroller of the “Arduino” type. I thought that some Hams may already have an Arduino Uno or Nano on their desk . As I had both of them in hand, I tried with both. Then when I had the project up and running, I tried also with a barebones standalone ATmega328P-PU. That is the chip that most of the boards is being used in the majority of the boards that are referred as Arduino boards.

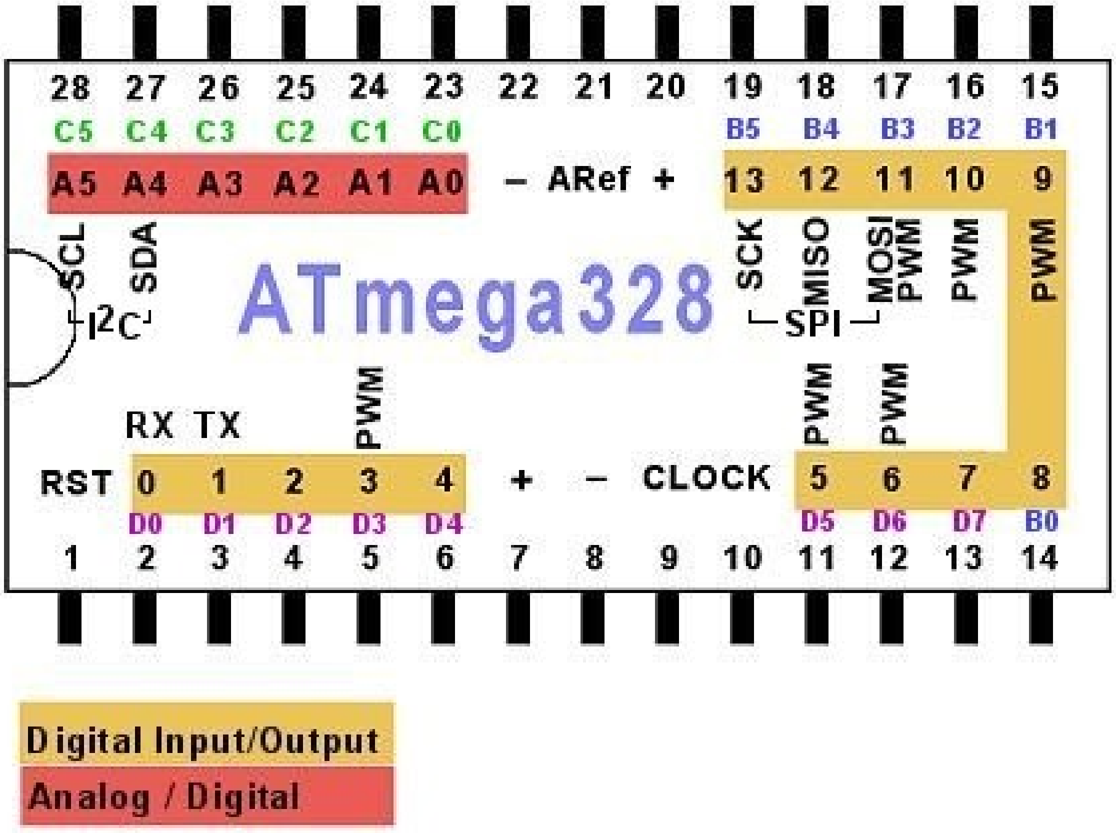




ATmega328P-PU ic.

The “heart” of the MCU system!

I continue with a short tutorial about the microcontroller portion of the project. Developed back in 2005 in Italy to help electronics students learn programming on an easy environment, Arduino is common open programing environment and development tool that enables us to easily create electronic applications that have as their core a “common” microcontroller (MCU) that is of the Atmel Atmega family.
Those boards with the microcontroller are usually attached to a programing environment that runs on our personal computer or laptop using a serial interface port of the USB type (but not only that allows the programing of our Arduino board from our computer. In order to communicate (“talk”) with the development system on our computer the microcontroller has installed in a small portion of its internal memory (0,5 Kbytes to 2 Kbytes) a small program, called the bootloader that enables its communication with the computer system.
When we have loaded our program to the Arduino board, it can work autonomously and perform the tasks that we have programmed it to perform, operating for this purpose from an independent power source . The microcontroller has a number of digital and analog ports that can be configured as i/o's (inputs/outputs). Various types of sensors, switches and other peripheral devices that communicate with the microcontroller through the I2C serial protocol can be connected to those ports.

