Si5351 Library for Arduino

This is a library for the Si5351 series of clock generator ICs from Silicon Labs for the Arduino development environment. It will allow you to control the Si5351 with an Arduino, and without depending on the proprietary ClockBuilder software from Silicon Labs.

This library is focused towards usage in RF/amateur radio applications, but it may be useful in other cases. However, keep in mind that coding decisions are and will be made with those applications in mind first, so if you need something a bit different, please do fork this repository.

Please feel free to use the Issues feature of GitHub if you run into problems or have suggestions for important features to implement. This is the best way to get in touch.

Thanks For Your Support!

If you would like to support my library development efforts, I would ask that you please consider purchasing a Si5351A Breakout Board from my online store at etherkit.com and/or sending a PayPal tip. Thank you!

Library Installation

The best way to install the library is via the Arduino Library Manager, which is available if you are using Arduino IDE version 1.6.2 or greater. To install it this way, simply go to the menu Sketch > Include Library > Manage Libraries..., and then in the search box at the upper-right, type "Etherkit Si5351". Click on the entry in the list below, then click on the provided "Install" button. By installing the library this way, you will always have notifications of future library updates, and can easily switch between library versions.

If you need to or would like to install the library in the old way, then you can download a copy of the library in a ZIP file. Download a ZIP file of the library from the GitHub repository by using the "Download ZIP" button at the right of the main repository page. Extract the ZIP file, then rename the unzipped folder as "Si5351". Finally, open the Arduino IDE, select menu Sketch > Import Library... > Add Library..., and select the renamed folder that you just downloaded. Restart the IDE and you should have access to the new library.

Hardware Requirements and Setup

This library has been written for the Arduino platform and has been successfully tested on the Arduino Uno and an Uno clone. There should be no reason that it would not work on any other Arduino hardware with I2C support.

The Si5351 is a +3.3 V only part, so if you are not using a +3.3 V microcontroller, be sure you have some kind of level conversion strategy.

Wire the SDA and SCL pins of the Si5351 to the corresponding pins on the Arduino. Use the pin assignments posted on the Arduino Wire library page. Since the I2C interface is set to 100 kHz, use 1 to 10 kΩ pullup resistors from +3.3 V to the SDA and SCL lines.

Connect a 25 MHz or 27 MHz crystal with a load capacitance of 6, 8, or 10 pF to the Si5351 XA and XB pins. Locate the crystal as close to the Si5351 as possible and keep the traces as short as possible. Please use a SMT crystal. A crystal with leads will have too much stray capacitance.

Changes from v1 to v2

The public interface to the v2 library is similar to the v1 library, but a few of the most-used methods have had their signatures changed, so your old programs won't compile right out-of-the-box after a library upgrade. Most importantly, the *init()* and *set_freq()* methods are different, so you'll at least need to change these calls in your old sketches.

The *init()* method now has three parameters: the crystal load capacitance, the reference frequency, and the frequency correction value (with this last parameter being a new addition). You'll need to add that third parameter to your old *init()* calls, but then you can delete any *set_correction()* calls after that (unless you explicitly are changing the frequency correction after the initialization).

The *set_freq()* method is now more streamlined and only requires two parameters: the desired output frequency (from 4 kHz to 225 MHz) and clock output. In your old code, you can delete the 2nd parameter in *set_freq()*, which was the PLL frequency. In case you want to do things manually, there is now a new method called *set_freq_manual()* (see below for details).

Those two changes should cover nearly all upgrade scenarios, unless you were doing some lower-level use of the Si5351.

Example

First, install the Si5351Arduino library into your instance of the Arduino IDE as described above.

There is a simple example named **si5351_example.ino** that is placed in your examples menu under the Si5351Arduino folder. Open this to see how to initialize the Si5351 and set a couple of the outputs to different frequencies. The commentary below will analyze the sample sketch.

Before you do anything with the Si5351, you will need to include the "si5351.h" and "Wire.h" header files and instantiate the Si5351 class.

```
#include "si5351.h"
#include "Wire.h"

Si5351 si5351;
```

Now in the *Setup()* function, let's initialize communications with the Si5351, specify the load capacitance of the reference crystal, that we want to use the default reference oscillator frequency of 25 MHz (the second argument of "0" indicates that we want to use the default), and that we will apply no frequency correction at this point (the third argument of "0"):

```
i2c_found = si5351.init(SI5351_CRYSTAL_LOAD_8PF, 0, 0);
```

The *init()* method returns a *bool* which indicates whether the Arduino can communicate with a device on the I2C bus at the specified address (it does not verify that the device is an actual Si5351, but this is useful for ensuring that I2C communication is working).

