Cascade Platform of SiT9514x ClockSoC Products

Clock Generators, Jitter Cleaners, and Network Synchronizers

GUI User Manual

Contents

1 Introduction

The SiT9514x ClockSoC™ products are based on the SiTime's Cascade Platform™ that integrates multiple clock ICs and oscillators into a single device. The SiTime ClockSoC products include clock generators, jitter cleaners, and network synchronizers that support up to four clock inputs and up to the 11 differential or the 22 single-ended clock outputs. The clock outputs can be derived from the 4 PLLs in a manner that provides high flexibility in terms of frequency planning options. These clocks are fully programmable with the I2C/SPI interface for selecting the input frequency to output frequency translations and associated jitter attenuation loop bandwidths. Using advanced design technology, SiT9514x devices provide excellent jitter performance while working reliably under ambient temperatures from -40°C to 85°C. These features make it ideally suited for communications applications (e.g., OTN, SONET/SDH, xDSL, GbE, networking, wireless infrastructure, IEEE 1588 clock steering), broadcast video with genlock, test and instrumentation applications, and high-speed data converters. Additionally, on-chip programmable non-volatile memory enables factory preprogrammed devices that power up with a known configuration.

2 Document applicability

This document applies to the Cascade Platform products shown in Table 1.

This document only describes operation of the Cascade SiTime GUI version 1.30.4 with the SiTime products shown in Table 1. Operation with other versions may differ from that described in in this document.

Refer to the user documentation for your SiTime evaluation board for information about its deployment.

Table 1: Applicable part numbers

Part number	Number of outputs	Device type	Supported by GUI as described in this document	Supported evaluation board
SiT95141	10	Clock Generator	**Yes**	SiT6503EB
SiT95143	10	Clock Generator	**No** — contact SiTime Technical Support to configure	SiT6503EB
SiT95145	10	Jitter Cleaner	**Yes**	SiT6503EB
SiT95147	8	Network Synchronizer	**Yes**	SiT6502EB
SiT95148	11	Network Synchronizer	**Yes**	SiT6503EB

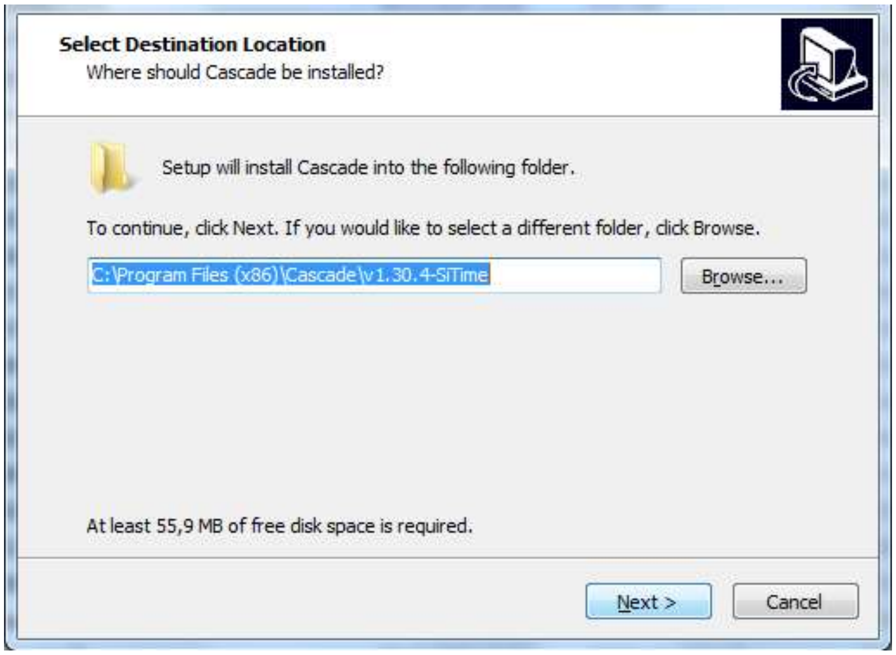
NOTE: About operating parameters

For this document, the application was tested using Microsoft Windows^{TM} 10 Pro, version 1909. Other versions should also work in most cases.

Recommended display resolution is 1600 x 1080 or better. Other resolutions will work; however, if you operate this application using a display with lower resolution (e.g. 1366 x 768), portions of the screen may appear different, such as overlapping sections or wrapping of labels, buttons, fields, etc. Some screenshots in this document were created using lower resolution and may display somewhat different in your deployment.

3 Cascade GUI installation

Open the **setup_Cascade-vn.n.n-SiTime.exe** file and select the folder in which to install the Cascade GUI application, see Figure 1 and Figure 2.


Select Destination Location
Where should Cascade be installed?

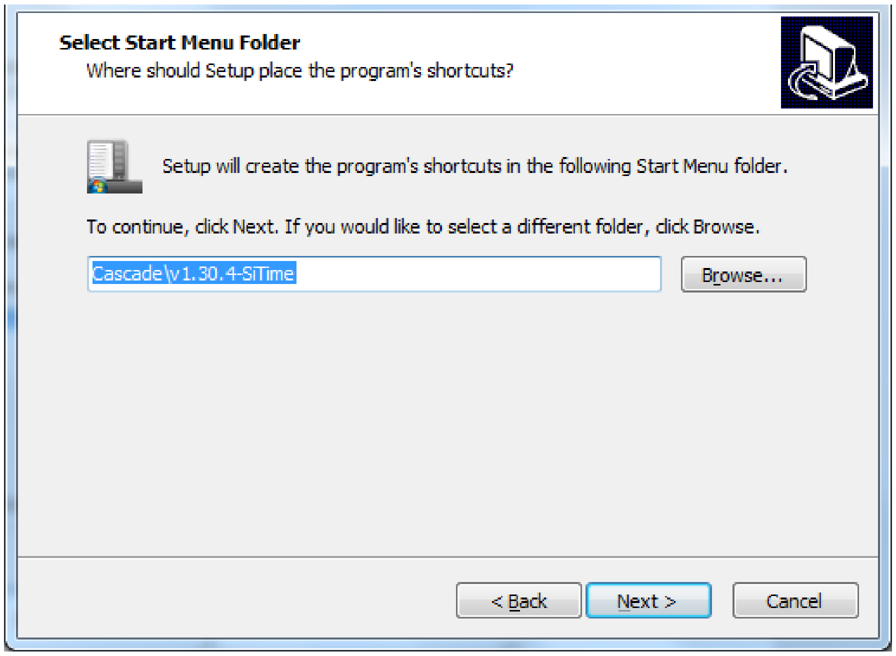
Setup will install Cascade into the following folder.