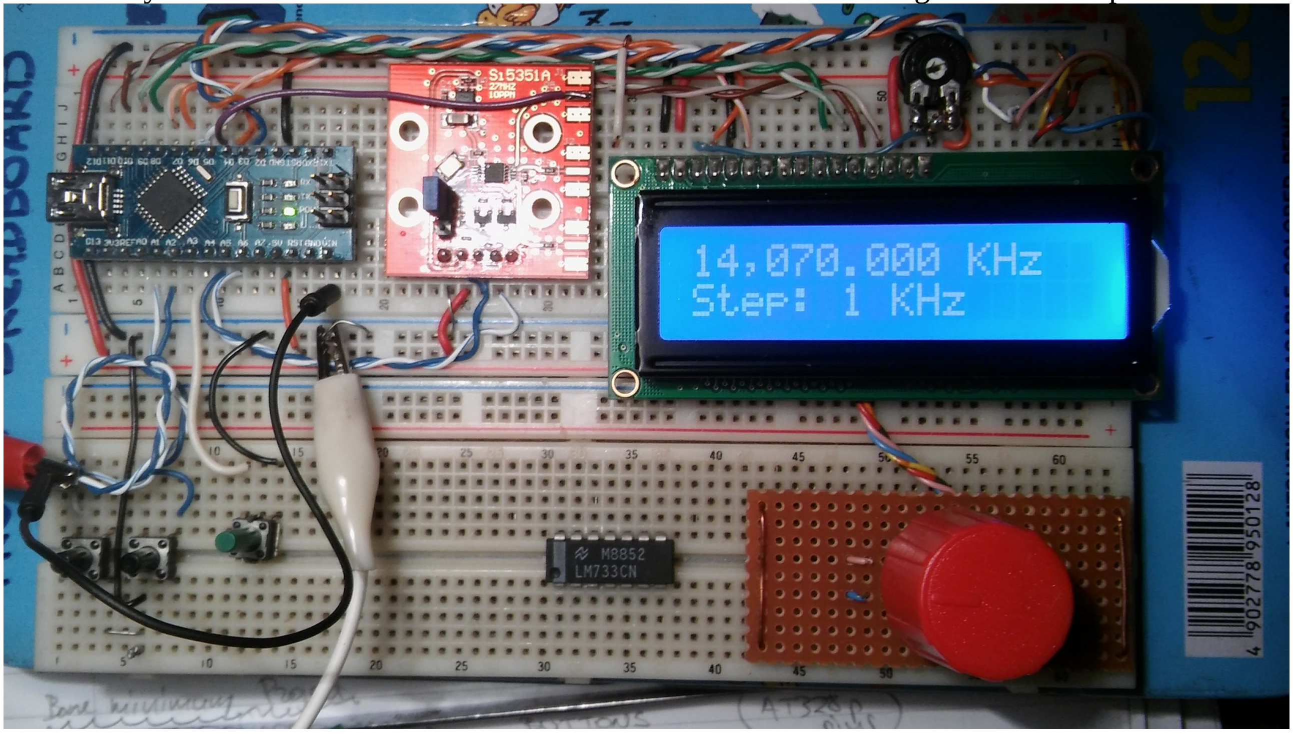


This is the outline schematic of the ATmega328P-PU chip.
Finally in order to read the tasks or the results that we have programed the microcontroller to perform, we can connect a LCD (Liquid Crystal Display) to it.

This is the end of my very elementary crash course on the system that we are going to use to control the Si5351A chip. The display that I use is of the 1602 type (backlit with 2 lines by 16 characters). For sourcing it in Greece for example, follow the link http://grobotronics.com/basic-16x2-character-lcd-black-on-gray.html
There are tons of information about Arduino on the web, as well as an incredible amount of d.i.y.

projects. There is also a very strong Arduino forum where you can practically find the answer for whatever comes up to your mind. Just search for it. You can start here https://forum.arduino.cc/ . Concerning how to install the software on your computer, the Arduino IDE – Integrated Development Environment and how to load a program (called a sketch in Arduino nomenclature) on the microcontroller board, I strongly recommend that you follow one of the many tutorials, based on the Operating System that you use. Simply google “installing arduino on xxxxx operating system” and you will be rewarded.