Next, let's set the CLK0 output to 14 MHz:

```
si5351.set_freq(1400000000ULL, SI5351_CLK0);
```

Frequencies are indicated in units of 0.01 Hz. Therefore, if you prefer to work in 1 Hz increments in your own code, simply multiply each frequency passed to the library by 100ULL (better yet, use the define called *SI5351_FREQ_MULT* in the header file for this multiplication).

In the main *Loop()*, we use the Serial port to monitor the status of the Si5351, using a method to update a public struct which holds the status bits:

```
si5351.update_status();
Serial.print("SYS_INIT: ");
Serial.print(si5351.dev_status.SYS_INIT);
Serial.print("  LOL_A: ");
Serial.print(si5351.dev_status.LOL_A);
Serial.print("  LOL_B: ");
Serial.print(si5351.dev_status.LOL_B);
Serial.print("  LOS: ");
Serial.print(si5351.dev_status.LOS);
Serial.print("  REVID: ");
Serial.println(si5351.dev_status.REVID);
```

When the synthesizers are locked and the Si5351 is working correctly, you'll see an output similar to this one (the REVID may be different):

```
SYS_INIT: 0  LOL_A: 0  LOL_B: 0  LOS: 0  REVID: 3
```

The nominal status for each of those flags is a 0. When the program indicates 1, there may be a reference clock problem, tuning problem, or some kind of other issue. (Note that it may take the Si5351 a bit of time to return the proper status flags, so in program initialization issue *update_status()* and then give the Si5351 a few hundred milliseconds to initialize before querying the status flags again.)

A Brief Word about the Si5351 Architecture

The Si5351 consists of two main stages: two PLLs which are locked to the reference oscillator (a 25/27 MHz crystal) and which can be set from 600 to 900 MHz, and the output (multisynth) clocks which are locked to a PLL of choice and can be set from 500 kHz to 200 MHz (per the datasheet, although it does seem to be possible to set an output up to 225 MHz).

The B variant has an additional VCXO stage with control voltage pin which can be used as a reference synth for a clock output (PLLB must be used as the source for any VCXO output clock).

The C variant is able to take a reference clock input from 10 to 100 MHz separate from the standard crystal reference. If using this reference input, be sure to initialize the library with the correct frequency.

This library makes PLL assignments based on ease of use. They can be changed manually if needed, although that can introduce complications (see *Manually Selecting a PLL Frequency* below).

Setting the Output Frequency

As indicated above, the library accepts and indicates clock and PLL frequencies in units of 0.01 Hz, as an *unsigned long long* variable type (or *uint64_t*). When entering literal values, append `ULL` to make an explicit unsigned long long number to ensure proper tuning. Since many applications won't require sub-Hertz tuning, you may wish to use an *unsigned long* (or *uint32_t*) variable to hold your tune frequency, then scale it up by multiplying by 100ULL before passing it to the *set_freq()* method.

Using the *set_freq()* method is the easiest way to use the library and gives you a wide range of tuning options, but has some constraints in its usage. Outputs CLK0 through CLK5 by default are all locked to PLLA while CLK6 and CLK7 are locked to PLLB. Due to the nature of the Si5351 architecture, there may only be one CLK output among those sharing a PLL which may be set greater than 100 MHz (actually specified at 112.5 MHz by SiLabs, but stability issues have been found at the upper end). Therefore, once one CLK output has been set above 100 MHz, no more CLKs on the same PLL will be allowed to be set greater than 100 MHz (unless the one which is already set is changed to a frequency below this threshold).

If the above constraints are not suitable, you need glitch-free tuning, or you are counting on multiple clocks being locked to the same reference, you may set the PLL frequency manually then make clock reference assignments to either of the PLLs.

Manually Selecting a PLL Frequency

Instead of letting the library choose a PLL frequency for your chosen output frequency, you can choose it yourself in the *set_freq_manual()* method. This method is similar to *set_freq()*, but the second argument is the desired PLL frequency:

```
si5351.set_freq_manual(19800000000ULL, 79200000000ULL, SI5351_CLK0);
```

If you use this method (or the other methods to tweak the PLL and multisynth settings manually), it is very important to remember that the library will no longer properly track the PLL and multisynth settings and that you alone will be responsible for keeping the synths tuned properly. Strange things can happen to your other outputs if they are already in use. Be sure to read the Si5351 datasheet and Silicon Labs AN619 before doing this so that you understand what you are doing.