To continue, click Next. If you would like to select a different folder, click Browse.

C:\Program Files (x86)\Cascade\v1.30.4-SiTime

At least 55,9 MB of free disk space is required.

Figure 1: Selection of the destination location


Select Start Menu Folder
Where should Setup place the program's shortcuts?

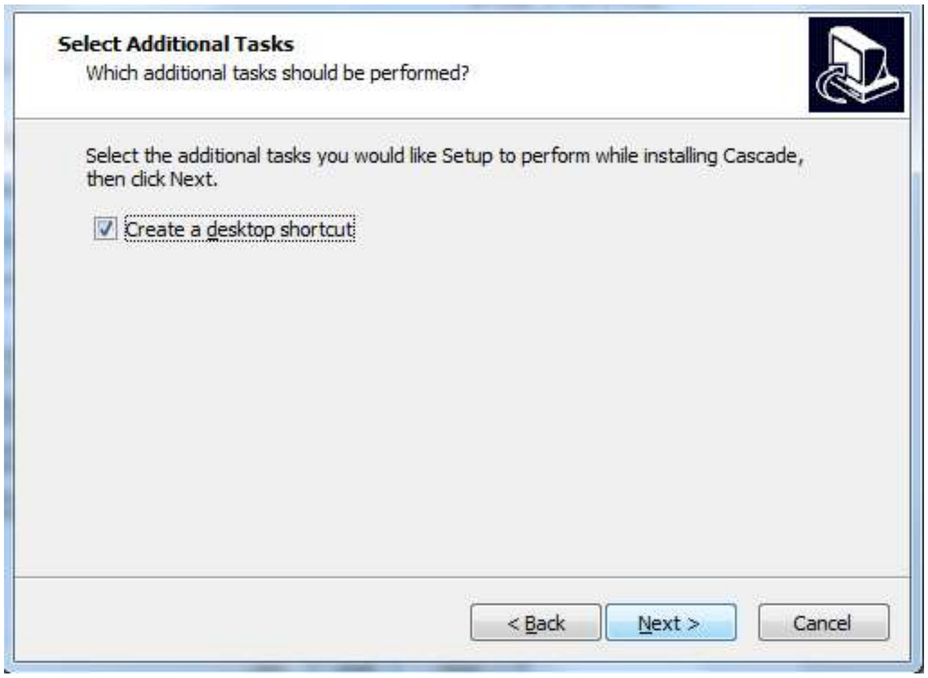
Setup will create the program's shortcuts in the following Start Menu folder.

To continue, click Next. If you would like to select a different folder, click Browse.

Cascade\v1.30.4-SiTime

Figure 2: Selection of the Start Menu folder

Optionally, click the checkbox to create a desktop shortcut. Click **Next** to proceed with the installation, see Figure 3.

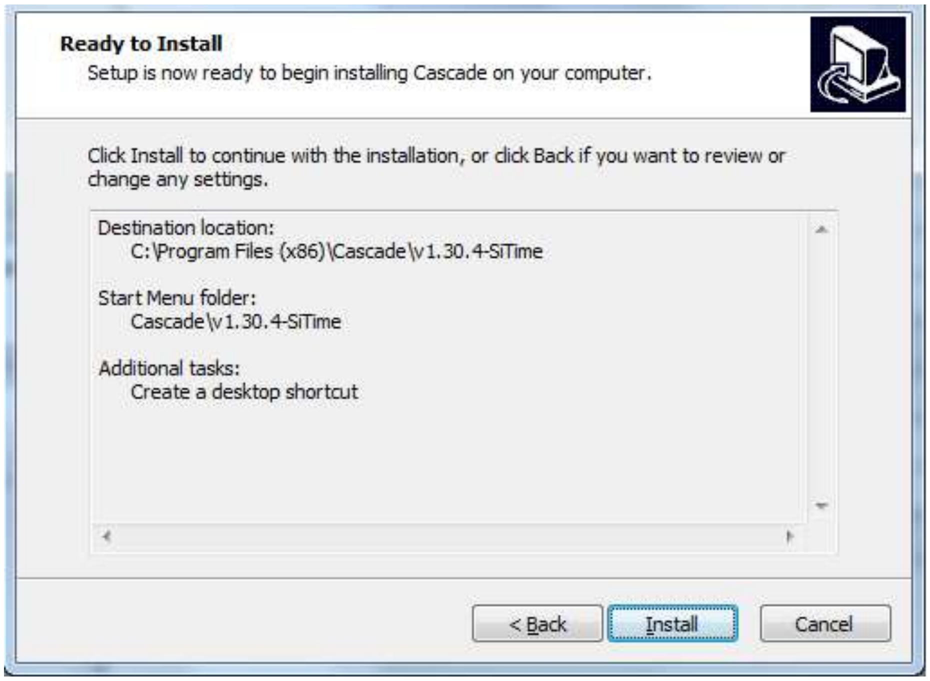

Select Additional Tasks
Which additional tasks should be performed?

Select the additional tasks you would like Setup to perform while installing Cascade, then click Next.

☑ Create a desktop shortcut

Figure 3: Option to create a desktop shortcut and proceed

All SiT9514x related software is installed first, see Figure 4.


Ready to Install
Setup is now ready to begin installing Cascade on your computer.

Click Install to continue with the installation, or click Back if you want to review or change any settings.