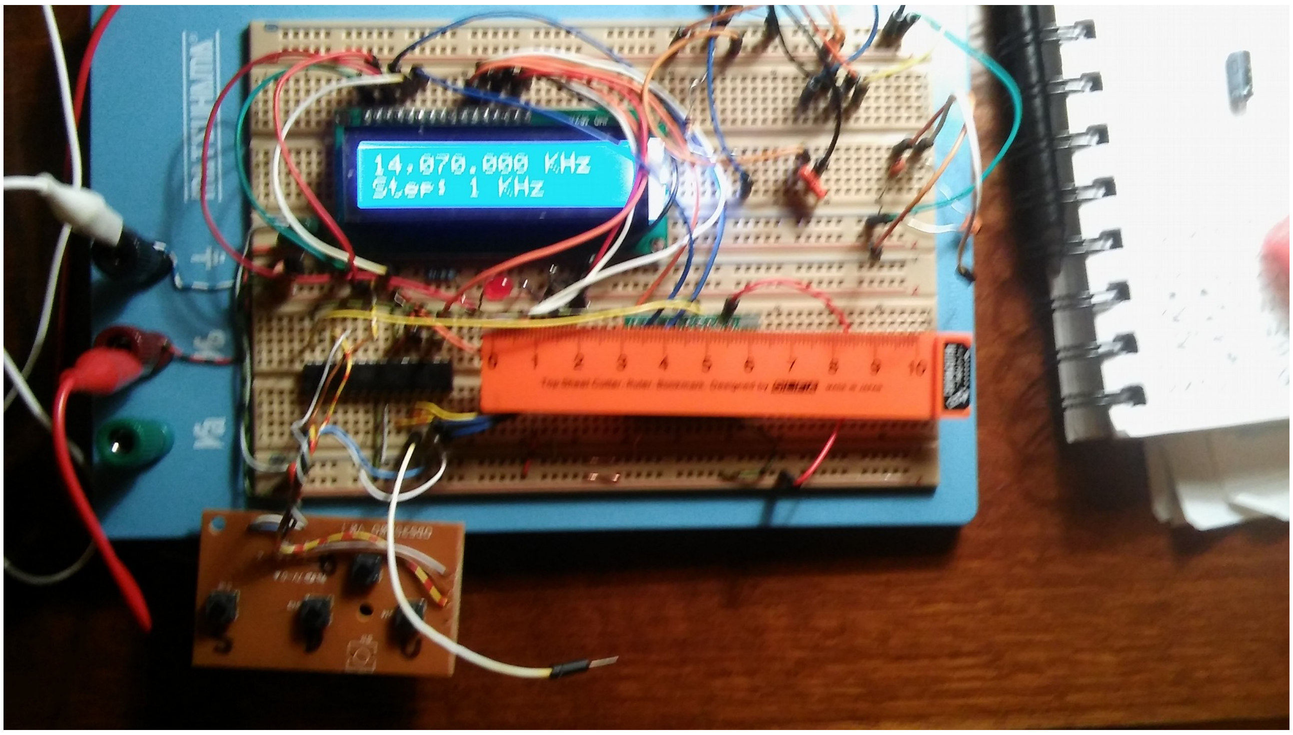
I will now proceed with the construction details.
Except from the Arduino board with the microcontroller and the LCD we will need 4 push buttons (single pole). I salvaged mine from old electronic devices. Before I recycle them, I remove all the useful components that can be reused. We will also need a small 10 K trimmer potentiometer, a rotary encoder (optional but preferable by many), 2 100nF capacitors and a proto breadboard or a perforated board if we want to solder the various subsystems (LCD, Si5351A board, microcontroller etc.). Depending on the version of the project we may additionally need 3 more 100nF, 1 10K 1/8 watt resistor and a 7805 regulator with a small heatsink. Those additional components may be needed only if we decide to build the barebones version with the ATmega328P-PU chip.



That is how the project should look like. No I did not forget the LM733CN chip. That was a leftover on the prototyping board from another project!

For this version of the construction I used an Arduino Nano that I directly pinned on the breadboard. The Nano version of Arduino is of small dimensions (42 by 16 millimeters) and has two series of male headers with 15 pins each resembling a large 30 pin integrated circuit. I also pinned the LCD in the same way. Most of the 1602 type are of universal type with the same pinout. So you can solder a SIL 16 pin male header on it. The board with Si5351A has a male 5-pin header, the rotary encoder that I used had 5 pins – contacts A, B, ground and two more contacts for a co-centric push button which can be used to replace the separate Step button. In that case we need a button less. The trimmer pot is a standard one (not multi-turn) and has three legs, while the push buttons have two or four. In the case of the salvaged buttons they had four legs, but each two of them were internally short circuited. That can be verified with an ohm meter. All the push buttons are of simple push action.
Of course you will need an adequate number of single solid wire in as many colors as possible to

make the connections on the breadboard. UTP wire is suitable for firm breadboarding.
When I program the Arduino board from the USB port the power supply to the board is provided from the very same USB connector (5 volts). Then when I have finished the uploading of the sketch to the Arduino and I am ready to use the board as an autonomous device I need to power the whole arrangement on the breadboard from a source (battery or power supply) that provides 9 to 12 volts. Then pin1 is the +Vin and pin2 is Ground on the Nano board. That board also has a built-in regulator that provides the required 5 volts for the processor chip.
Although this is the fifth project that I have built with the Si5351A chip, I decided to use the very well written program of our fellow Ham Gene Marcus W3PM, which I modified to suit the needs of my project. Therefore the credits for the software go to him.
I tested my project also with Arduino Uno V3 receiving signals on the HF bands and it worked satisfactorily.
As I had some blank ATmega328P-PU chips, I decided to test an absolute minimal version on a breadboard. I did not use even the rotary encoder as the software can change frequency upwards or downwards by means of two of the four push button. I even removed the external 16 MHz crystal that is necessary for the chip to operate, as I enabled the built-in internal oscillator that operates at half speed (8 MHz). The chip performed well without any problem.



In this setup we only have 4 push buttons that operate as follows:

S(tep) button. It defines the step of the frequency synthesis. It starts with an initial value of 1 Khz and progress cyclically to 10 Khz, 100 Khz, 1MHz, 1 Hz, 10 Hz, 100 Hz, 1 Khz and so on.