When you are setting the PLL manually you need to be mindful of the limits of the IC. The multisynth (MS0 through MS5) is a fractional PLL, with limits described in AN619 as:

> Valid Multisynth divider ratios are 4, 6, 8, and any fractional value between 8 + 1/1,048,575 and 900 + 0/1. This means that if any output is greater than 112.5 MHz (900 MHz/8), then this output frequency sets one of the VCO frequencies.

To put this in other words, if you want to manually set the PLL and wish to have an output frequency greater than 100 MHz (changed in this library from the stated 112.5 MHz due to stability issues which were noticed), then the choice of PLL frequency is dictated by the choice of output frequency, and will need to be an even multiple of 4, 6, or 8.

Further Details

If we like we can adjust the output drive power:

```
si5351.drive_strength(SI5351_CLK0, SI5351_DRIVE_4MA);
```

The drive strength is the amount of current into a 50Ω load. 2 mA roughly corresponds to 3 dBm output and 8 mA is approximately 10 dBm output.

Individual outputs can be turned on and off. In the second argument, use a 0 to disable and 1 to enable:

```
si5351.output_enable(SI5351_CLK0, 0);
```

You may invert a clock output signal by using this command:

```
si5351.set_clock_invert(SI5351_CLK0, 1);
```

Calibration

There will be some inherent error in the reference oscillator's actual frequency, so we can account for this by measuring the difference between the uncalibrated actual and nominal output frequencies, then using that difference as a correction factor in the library. The *init()* and *set_correction()* methods use a signed integer calibration constant measured in parts-per-billion. The easiest way to determine this correction factor is to measure a 10 MHz signal from one of the clock outputs (in Hz, or better resolution if you can measure it), scale it to parts-per-billion, then use it in the *set_correction()* method in future use of this particular reference oscillator. Once this correction factor is determined, it should not need to be measured again for the same reference oscillator/Si5351 pair unless you want to redo the calibration. With an accurate measurement at one frequency, this calibration should be good across the entire tuning range.

The calibration method is called like this:

```
si5351.set_correction(-6190, SI5351_PLL_INPUT_XO);
```

However, you may use the third argument in the *init()* method to specify the frequency correction and may not actually need to use the explict *set_correction()* method in your code.

A handy calibration program is provided with the library in the example folder named *si5351_calibration*. To use it, simply hook up your Arduino to your Si5351, then connect it to a PC with the Arduino IDE. Connect the CLK0 output of the Si5351 to a frequency counter capable of measuring at 10 MHz (the more resolution, the better). Load the sketch then open the serial terminal window. Follow the prompts in the serial terminal to change the output frequency until your frequency counter reads exactly 10.000 000 00 MHz. The output from the Arduino on your serial terminal will tell you the correction factor you will need for future use of that reference oscillator/Si5351 combination.

One thing to note: the library is set for a 25 MHz reference crystal. If you are using a 27 MHz crystal, use the second parameter in the *init()* method to specify that as the reference oscillator frequency.

Phase

*Please see the example sketch **si5351_phase.ino***

The phase of the output clock signal can be changed by using the set_phase() method. Phase is in relation to (and measured against the period of) the PLL that the output multisynth is referencing. When you change the phase register from its default of 0, you will need to keep a few considerations in mind.

Setting the phase of a clock requires that you manually set the PLL and take the PLL frequency into account when calculation the value to place in the phase register. As shown on page 10 of Silicon Labs Application Note 619 (AN619), the phase register is a 7-bit register, where a bit represents a phase difference of 1/4 the PLL period. Therefore, the best way to get an accurate phase setting is to make the PLL an even multiple of the clock frequency, depending on what phase you need.

If you need a 90 degree phase shift (as in many RF applications), then it is quite easy to determine your parameters. Pick a PLL frequency that is an even multiple of your clock frequency (remember that the PLL needs to be in the range of 600 to 900 MHz). Then to set a 90 degree phase shift, you simply enter that multiple into the phase register. Remember when setting multiple outputs to be phase-related to each other, they each need to be referenced to the same PLL.