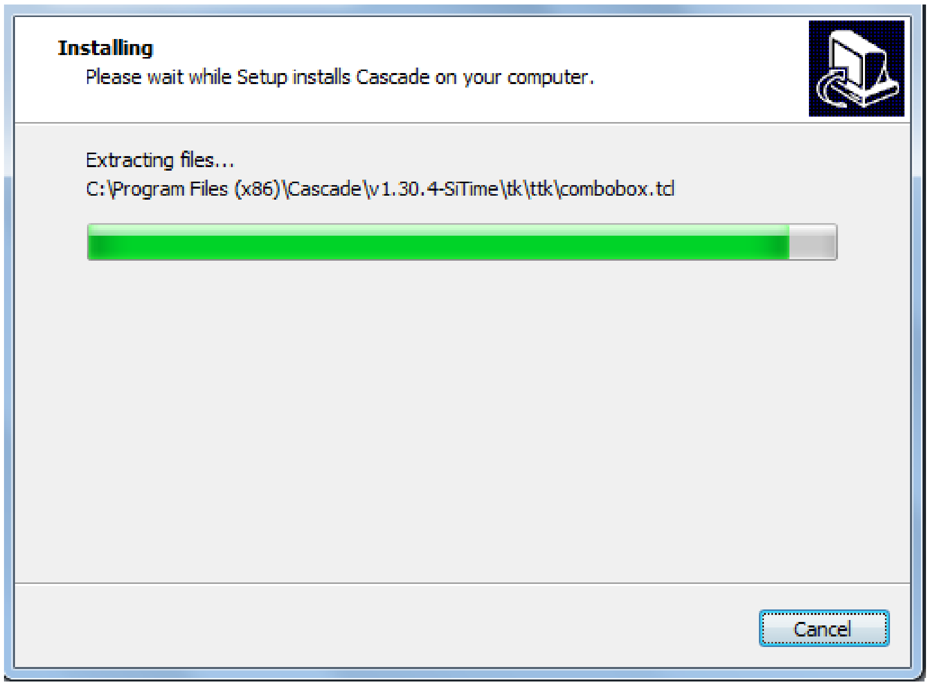
Destination location:
C:\Program Files (x86)\Cascade\v1.30.4-SiTime

Start Menu folder:
Cascade\v1.30.4-SiTime

Additional tasks:
Create a desktop shortcut

Figure 4: Ready to install

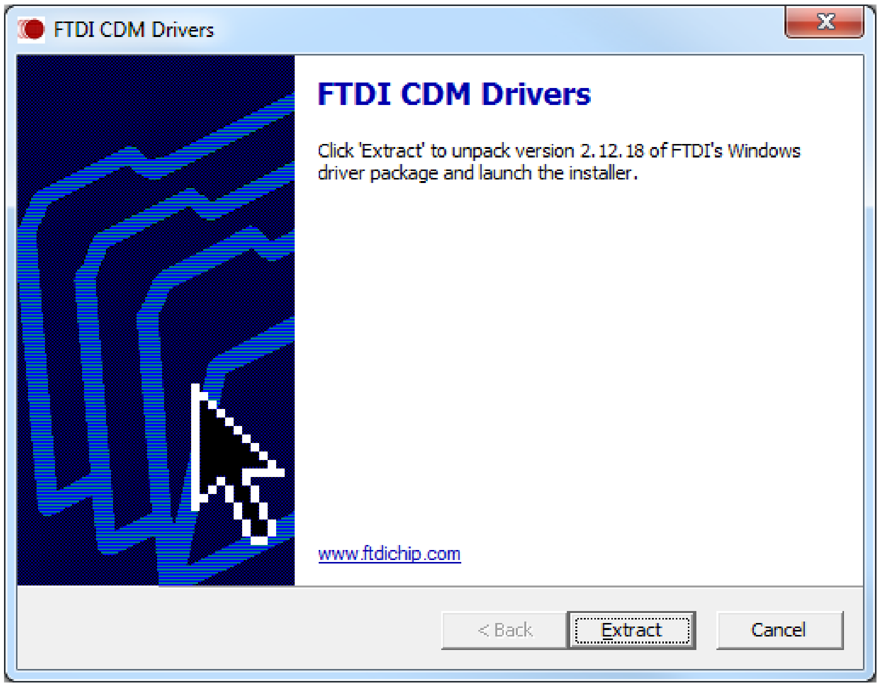
The SiTime evaluation boards use an FTDI chip solution for the USB-to-serial interface conversion. The FTDI driver is installed next, see Figure 5 and Figure 6.



Installing
Please wait while Setup installs Cascade on your computer.

Extracting files...
C:\Program Files (x86)\Cascade\v1.30.4-SiTime\tk\ttk\combobox.tcl

Figure 5: Installation progress



FTDI CDM Drivers

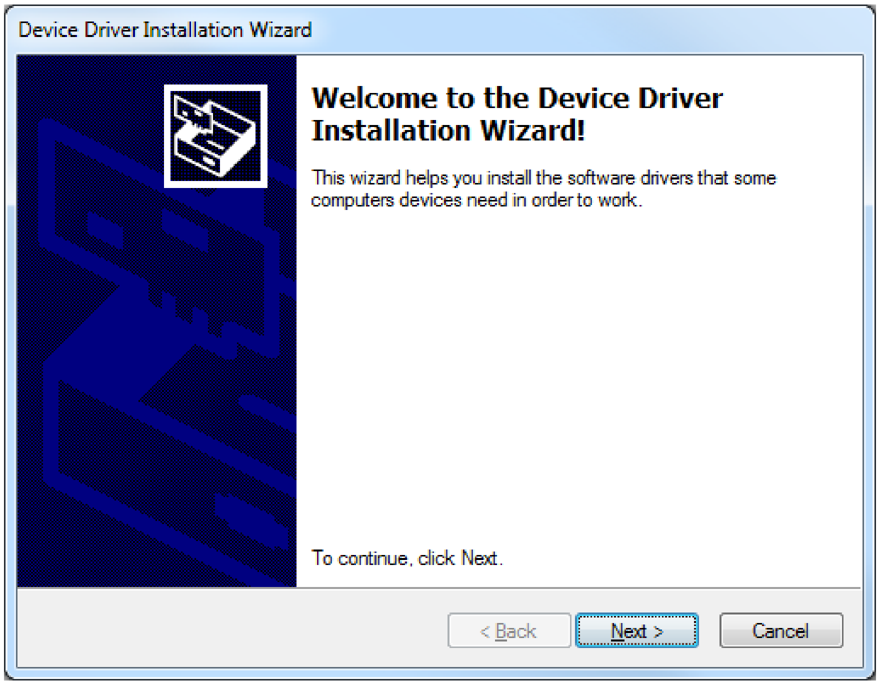
FTDI CDM Drivers

Click 'Extract' to unpack version 2.12.18 of FTDI's Windows driver package and launch the installer.

www.ftdichip.com

Figure 6: Extracting FTDI CDM drivers

Click **Extract** to proceed with the FTDI driver installation, see Figure 7, Figure 8, and Figure 9.