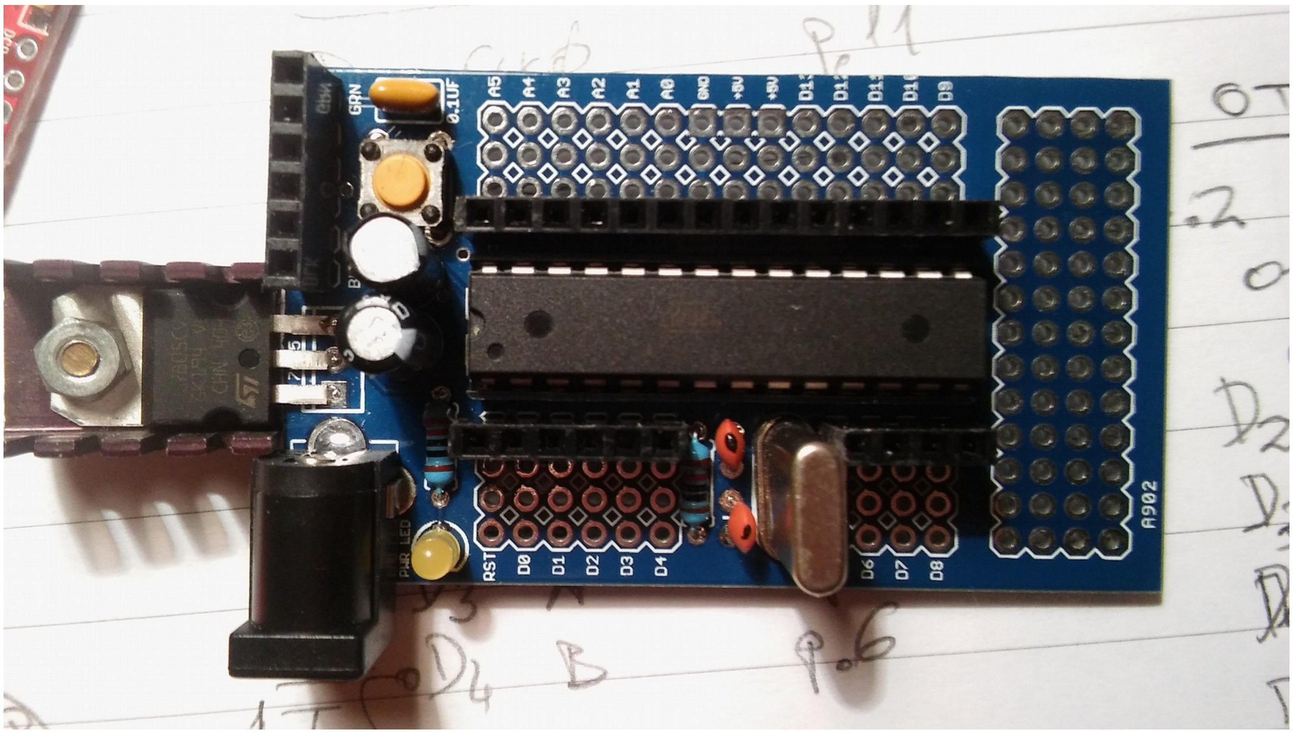
D(own) button. It decreases the frequency according to the step that we have defined with the previous button. It stops to 1MHz (lower limit).

U(p) button. It increases the frequency according to the step that we have defined with the S(tep) button. It stops to 112,5 MHz (upper limit).

B(and) button. It selects a predefined by the software band and an initial frequency value within the band from where we will start moving upwards or downwards. The band selection sequence and the initial frequency value can be changed through the program. For the shake of this project I