You can see this in action in a sketch in the examples folder called *si5351phase*. It shows how one would set up an I/Q pair of signals at 14.1 MHz.

```
// We will output 14.1 MHz on CLK0 and CLK1.
// A PLLA frequency of 705 MHz was chosen to give an even
// divisor by 14.1 MHz.
unsigned long long freq = 1410000000ULL;
unsigned long long pll_freq = 70500000000ULL;

// Set CLK0 and CLK1 to output 14.1 MHz with a fixed PLL frequency
si5351.set_freq_manual(freq, pll_freq, SI5351_CLK0);
si5351.set_freq_manual(freq, pll_freq, SI5351_CLK1);

// Now we can set CLK1 to have a 90 deg phase shift by entering
// 50 in the CLK1 phase register, since the ratio of the PLL to
// the clock frequency is 50.
si5351.set_phase(SI5351_CLK0, 0);
si5351.set_phase(SI5351_CLK1, 50);

// We need to reset the PLL before they will be in phase alignment
si5351.pll_reset(SI5351_PLLA);
```

CLK Output Options

*Please see the example sketch **si5351_outputs.ino***

In most cases, you will most likely end up using the multisynth associated with a CLK output, but the Si5351 has some other options available as well. The reference clocks (both the crystal oscillator and the CLKIN signal) can be mirrored to any CLK output. Also CLK1 through CLK3 can mirror the MS0 (CLK0) output, and likewise the CLK5 through CLK7 outputs can mirror the MS4 (CLK4) output.

If you choose to use one or more of these output options, you first need to enable the fanout option for that particular signal:

```
// Enable clock fanout for the XO
si5351.set_clock_fanout(SI5351_FANOUT_XO, 1);
```

Once that is done, you can use the *set_clock_source()* method to choose the output option you desire. Since the CLK outputs by default are turned off, you may need to turn your CLK output on as well:

```
// Set CLK1 to output the XO signal
si5351.set_clock_source(SI5351_CLK1, SI5351_CLK_SRC_XTAL);
si5351.output_enable(SI5351_CLK1, 1);
```

Using the VCXO (Si5351B)

*Please see the example sketch **si5351_vcxo.ino***

The Si5351B variant has a VCXO feature which can be used to provide voltage-tunable clock outputs, with a voltage control input on pin 3 of the IC. This functionality is provided on the PLLB oscillator internal to the Si5351, so you must assign any clock outputs that you wish to voltage control to PLLB.

The library has a method named *set_vcxo()* that allows you to set the PLLB frequency and the amount of pull range that you wish to use on that oscillator (from 30 to 240 parts-per-million). Using the VCXO is similar to manually setting an output frequency. First, call the *set_vcxo()* method:

```
#define PLLB_FREQ    87600000000ULL

// Set VCXO osc to 876 MHz (146 MHz x 6), 40 ppm pull
si5351.set_vcxo(PLLB_FREQ, 40);
```

Next, we assign the desired VCXO clock output to PLLB:

```
// Set CLK0 to be locked to VCXO
si5351.set_ms_source(SI5351_CLK0, SI5351_PLLB);
```

Finally, we use the *set_freq_manual()* method to set the clock output center frequency:

```
// Tune to 146 MHz center frequency
si5351.set_freq_manual(14600000000ULL, PLLB_FREQ, SI5351_CLK0);
```

Using an External Reference (Si5351C)

*Please see the example sketch **si5351_ext_ref.ino***

The Si5351C variant has a CLKIN input (pin 6) which allows the use of an alternate external CMOS clock reference from 10 to 100 MHz. Either PLLA and/or PLLB can be locked to this external reference. The library tracks the referenced frequencies and correction factors individually for both the crystal oscillator reference (XO) and external reference (CLKIN).

The XO reference frequency is set during the call to *init()*. If you are going to use the external reference clock, then set its nominal frequency with the *set_ref_freq()* method:

```
// Set the CLKIN reference frequency to 10 MHz
si5351.set_ref_freq(10000000UL, SI5351_PLL_INPUT_CLKIN);
```

A correction factor for the external reference clock may also now be set:

```
// Apply a correction factor to CLKIN
si5351.set_correction(0, SI5351_PLL_INPUT_CLKIN);
```

The *set_pll_input()* method is used to set the desired PLLs to reference to the external reference signal on CLKIN instead of the XO signal:

```
// Set PLLA and PLLB to use the signal on CLKIN instead of the XTAL
si5351.set_pll_input(SI5351_PLLA, SI5351_PLL_INPUT_CLKIN);
si5351.set_pll_input(SI5351_PLLB, SI5351_PLL_INPUT_CLKIN);
```

Once that is set, the library can be used as you normally would, with all of the frequency calculations done based on the reference frequency set in *set_ref_freq()*.