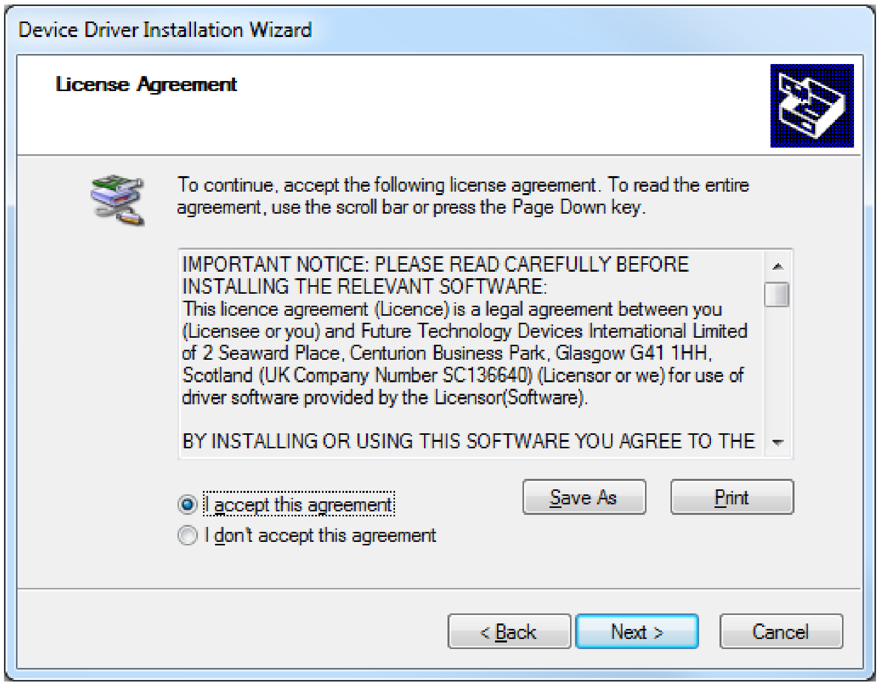

Device Driver Installation Wizard

Welcome to the Device Driver Installation Wizard!

This wizard helps you install the software drivers that some computers devices need in order to work.

To continue, click Next.

Figure 7: Click Next to start the device driver installation wizard


Device Driver Installation Wizard

License Agreement

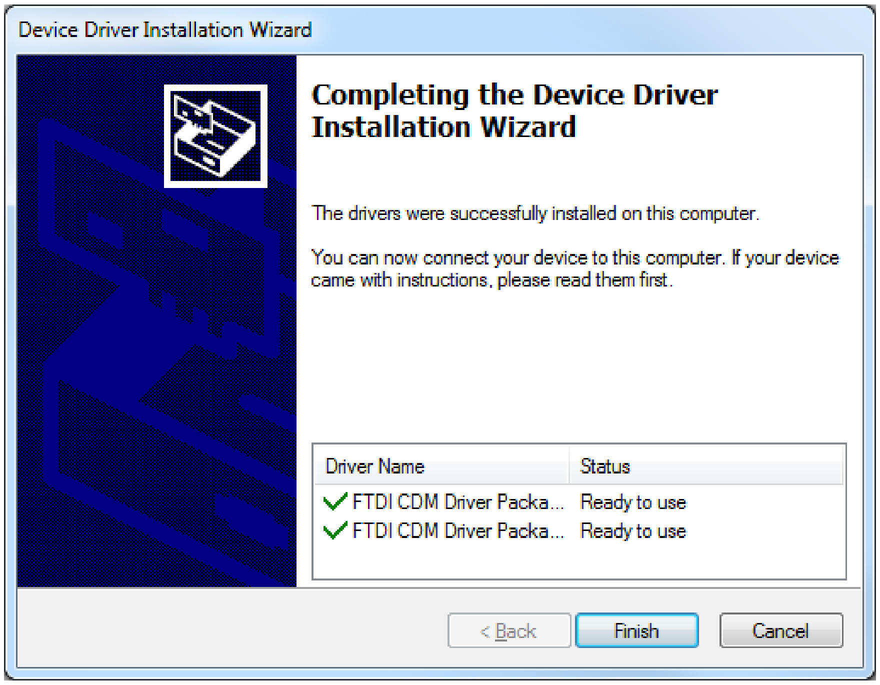
To continue, accept the following license agreement. To read the entire agreement, use the scroll bar or press the Page Down key.

IMPORTANT NOTICE: PLEASE READ CAREFULLY BEFORE INSTALLING THE RELEVANT SOFTWARE:
This licence agreement (Licence) is a legal agreement between you (Licensee or you) and Future Technology Devices International Limited of 2 Seaward Place, Centurion Business Park, Glasgow G41 1HH, Scotland (UK Company Number SC136640) (Licensor or we) for use of driver software provided by the Licensor(Software).

BY INSTALLING OR USING THIS SOFTWARE YOU AGREE TO THE

[✓ I accept this agreement]
I don't accept this agreement

Figure 8: Read and accept the license agreement



Device Driver Installation Wizard

Completing the Device Driver Installation Wizard

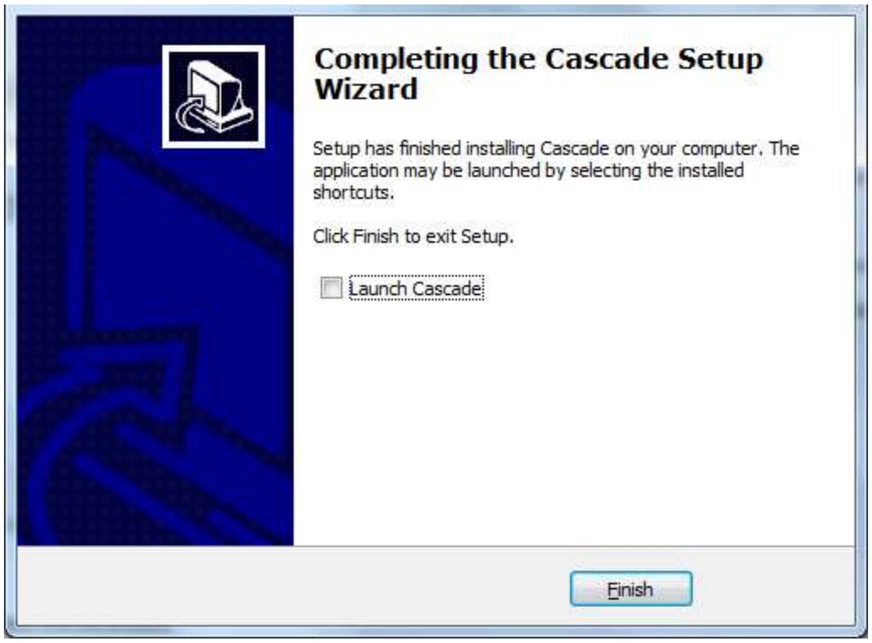
The drivers were successfully installed on this computer.