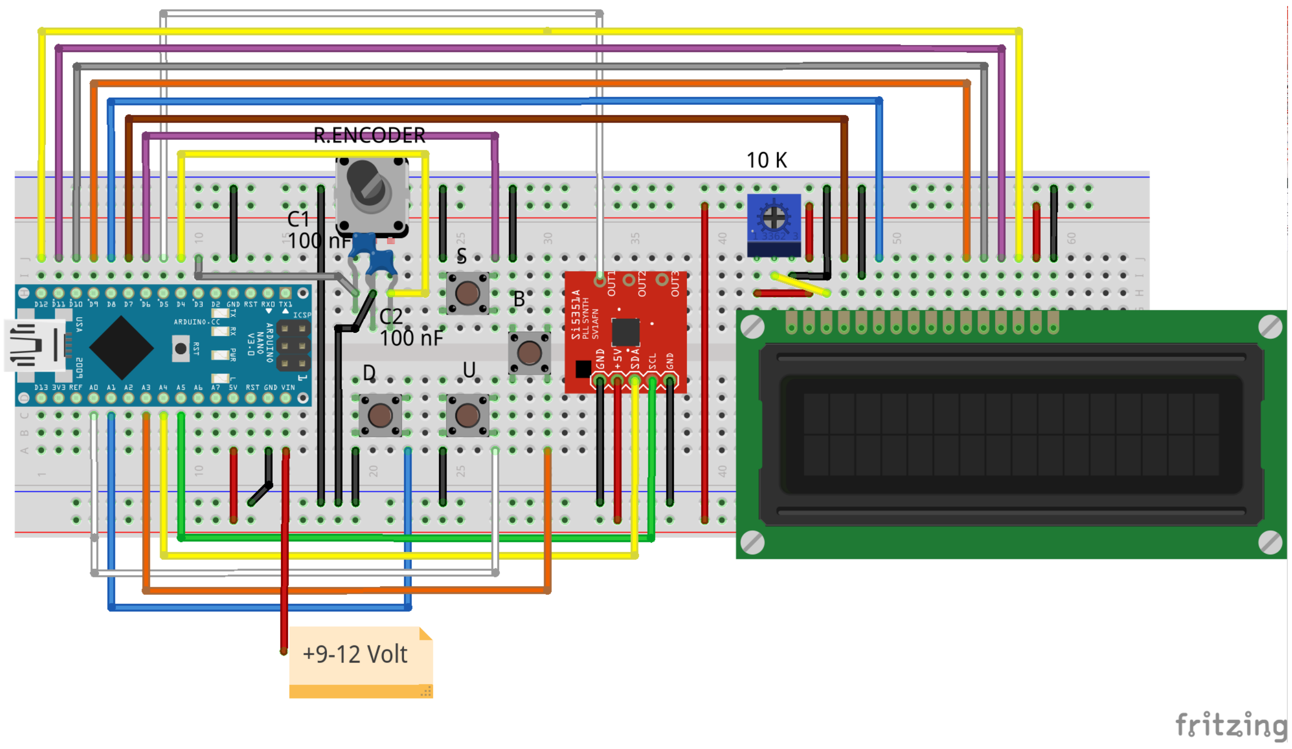
programmed some QRP frequencies with the following order:
14,070.000 PSK frequency, if offset 0. To test Direct Conversion receivers.
14,060.000 QRP CW frequency, if offset 0 (if= intermediate frequency)
  7,030.000 QRP CW frequency, if offset 0
  3,560.000 QRP CW frequency, if offset 0
  1,843.000 QRP CW frequency, if offset 0
10,106.000 QRP CW frequency, if offset 0
18,086.000 QRP CW frequency, if offset 0
21,060.000 QRP CW frequency, if offset 0
24,906.000 QRP CW frequency, if offset 0
28,060.000 QRP CW frequency, if offset 0
50,060.000 QRP CW frequency, if offset 0
All the above produce an Output on CLK1. CLK2 = 0
14,285.000 QRP SSB frequency, vfo 11,008.200 (CLK1), carrier osc 3,276.800 (CLK2) *
  7,090.000 QRP SSB frequency, vfo 12,005.200 (CLK1), carrier osc 4,915.200 (CLK2) *
  7,185.000 Local SSB call frequency, vfo 12,110.200 (CLK1), carrier osc 4,915.200 (CLK2) *
27,000.000 Frequency to check external crystal accuracy on CLK1.
CLK2 = 0 and CLK0 = 2,700.000 MHz (1/10 of CLK1).

The first frequencies on the above table are the ones that are displayed on the screen when we select each band. Then we can increase or decrease frequency according to the selected S(tep) using either the U(p) and D(own) push buttons or the rotary encoder if we decide to have it incorporated in our project.
The values for the 3 last frequencies marked with asterisk include vfo and carrier oscillator values. They have been selected by me to cover my personal needs during the tests. If they are meaningless to you they can be erased or substituted with other values. You can do that from the sketch before you have it uploaded to the Arduino board or our AT mega chip.