Alternate I2C Addresses

The standard I2C bus address for the Si5351 is 0x60, however there are other ICs in the wild that use alternate bus addresses. In order to accommodate these ICs, the class constructor can be called with the I2C bus address as a parameter, as shown in this example:

```
Si5351 si5351(0x61);
```

Startup Conditions

This library initializes the Si5351 parameters to the following values upon startup and on reset:

Multisynths 0 through 5 (and hence the matching clock outputs CLK0 through CLK5) are assigned to PLLA, while Multisynths 6 and 7 are assigned to PLLB.

PLLA and PLLB are set to 800 MHz (also defined as *SI5351_PLL_FIXED* in the library).

All CLK outputs are set to 0 Hz and disabled.

Default drive strength is 2 mA on each output.

Constraints

- Two multisynths cannot share a PLL with when both outputs are >= 100 MHz. The library will refuse to set another multisynth to a frequency in that range if another multisynth sharing the same PLL is already within that frequency range.
- Setting phase will be limited in the extreme edges of the output tuning ranges. Because the phase register is 7-bits in size and is denominated in units representing 1/4 the PLL period, not all phases can be set for all output frequencies. For example, if you need a 90° phase shift, the lowest frequency you can set it at is 4.6875 MHz (600 MHz PLL/128).
- The frequency range of Multisynth 6 and 7 is ~18.45 kHz to 150 MHz. The library assigns PLLB to these two multisynths, so if you choose to use both, then both frequencies must be an even divisor of the PLL frequency (between 6 and 254), so plan accordingly. You can see the current PLLB frequency by accessing the *pllb_freq* public member.
- VCXO pull range can be ±30 to ±240 ppm

Public Methods

init()

```
/*
 * init(uint8_t xtal_load_c, uint32_t ref_osc_freq, int32_t corr)
 *
 * Setup communications to the Si5351 and set the crystal
 * load capacitance.
 *
 * xtal_load_c - Crystal load capacitance. Use the SI5351_CRYSTAL_LOAD_*PF
 * defines in the header file
 * xo_freq - Crystal/reference oscillator frequency in 1 Hz increments.
 * Defaults to 25000000 if a 0 is used here.
 * corr - Frequency correction constant in parts-per-billion
 *
 * Returns a boolean that indicates whether a device was found on the desired
 * I2C address.
 *
 */
bool Si5351::init(uint8_t xtal_load_c, uint32_t ref_osc_freq, uint32_t ref_osc_fre
```

reset()

```
/*
 * reset(void)
 *
 * Call to reset the Si5351 to the state initialized by the library.
 *
 */
void Si5351::reset(void)
```

set_freq()

```
/*
 * set_freq(uint64_t freq, enum si5351_clock clk)
 *
 * Sets the clock frequency of the specified CLK output
 *
 * freq - Output frequency in Hz
 * clk - Clock output
 *   (use the si5351_clock enum)
 */
uint8_t Si5351::set_freq(uint64_t freq, enum si5351_clock clk)
```

set_freq_manual()

```
/*
 * set_freq_manual(uint64_t freq, uint64_t pll_freq, enum si5351_clock clk)
 *
 * Sets the clock frequency of the specified CLK output using the given PLL
 * frequency. You must ensure that the MS is assigned to the correct PLL and
 * that the PLL is set to the correct frequency before using this method.
 *
 * It is important to note that if you use this method, you will have to
 * track that all settings are sane yourself.
 *
 * freq - Output frequency in Hz
 * pll_freq - Frequency of the PLL driving the Multisynth in Hz * 100
 * clk - Clock output
 *   (use the si5351_clock enum)
 */
```

set_pll()

```
/*
 * set_pll(uint64_t pll_freq, enum si5351_pll target_pll)
 *
 * Set the specified PLL to a specific oscillation frequency
 *
 * pll_freq - Desired PLL frequency in Hz * 100
 * target_pll - Which PLL to set
 *     (use the si5351_pll enum)
 */
void Si5351::set_pll(uint64_t pll_freq, enum si5351_pll target_pll)
```

set_ms()

```
/*
 * set_ms(enum si5351_clock clk, struct Si5351RegSet ms_reg, uint8_t int_mode, uin
 *
 * Set the specified multisynth parameters. Not normally needed, but public for ad
 *
 * clk - Clock output
 *   (use the si5351_clock enum)
 * int_mode - Set integer mode
 *  Set to 1 to enable, 0 to disable
 * r_div - Desired r_div ratio
 * div_by_4 - Set Divide By 4 mode
 *   Set to 1 to enable, 0 to disable
 */
void Si5351::set_ms(enum si5351_clock clk, struct Si5351RegSet ms_reg, uint8_t int
```