You can now connect your device to this computer. If your device came with instructions, please read them first.

Driver Name	Status
✓ FTDI CDM Driver Packa...	Ready to use
✓ FTDI CDM Driver Packa...	Ready to use

Figure 9: Finish the driver installation

Click **Finish** to complete setup installation Figure 10.



Completing the Cascade Setup Wizard

Setup has finished installing Cascade on your computer. The application may be launched by selecting the installed shortcuts.

Click Finish to exit Setup.

☐ Launch Cascade

Figure 10: Finish the Cascade setup wizard

4 Starting the GUI

The Cascade GUI provides an easy interface to configure the selected device. Since the GUI uses the FTDI chip for the USB-to-serial I2C communication on the SiTime evaluation board, the FTDI chip related software drivers are also installed by the installation file.

When the Cascade GUI application is launched, the **Choose Variant** selection menu appears to prompt for selection of one of the product variants. Select the **SiT9514x** variant you are using and click **Select**.

It is possible to start and use multiple instances of the application simultaneously.

NOTE: Contact SiTime Technical Support to configure the SiT95143 device.

An example selection of the SiT95141 product variant is shown in Figure 11**:**

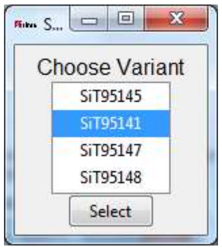


Figure 11: Option for SiT95141

The GUI software will launch for the selected SiT95141 device variant, see Figure 12**.**

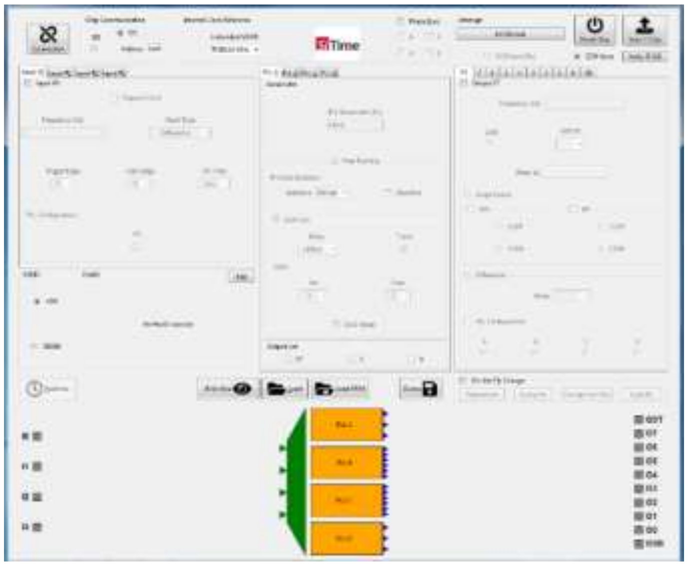


Figure 12: GUI view for SiT9514

Users can move between the individual sections, configuring parameters, while related parameters in the other sections remain visible. Programmed configurations can be saved in configuration profile files containing the configuration parameters, or in sets of I2C/SPI read and write scripts for reuse, etc.

5 Functional descriptions of SiT9514x device variants

The SiT95141 is a clock generator device that offers four fractional-frequency translations from the same input. Any one of the four clock inputs map to all four PLLs. The PLL outputs are mapped to the 10 outputs, offering flexible frequency translation configurations, see Figure 13.

The SiT95145 is a jitter attenuating frequency translation device that offers four fractional translations from the same input. The four clock inputs map to all four PLLs. The PLL outputs can be mapped to a subset of the 10 outputs, offering flexible frequency translation configuration with independent control of each PLL in terms of jitter attenuation, bandwidth control, and input clock selection with redundancy, see Figure 13.