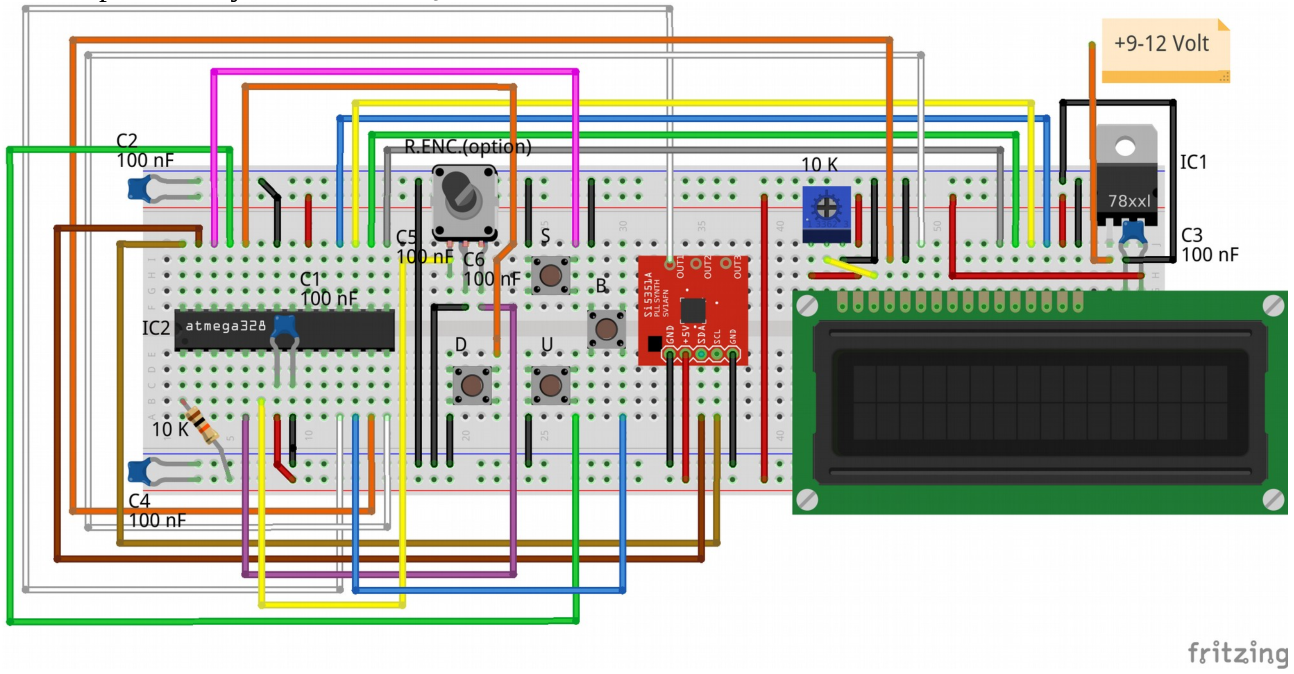
And here is another different version. ATmega328P-PU chip mounted on ready made Arduino experimenting pcb. I had found it in a local hobby shop some months ago. Again as I have activated the internal oscillator on the chip the external 16 MHz crystal and the two 22pF capacitors that I had soldered on the board for other experiments become redundant. I just left them in place for future use. Notice that there is a 7805 regulator with a heat-sink and the necessary capacitors as well as a reset switch.

The schematic and the connection of the components for my various construction versions remains the same. Only the pin numbering changes among the versions as shown below. The drawing has been made with Fritzing software.



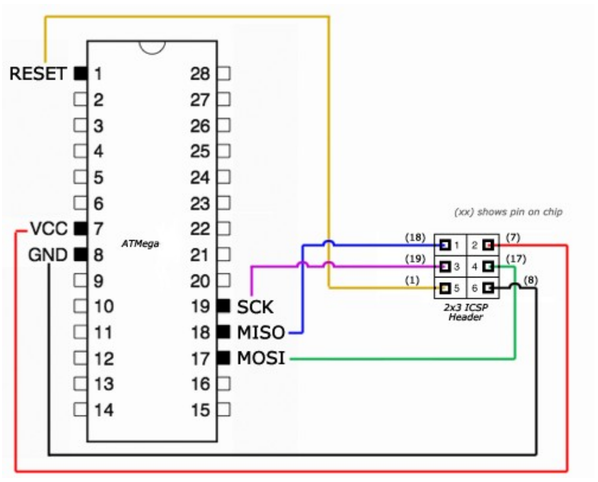
Just some points on the schematics:
Output CLK0 on the Si5351A board goes to digital i/o D5 of Arduino with a cable that I solderd carefully. In this current phase I decided not to use the SMA connectors yet.

If we ground analog input A2 of Arduino then the produced frequencies of output CLK1 are shifted according to predefined +/- offset value in Hertz. This has been initially set to -600 Hz in the sketch and can be altered positively or negatively to suit our needs before we upload it to our Arduino. That is particularly useful for a QRP CW transceiver.



In case of the circuit with ATmega328P-PU chip one could reasonably ask how the chip could be programed?

The sketch is compiled with the Arduino IDE and is then uploaded to the chip by using an external programmer. There is such a selection on the Arduino IDE software.
All Atmel microcontrollers (and those of other manufacturers) can be programmed “in circuit” by using the ICSP (In Circuit Serial Programming) bus/protocol. That enables the connection of the external programmer to certain pins of the microcontroller. For programmer I use either of the two Arduino boards that I have (Uno or Nano) by uploading the appropriate sketch (ArduinoISP) that is included in the Examples of the Arduino IDE. Uno and Nano boards incorporate ICSP bus as a standard on a 6-pin DIL connector.



Therefore should we wish to program the ATmega328P-PU chip of the second circuit we should make the following connections. They can be made temporarily or permanently on your circuit board. The choice is yours.

