
AdafruitSI5351 Library Documentation

Release 1.0

Tony DiCola

Dec 21, 2018

Contents

1	Dependencies	3
2	Usage Example	5
3	Contributing	7
4	Building locally	9
4.1	Zip release files	9
4.2	Sphinx documentation	9
5	Table of Contents	11
5.1	Simple test	11
5.2	adafruit_si5351	12
6	Indices and tables	15
	Python Module Index	17

SI5351 clock generator module.

CHAPTER 1

Dependencies

This driver depends on:

- [Adafruit CircuitPython](#)
- [Bus Device](#)

Please ensure all dependencies are available on the CircuitPython filesystem. This is easily achieved by downloading the [Adafruit library and driver bundle](#).

CHAPTER 2

Usage Example

See `examples/simpletest.py` for a demo of the usage.

CHAPTER 3

Contributing

Contributions are welcome! Please read our [Code of Conduct](#) before contributing to help this project stay welcoming.

4.1 Zip release files

To build this library locally you'll need to install the `circuitpython-build-tools` package.

```
python3 -m venv .env
source .env/bin/activate
pip install circuitpython-build-tools
```

Once installed, make sure you are in the virtual environment:

```
source .env/bin/activate
```

Then run the build:

```
circuitpython-build-bundles --filename_prefix adafruit-circuitpython-si5351 --library_
↪location .
```

4.2 Sphinx documentation

Sphinx is used to build the documentation based on rST files and comments in the code. First, install dependencies (feel free to reuse the virtual environment from above):

```
python3 -m venv .env
source .env/bin/activate
pip install Sphinx sphinx-rtd-theme
```

Now, once you have the virtual environment activated:

```
cd docs
sphinx-build -E -W -b html . _build/html
```

This will output the documentation to `docs/_build/html`. Open the `index.html` in your browser to view them. It will also (due to `-W`) error out on any warning like Travis will. This is a good way to locally verify it will pass.

5.1 Simple test

Ensure your device works with this simple test.

Listing 1: examples/simpletest.py

```
1  # Simple demo of the SI5351 clock generator.
2  # This is like the Arduino library example:
3  #   https://github.com/adafruit/Adafruit_SI5351_Library/blob/master/examples/si5351/
   ↪ si5351.ino
4  # Which will configure the chip with:
5  #   - PLL A at 900mhz
6  #   - PLL B at 616.66667mhz
7  #   - Clock 0 at 112.5mhz, using PLL A as a source divided by 8
8  #   - Clock 1 at 13.553115mhz, using PLL B as a source divided by 45.5
9  #   - Clock 2 at 10.76khz, using PLL B as a source divided by 900 and further
10 #     divided with an R divider of 64.
11 import board
12 import busio
13
14 import adafruit_si5351
15
16
17 # Initialize I2C bus.
18 i2c = busio.I2C(board.SCL, board.SDA)
19
20 # Initialize SI5351.
21 si5351 = adafruit_si5351.SI5351(i2c)
22 # Alternatively you can specify the I2C address if it has been changed:
23 #si5351 = adafruit_si5351.SI5351(i2c, address=0x61)
24
25 # Now configure the PLLs and clock outputs.
26 # The PLLs can be configured with a multiplier and division of the on-board
```

(continues on next page)

(continued from previous page)

```

27 # 25mhz reference crystal. For example configure PLL A to 900mhz by multiplying
28 # by 36. This uses an integer multiplier which is more accurate over time
29 # but allows less of a range of frequencies compared to a fractional
30 # multiplier shown next.
31 si5351.pll_a.configure_integer(36) # Multiply 25mhz by 36
32 print('PLL A frequency: {0}mhz'.format(si5351.pll_a.frequency/1000000))
33
34 # And next configure PLL B to 616.6667mhz by multiplying 25mhz by 24.667 using
35 # the fractional multiplier configuration. Notice you specify the integer
36 # multiplier and then a numerator and denominator as separate values, i.e.
37 # numerator 2 and denominator 3 means 2/3 or 0.667. This fractional
38 # configuration is susceptible to some jitter over time but can set a larger
39 # range of frequencies.
40 si5351.pll_b.configure_fractional(24, 2, 3) # Multiply 25mhz by 24.667 (24 2/3)
41 print('PLL B frequency: {0}mhz'.format(si5351.pll_b.frequency/1000000))
42
43 # Now configure the clock outputs. Each is driven by a PLL frequency as input
44 # and then further divides that down to a specific frequency.
45 # Configure clock 0 output to be driven by PLL A divided by 8, so an output
46 # of 112.5mhz (900mhz / 8). Again this uses the most precise integer division
47 # but can't set as wide a range of values.
48 si5351.clock_0.configure_integer(si5351.pll_a, 8)
49 print('Clock 0: {0}mhz'.format(si5351.clock_0.frequency/1000000))
50
51 # Next configure clock 1 to be driven by PLL B divided by 45.5 to get
52 # 13.5531mhz (616.6667mhz / 45.5). This uses fractional division and again
53 # notice the numerator and denominator are explicitly specified. This is less
54 # precise but allows a large range of frequencies.
55 si5351.clock_1.configure_fractional(si5351.pll_b, 45, 1, 2) # Divide by 45.5 (45 1/2)
56 print('Clock 1: {0}mhz'.format(si5351.clock_1.frequency/1000000))
57
58 # Finally configure clock 2 to be driven by PLL B divided once by 900 to get
59 # down to 685.15 khz and then further divided by a special R divider that
60 # divides 685.15 khz by 64 to get a final output of 10.706khz.
61 si5351.clock_2.configure_integer(si5351.pll_b, 900)
62 # Set the R divider, this can be a value of:
63 # - R_DIV_1: divider of 1
64 # - R_DIV_2: divider of 2
65 # - R_DIV_4: divider of 4
66 # - R_DIV_8: divider of 8
67 # - R_DIV_16: divider of 16
68 # - R_DIV_32: divider of 32
69 # - R_DIV_64: divider of 64
70 # - R_DIV_128: divider of 128
71 si5351.clock_2.r_divider = adafruit_si5351.R_DIV_64
72 print('Clock 2: {0}khz'.format(si5351.clock_2.frequency/1000))
73
74 # After configuring PLLs and clocks, enable the outputs.
75 si5351.outputs_enabled = True
76 # You can disable them by setting false.

```

5.2 adafruit_si5351

CircuitPython module to control the SI5351 clock generator. See examples/simpletest.py for a demo of the usage. This is based on the Arduino library at: https://github.com/adafruit/Adafruit_SI5351_Library/

- Author(s): Tony DiCola

class adafruit_si5351.**SI5351** (*i2c*, *, *address=96*)

SI5351 clock generator. Initialize this class by specifying:

- *i2c*: The I2C bus connected to the chip.

Optionally specify:

- *address*: The I2C address of the device if it differs from the default.

outputs_enabled

Get and set the enabled state of all clock outputs as a boolean. If true then all clock outputs are enabled, and if false then they are all disabled.

CHAPTER 6

Indices and tables

- `genindex`
- `modindex`
- `search`

a

adafruit_si5351, [12](#)

A

`adafruit_si5351` (module), [12](#)

O

`outputs_enabled` (`adafruit_si5351.SI5351` attribute), [13](#)

S

`SI5351` (class in `adafruit_si5351`), [13](#)