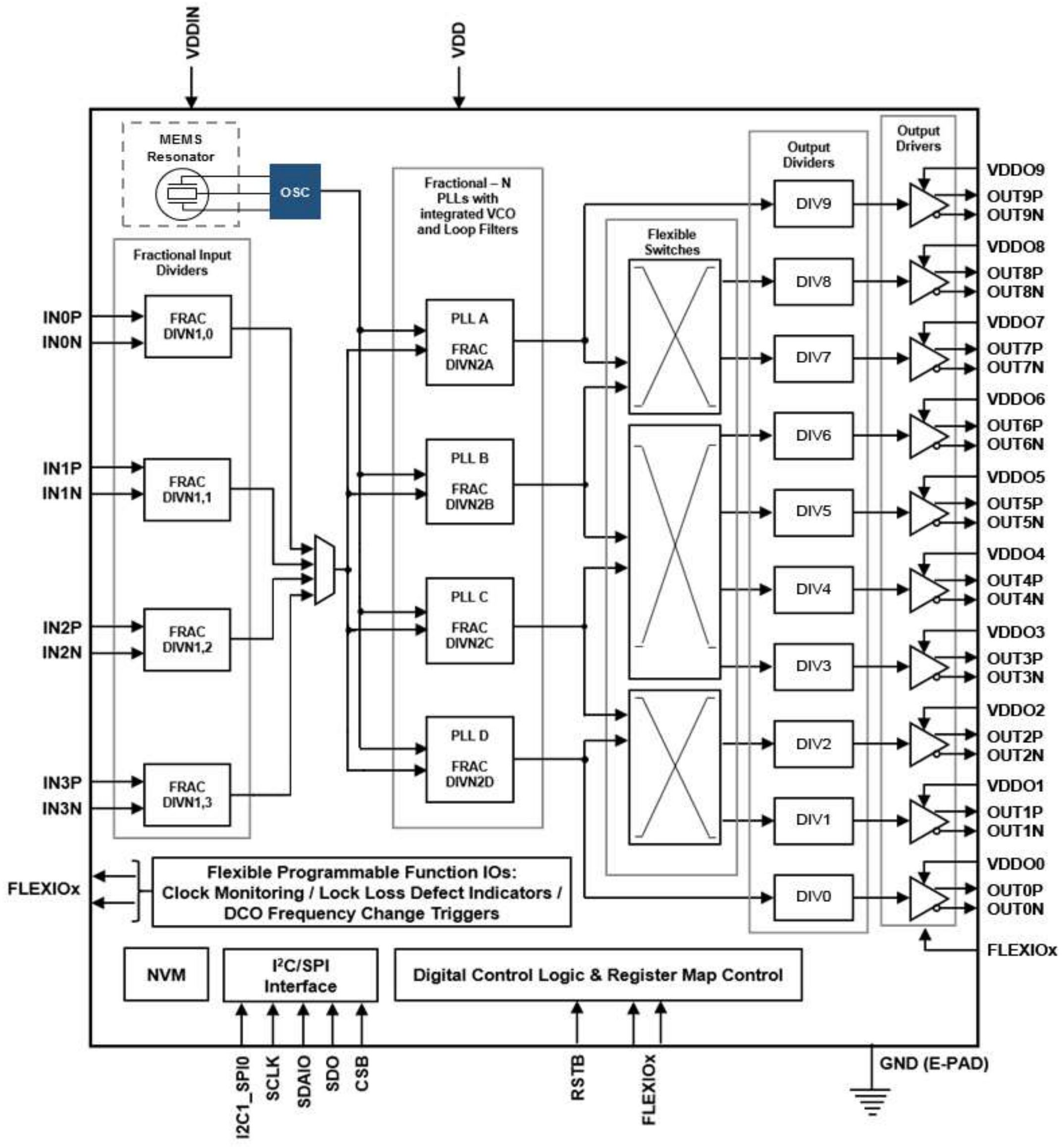


Figure 13: SiT95141/SiT95145 overall architecture

SiT95147 is a jitter attenuating and network synchronizing frequency translation device that offers four independent PLLs. The four clock inputs can map to any of the four PLLs. The PLL outputs are mapped to the eight outputs offering flexible frequency translation configurations with independent control of each PLL in terms of jitter attenuation, bandwidth control, and input clock selection with redundancy, see Figure 14.

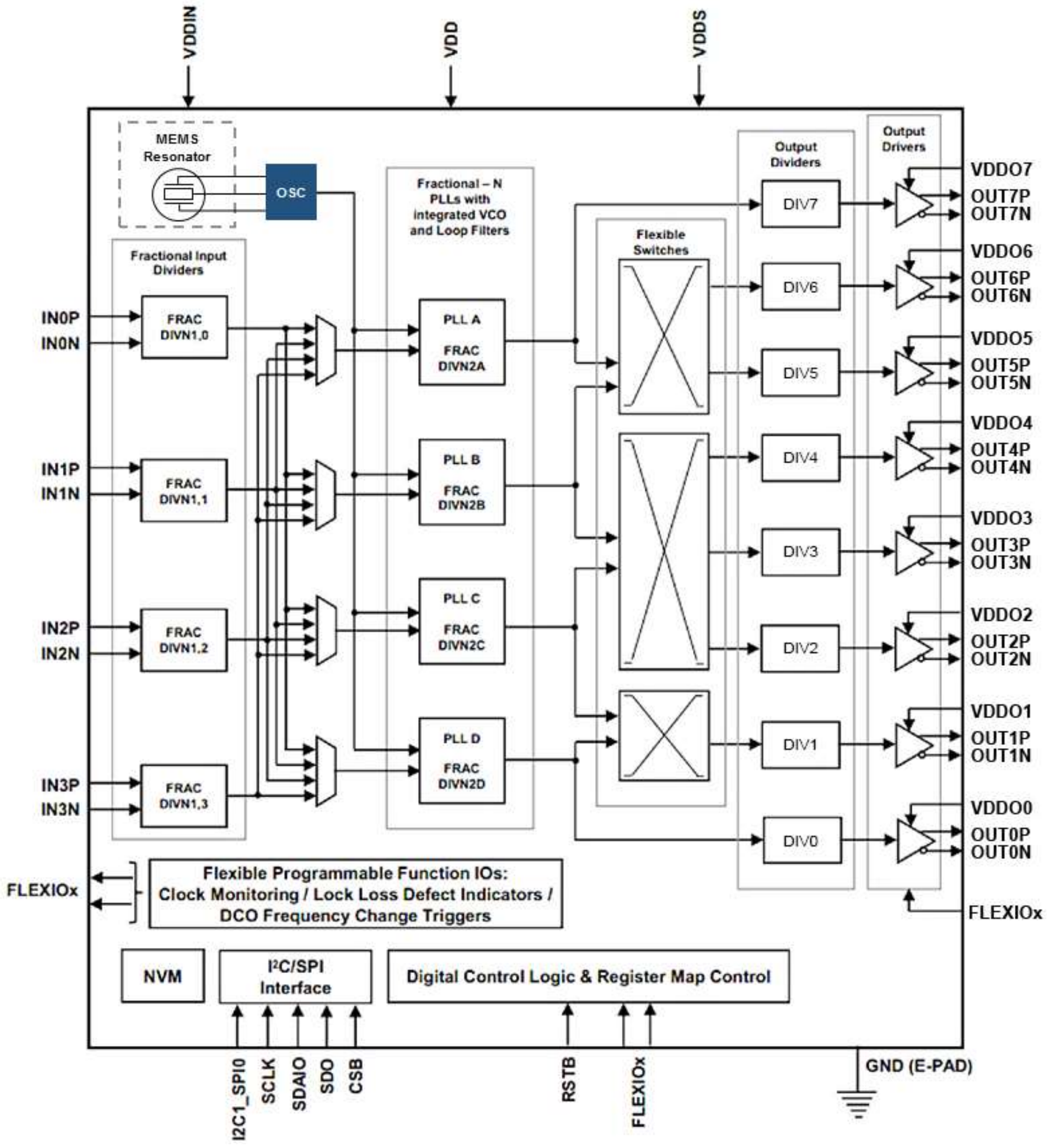


Figure 14: SiT95147 overall architecture

SiT95148 is a jitter attenuating frequency translation device that offers four independent fractional PLLs. PLL outputs are mapped to the 11 outputs. This allows flexible frequency translation configuration with independent control of each PLL in terms of jitter attenuation, bandwidth control, and input clock selection with redundancy, see Figure 15.