output_enable()

```
/*
 * output_enable(enum si5351_clock clk, uint8_t enable)
 *
 * Enable or disable a chosen output
 * clk - Clock output
 *   (use the si5351_clock enum)
 * enable - Set to 1 to enable, 0 to disable
 */
void Si5351::output_enable(enum si5351_clock clk, uint8_t enable)
```

drive_strength()

```
/*
 * drive_strength(enum si5351_clock clk, enum si5351_drive drive)
 *
 * Sets the drive strength of the specified clock output
 *
 * clk - Clock output
 *   (use the si5351_clock enum)
 * drive - Desired drive level
 *   (use the si5351_drive enum)
 */
void Si5351::drive_strength(enum si5351_clock clk, enum si5351_drive drive)
```

update_status()

```
/*
 * update_status(void)
 *
 * Call this to update the status structs, then access them
 * via the dev_status and dev_int_status global variables.
 *
 * See the header file for the struct definitions. These
 * correspond to the flag names for registers 0 and 1 in
 * the Si5351 datasheet.
 */
void Si5351::update_status(void)
```

set_correction()

```
/*
 * set_correction(int32_t corr, enum si5351_pll_input ref_osc)
 *
 * corr - Correction factor in ppb
 * ref_osc - Desired reference oscillator
 *     (use the si5351_pll_input enum)
 *
 * Use this to set the oscillator correction factor.
 * This value is a signed 32-bit integer of the
 * parts-per-billion value that the actual oscillation
 * frequency deviates from the specified frequency.
 *
 * The frequency calibration is done as a one-time procedure.
 * Any desired test frequency within the normal range of the
 * Si5351 should be set, then the actual output frequency
 * should be measured as accurately as possible. The
 * difference between the measured and specified frequencies
 * should be calculated in Hertz, then multiplied by 10 in
 * order to get the parts-per-billion value.
 *
 * Since the Si5351 itself has an intrinsic 0 PPM error, this
 * correction factor is good across the entire tuning range of
 * the Si5351. Once this calibration is done accurately, it
 * should not have to be done again for the same Si5351 and
 * crystal.
 */
void Si5351::set_correction(int32_t corr, enum si5351_pll_input ref_osc)
```

set_phase()

```
/*
 * set_phase(enum si5351_clock clk, uint8_t phase)
 *
 * clk - Clock output
 *   (use the si5351_clock enum)
 * phase - 7-bit phase word
 *   (in units of VCO/4 period)
 *
 * Write the 7-bit phase register. This must be used
 * with a user-set PLL frequency so that the user can
 * calculate the proper tuning word based on the PLL period.
 */
void Si5351::set_phase(enum si5351_clock clk, uint8_t phase)
```

get_correction()

```
/*
 * get_correction(enum si5351_pll_input ref_osc)
 *
 * ref_osc - Desired reference oscillator
 *     0: crystal oscillator (XO)
 *     1: external clock input (CLKIN)
 *
 * Returns the oscillator correction factor stored
 * in RAM.
 */
int32_t Si5351::get_correction(enum si5351_pll_input ref_osc)
```

pll_reset()

```
/*
 * pll_reset(enum si5351_pll target_pll)
 *
 * target_pll - Which PLL to reset
 *     (use the si5351_pll enum)
 *
 * Apply a reset to the indicated PLL.
 */
void Si5351::pll_reset(enum si5351_pll target_pll)
```

set_ms_source()

```
/*
 * set_ms_source(enum si5351_clock clk, enum si5351_pll pll)
 *
 * clk - Clock output
 *   (use the si5351_clock enum)
 * pll - Which PLL to use as the source
 *     (use the si5351_pll enum)
 *
 * Set the desired PLL source for a multisynth.
 */
void Si5351::set_ms_source(enum si5351_clock clk, enum si5351_pll pll)
```

set_int()

```
/*
 * set_int(enum si5351_clock clk, uint8_t int_mode)
 *
 * clk - Clock output
 *   (use the si5351_clock enum)
 * enable - Set to 1 to enable, 0 to disable
 *
 * Set the indicated multisynth into integer mode.
 */
void Si5351::set_int(enum si5351_clock clk, uint8_t enable)
```