Useful tip. If we have a Uno board, we can alternatively remove the original ATmega328-P chip that comes with the board and replace it temporarily with the new chip that we want to program. To do that the chip that we want to program must have a bootloader. The program is the uploaded normally through the USB bus.
ATmega328P-PUwith Optiboot bootloader can be found in our market for 5,20 Euro (http://grobotronics.com/atmega328-with-arduino-optiboot-el.html), while without bootloader 3,49 Ευρώ (http://grobotronics.com/atmega328p-pu.html). Chips can be sourced cheaper on the web. The latest being a bit more trickier to program. But the resume is that we can put a bootloader to an ATmega328P-PU from the menu of the Arduino IDE programing environment using the programmer. If the chip is going to be used only for the specific project there is no need to use a bootloader, provided that you know how to program the chip. You can do it with information found on the Arduino forum.
But again such issues can make you deviate from the primary scope of the current project.
The sketch for the construction of this project can be downloaded from https://github.com/sv1onw/si5351a-gen . It does not require the installation of any additional external libraries other than the ones that are being supplied with Arduino IDE when you have it installed in your computer. I use version 1.6.5 of it.

For the calibration of our Generator we need the following procedure:

We connect our output CLK1 of our Generator to a Frequency meter.
We select the preset frequency of 27 MHz (last band with the B(and) push button) .
or we can go to that frequency through the use of U(p)/D(own) buttons or Rotary Encoder.
We take a note of the measured frequency with the highest possible accuracy.
We subtract the nominal 27,000.000 MHz frequency of our crystal from the Frequency meter measurement.
We take a note of the difference in Hz. (e.g. -341)..

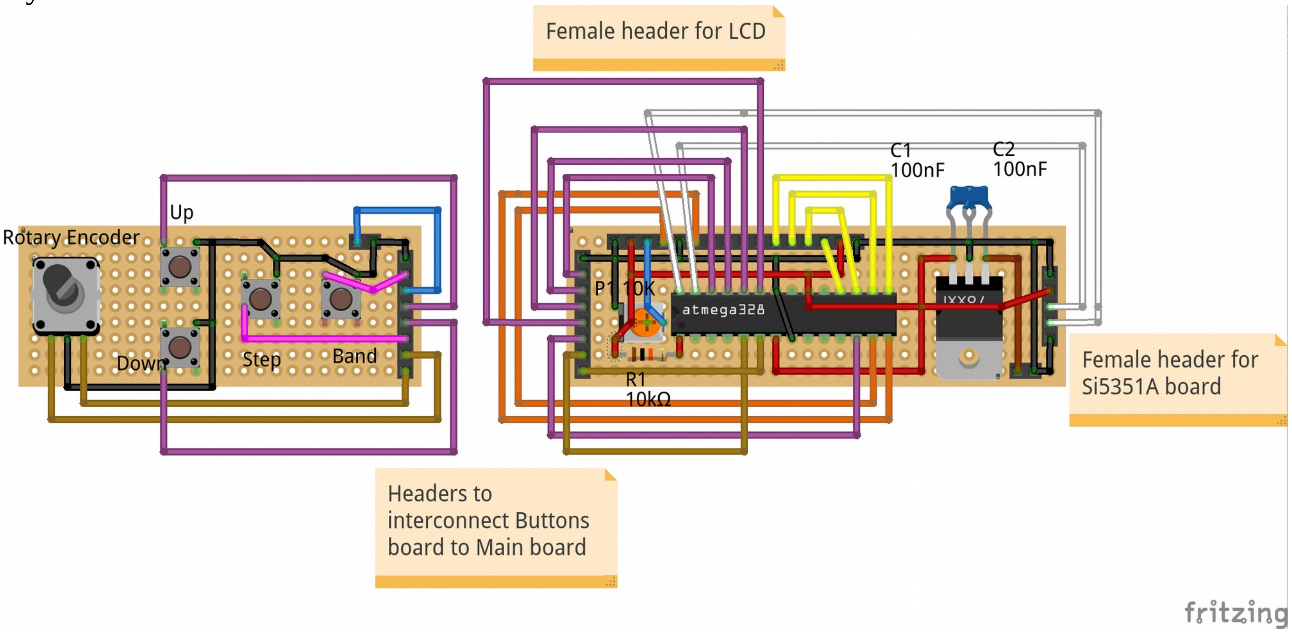
We go to the sketch and find variable “CalFactor” that has an initial zero (0) value.
We replace that value with the calculated difference.
We upload again the sketch with the corrected value.

I will also briefly refer to the sketch table where the initial frequencies for every band we select through the B(and) button are being defined.
The code is as follows:

```
 ___________Enter Band Select frequencies below_____________________
 */
const unsigned long Freq_array [] [3] = {
  { 14070000,0,0      },          // CLK1=14.070 MHz, CLK2=0 MHz, Display=14,070.000 KHz
  { 14060000,0,0      },
  {   7030000,0,0      },
  {   3560000,0,0      },
  {   1843000,0,0      },
  { 10106000,0,0      },
  { 18086000,0,0      },
  { 21060000,0,0      },
  { 24906000,0,0      },
  { 28060000,0,0      },
  { 50060000,0,0      },
  { 3276800,11008200,1 },      // CLK1:3,276800 MHz, CLK2:11,082200 MHz, Display:14,285.000 KHz
  { 4915200,12005200,2 },      // CLK1:4,915200 MHz, CLK2:12,005200 MHz, Display:7,090.000 KHz
  { 4915200,12100200,2 },      // CLK1:4,915200 MHz, CLK2:12,100200 MHz, Display:7,185.000 KHz
  { 27000000,0,0      },          //  CLK1 has the Calibration Frequency of 27 MHz for easy selection
  (0,0,0)
  };
```