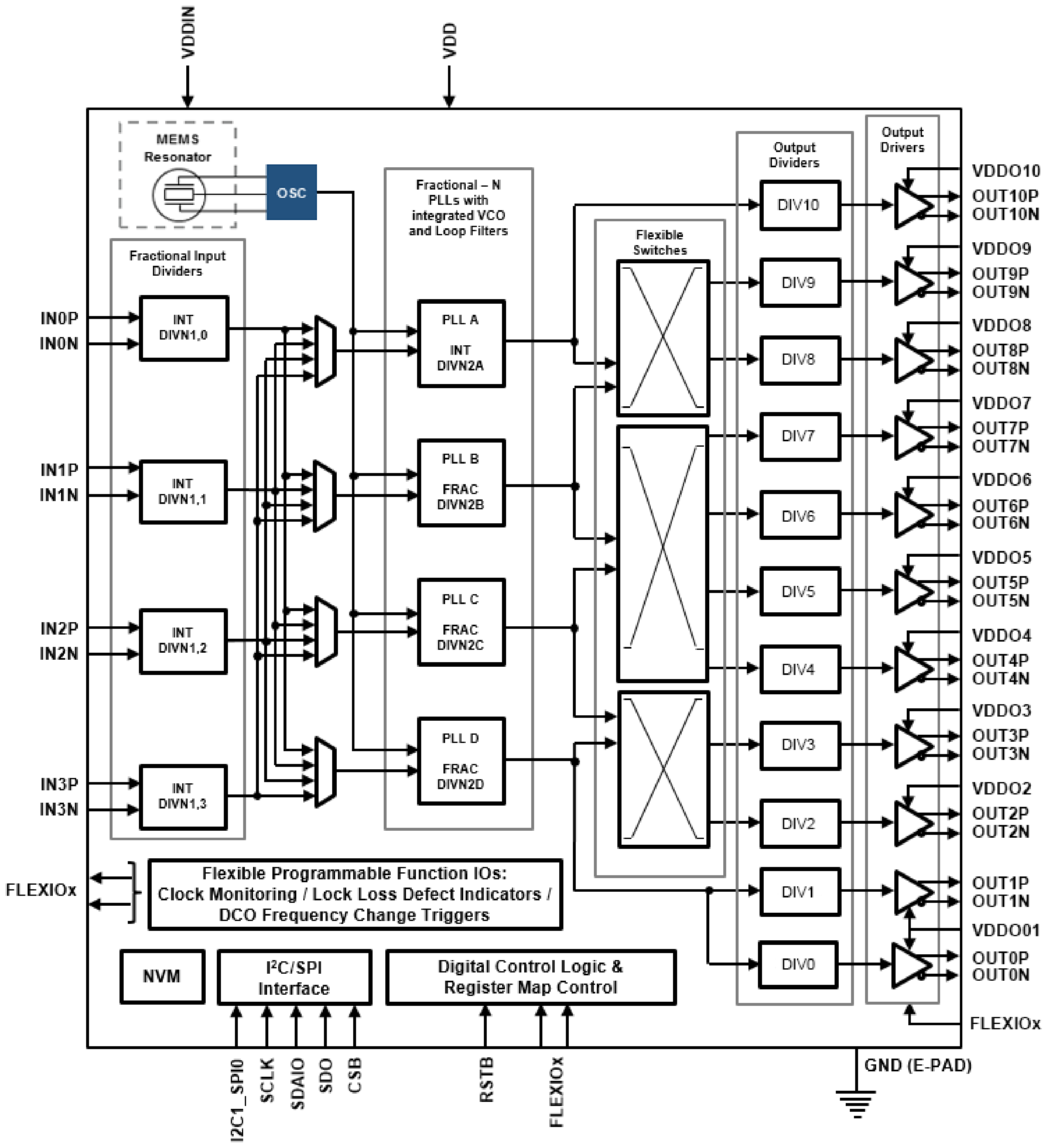


Figure 15: SiT95148 overall architecture

Clock hierarchy, various frequency dividers nomenclature, and clock translation pathways available are shown in Figure 16 for the SiT95141 and SiT95145 devices.

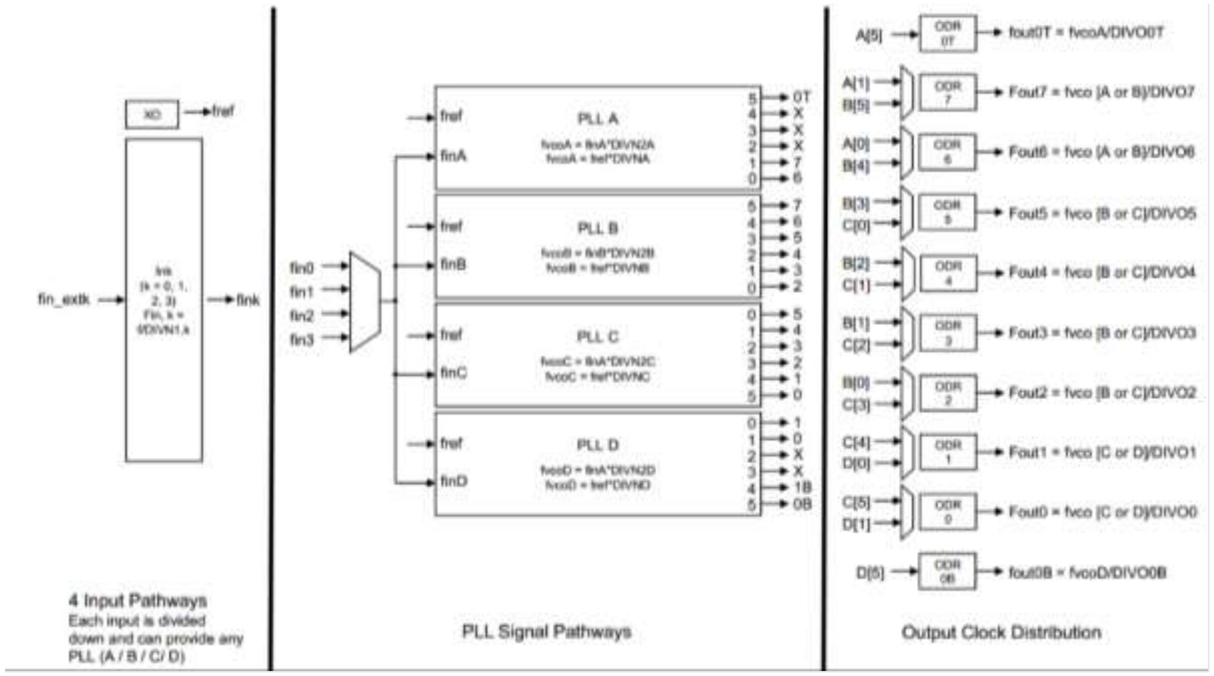


Figure 16: SiT95141 and SiT95145 clock hierarchy

The clock hierarchy, nomenclature of the various frequency dividers, and clock translation pathways available for the SiT95147 are shown in Figure 17.