set_clock_pwr()

```
/*
 * set_clock_pwr(enum si5351_clock clk, uint8_t pwr)
 *
 * clk - Clock output
 *   (use the si5351_clock enum)
 * pwr - Set to 1 to enable, 0 to disable
 *
 * Enable or disable power to a clock output (a power
 * saving feature).
 */
void Si5351::set_clock_pwr(enum si5351_clock clk, uint8_t pwr)
```

set_clock_invert()

```
/*
 * set_clock_invert(enum si5351_clock clk, uint8_t inv)
 *
 * clk - Clock output
 *   (use the si5351_clock enum)
 * inv - Set to 1 to enable, 0 to disable
 *
 * Enable to invert the clock output waveform.
 */
void Si5351::set_clock_invert(enum si5351_clock clk, uint8_t inv)
```

set_clock_source()

```
/*
 * set_clock_source(enum si5351_clock clk, enum si5351_clock_source src)
 *
 * clk - Clock output
 *   (use the si5351_clock enum)
 * src - Which clock source to use for the multisynth
 *   (use the si5351_clock_source enum)
 *
 * Set the clock source for a multisynth (based on the options
 * presented for Registers 16-23 in the Silicon Labs AN619 document).
 * Choices are XTAL, CLKIN, MS0, or the multisynth associated with
 * the clock output.
 */
void Si5351::set_clock_source(enum si5351_clock clk, enum si5351_clock_source src)
```

set_clock_disable()

```
/*
 * set_clock_disable(enum si5351_clock clk, enum si5351_clock_disable dis_state)
 *
 * clk - Clock output
 *   (use the si5351_clock enum)
 * dis_state - Desired state of the output upon disable
 *   (use the si5351_clock_disable enum)
 *
 * Set the state of the clock output when it is disabled. Per page 27
 * of AN619 (Registers 24 and 25), there are four possible values: low,
 * high, high impedance, and never disabled.
 */
void Si5351::set_clock_disable(enum si5351_clock clk, enum si5351_clock_disable di
```

set_clock_fanout()

```
/*
 * set_clock_fanout(enum si5351_clock_fanout fanout, uint8_t enable)
 *
 * fanout - Desired clock fanout
 *   (use the si5351_clock_fanout enum)
 * enable - Set to 1 to enable, 0 to disable
 *
 * Use this function to enable or disable the clock fanout options
 * for individual clock outputs. If you intend to output the XO or
 * CLKIN on the clock outputs, enable this first.
 *
 * By default, only the Multisynth fanout is enabled at startup.
 */
void Si5351::set_clock_fanout(enum si5351_clock_fanout fanout, uint8_t enable)
```

set_pll_input()

```
/*
 * set_pll_input(enum si5351_pll pll, enum si5351_pll_input input)
 *
 * pll - Which PLL to use as the source
 *     (use the si5351_pll enum)
 * input - Which reference oscillator to use as PLL input
 *     (use the si5351_pll_input enum)
 *
 * Set the desired reference oscillator source for the given PLL.
 */
void Si5351::set_pll_input(enum si5351_pll pll, enum si5351_pll_input input)
```

set_vcxo()

```
/*
 * set_vcxo(uint64_t pll_freq, uint8_t ppm)
 *
 * pll_freq - Desired PLL base frequency in Hz * 100
 * ppm - VCXO pull limit in ppm
 *
 * Set the parameters for the VCXO on the Si5351B.
 */
void Si5351::set_vcxo(uint64_t pll_freq, uint8_t ppm)
```

set_ref_freq()

```
/*
 * set_ref_freq(uint32_t ref_freq, enum si5351_pll_input ref_osc)
 *
 * ref_freq - Reference oscillator frequency in Hz
 * ref_osc - Which reference oscillator frequency to set
 *    (use the si5351_pll_input enum)
 *
 * Set the reference frequency value for the desired reference oscillator
 */
void Si5351::set_ref_freq(uint32_t ref_freq, enum si5351_pll_input ref_osc)
```

si5351_write_bulk()

```
uint8_t Si5351::si5351_write_bulk(uint8_t addr, uint8_t bytes, uint8_t *data)
```

si5351_write()

```
uint8_t Si5351::si5351_write(uint8_t addr, uint8_t data)
```

si5351_read()

```
uint8_t Si5351::si5351_read(uint8_t addr)
```

Public Variables

```
struct Si5351Status dev_status;
struct Si5351IntStatus dev_int_status;
enum si5351_pll pll_assignment[8];
uint64_t clk_freq[8];
uint64_t plla_freq;
uint64_t pllb_freq;
uint32_t xtal_freq;
```

Tokens

Here are the defines, structs, and enumerations you will find handy to use with the library.