This is a three dimensional array. Each line has enclosed in curly brackets three columns (positions) separated by commas. Every line, after the closing of the curly bracket is terminated with a comma. The last line of the table ends with 3 zeros within curly brackets. Any text that follows the two slashes (like in the first or last line) is optional comments. If we want we can add more lines before the last line with the three zeros {0,0,0}, . Every time that we press the B(and) button we move to the next position (line) of the array, starting always from the values that we have put in the first line when we power up the Generator. If we delete some delimiter by mistake, the program will not compile.
In the first column of each line we place (without commas or points) the frequency that we want our Generator to produce in output CLK1 (VFO variable frequency) and on the second column if we need the frequency that will be produced from output CLK2 (LO fixed frequency of local oscillator, or carrier oscillator or BFO). In the third column we put one of the numbers 0 or 1 or 2. These function as arithmetic operators. Zero does not make any operation. One adds CLK0 with CLK1, therefore on our LCD we read the sum of CLK1+CLK2. If we put the number “two” it subtracts the second frequency from the first CLK1-CLK2. Comments next to the lines of the table explain each function.
So in the first line of the table I have made up for my needs, the variable frequency CLK1of the Generator (only CLK1 is continuously adjustable) is 14070000, CLK2 is 0, and there will be no arithmetic operation between CLK1 and CLK2 as the third value is 0; on the LCD we will get 14,070.000 Khz.
If we now wish to change the value of the first line, to start say from 14,125.000, our line will become { 14125000,0,0   }, . This is just the initial frequency value for each band. Then by using the U(p) and D(own) buttons or the Rotary Encoder and selecting the desired S(tep) suiting our resoloution we can move upwards or downwards. Program takes into consideration the arithmetic logic that we have defined in the specific line of the table (CLK1 is adjustable frequency, CLK2=0).
If we now wish to remove a certain band, say 50 MHz with the conditions that we have defined, we simply delete the entire line { 50060000,0,0   }, .
If we want to use our construction only as a simple Generator, we can keep only the first line of the table, defining the desired start up frequency (e.g. 1 MHz) and the last line with the three zeros and omit even the button that switches bands. It cannot be simpler than that.

We can build our design with the desired buttons even on a perforated board as you can see in the layout that follows.



This is just an alternative idea with perforated board pieces that I had in my lab.

What follows is an interconnection table with all the components, so that you can correctly wire all the connections.

Interconnecting Table for all components

LCD	**ARDUINO**	**ATmega328P-PU**	**Other component**
1	GND	8 and 22	
2	+5V	7 and 20	
3			POT center leg
4	D7	13	
5	GND	8 and 22	
6	D8	14	
7	not connected	not connected	
8	not connected	not connected	
9	not connected	not connected	
10	not connected	not connected	
11	D9	15	
12	D10	16	
13	D11	17	
14	D12	18	
15	+5V	7 and 20	
16	GND	8 and 22	

Push BUTTONS	**ARDUINO**	**ATmega328P-PU**	**Other component**
D(own)	A0	23	
U(p)	A1	24	
B(and)	A3	26	
S(tep)	D6	12	

ROTARY ENCODER	**ARDUINO**	**ATmega328P-PU**	**Other component**
A	D3	5	
B	D4	6	
GND	GND	8 and 22	

OFFSET Control	**ARDUINO**	**ATmega328P-PU**	**Other component**
GND activates	A2	25	

Si5351A MODULE	**ARDUINO**	**ATmega328P-PU**	**Other component**
1(GND)	GND	8 and 22	
2(+5V)	+5V	7 and 20	
3(SDA)	A4	27	
4(SCL)	A5	28	
5(GND)	GND	8 and 22	

OUTPUT CLK0	D5	11	
OUTPUT CLK1			output J2
OUTPUT CLK1			output J3

ICSP (Programing)	**ARDUINO**	**ATmega328P-PU**	**Other component**
RESET		1	
SCLK		19	
MISO		19	
MOSI		17	
+5V		7 and 20	
GND		8 and 22	

Trimmer Pot	**ARDUINO**	**ATmega328P-PU**	**Other component**
c.w.			GND
mid.			Pin 3 LCD
c.c.w.			+5V

Closing this article which is a free interpretation of the original article in Greek that was published in the December issue of SV-QRP electronic magazine, I want to make known that except the genuine Arduino Uno V3 and Arduino Nano V3 boards that cost about 30 to 35Euro, there are Chinese clones that are much cheaper (5 – 7 Euro). The choice of what to use is of course yours.
Wishing good luck with your project, Konstantinos SV1ONW.