Crystal load capacitance:

```
SI5351_CRYSTAL_LOAD_0PF
SI5351_CRYSTAL_LOAD_6PF
SI5351_CRYSTAL_LOAD_8PF
SI5351_CRYSTAL_LOAD_10PF
```

Clock outputs:

```
enum si5351_clock {SI5351_CLK0, SI5351_CLK1, SI5351_CLK2, SI5351_CLK3,
  SI5351_CLK4, SI5351_CLK5, SI5351_CLK6, SI5351_CLK7};
```

PLL sources:

```
enum si5351_pll {SI5351_PLLA, SI5351_PLLB};
```

Drive levels:

```
enum si5351_drive {SI5351_DRIVE_2MA, SI5351_DRIVE_4MA, SI5351_DRIVE_6MA, SI5351_DR
```

Clock sources:

```
enum si5351_clock_source {SI5351_CLK_SRC_XTAL, SI5351_CLK_SRC_CLKIN, SI5351_CLK_SR
```

Clock disable states:

```
enum si5351_clock_disable {SI5351_CLK_DISABLE_LOW, SI5351_CLK_DISABLE_HIGH, SI5351
```

Clock fanout:

```
enum si5351_clock_fanout {SI5351_FANOUT_CLKIN, SI5351_FANOUT_XO, SI5351_FANOUT_MS}
```

PLL input sources:

```
enum si5351_pll_input{SI5351_PLL_INPUT_XO, SI5351_PLL_INPUT_CLKIN};
```

Status register:

```
struct Si5351Status
{
  uint8_t SYS_INIT;
  uint8_t LOL_B;
  uint8_t LOL_A;
  uint8_t LOS;
  uint8_t REVID;
};
```

Interrupt register:

```
struct Si5351IntStatus
{
  uint8_t SYS_INIT_STKY;
  uint8_t LOL_B_STKY;
  uint8_t LOL_A_STKY;
  uint8_t LOS_STKY;
};
```

Raw Commands

If you need to read and write raw data to the Si5351, there is public access to the library's *read()*, *write()*, and *write_bulk()* methods.

Unsupported Features

This library does not currently support the spread spectrum function of the Si5351.

Changelog

- v2.1.2
    - Correct error in si5351_calibration.ino sketch
- v2.1.1

    - Add bool return value to *init()* indicating whether a device is on the I2C bus
- v2.1.0
    - Add support for reference frequencies and corrections for both the XO and CLKIN
- v2.0.7
    - Change *set_freq()* behavior so that the output is only automatically enabled the very first time that *set_freq()* is called
- v2.0.6
    - Call *set_pll()* in *set_correction()* to ensure that the new correction factor is applied
- v2.0.5
    - Remove PLL reset from *set_freq()* when not necessary
- v2.0.4
    - Fix error in VCXO algorithm
- v2.0.3
    - Fix regression in *set_freq()* that wiped out proper R div setting, causing errors in setting low frequency outputs
- v2.0.2
    - Increase maximum frequency in *set_freq()* to 225 MHz
    - Change SI5351_MULTISYNTH_SHARE_MAX from 112.5 MHz to 100 MHz due to stability issues
    - Add explicit reset of VCXO param in *reset()*
    - Add I2C bus address parameter and default to class constructor
    - Update si5351_calibration example sketch
- v2.0.1
    - Fix logic error in *set_freq()* which causes errors in setting multiple clocks >100 MHz
- v2.0.0
    - Complete rewrite of tuning algorithm

    - Add support for setting CLK6 and CLK7
    - Add support for VCXO (on Si5351B)
    - Change interface of *init()* and *set_freq()*
    - Add *set_freq_manual()* method
    - Add *reset()* method
    - Added many new example sketches

- v1.1.2
    - Fix error where register 183 is not pre-loaded with correct value per AN619. Add define for SI5351_CRYSTAL_LOAD_0PF (undocumented in AN619 but present in the official ClockBuilder software).

- v1.1.1
    - Fix if statement eval error in *set_clock_disable()*

- v1.1.0
    - Added *set_pll_input()* method to allow toggling the PLL reference source for the Si5351C variant and added support to *init()* for different PLL reference frequencies from 10 to 100 MHz.

- v1.0.0
    - Initial release