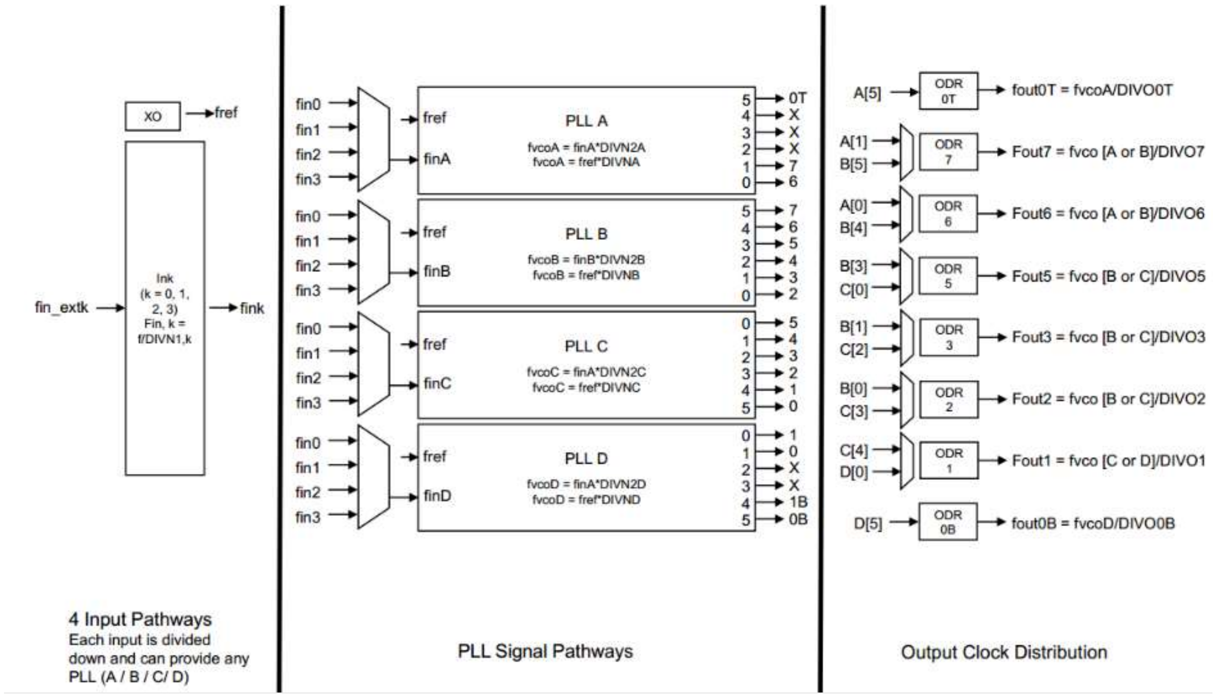


Figure 17: SiT95147 clock hierarchy

The clock hierarchy, various frequency dividers nomenclature, and clock translation pathways available on the SiT95148 device are shown in Figure 18.

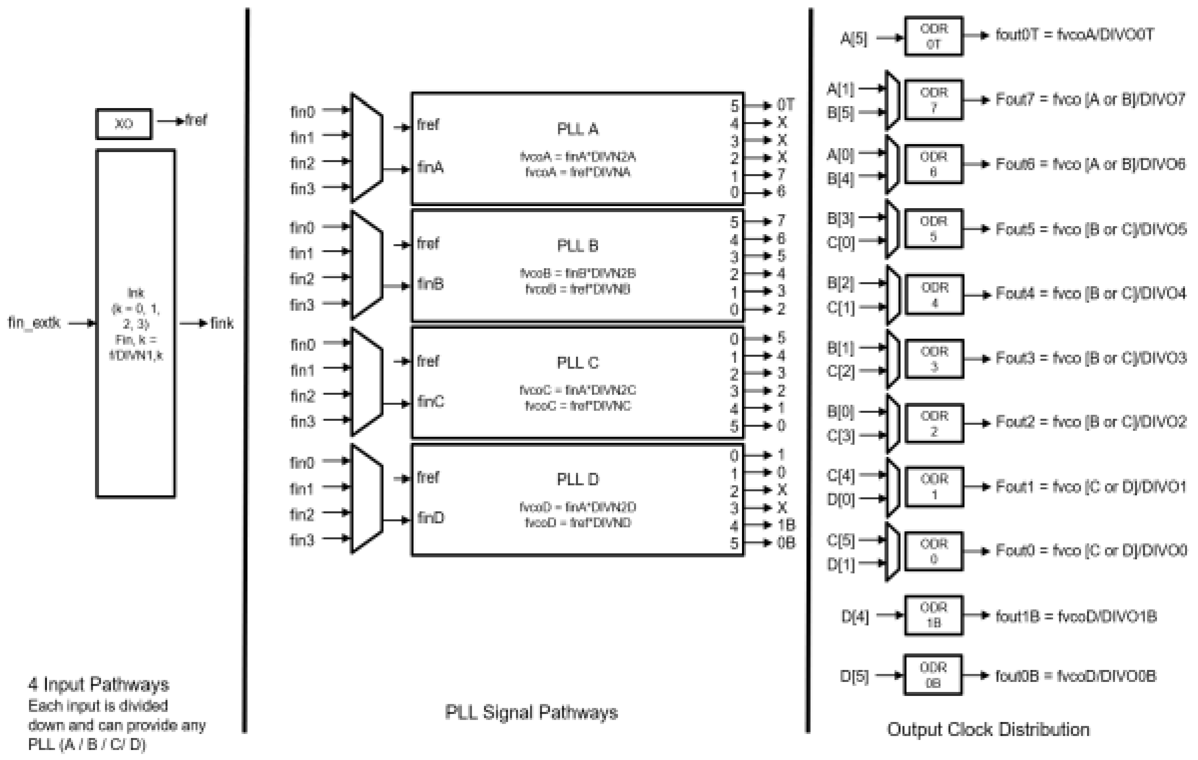


Figure 18: SiT95148 clock hierarchy

6 GUI usage

6.1 Sections in the GUI

6.1.1 GUI Sections overview

The GUI layout is divided into sections, used to configure distinct sets of parameters, and unique to the specific sections, described in detail in subsequent section of this document.

- Chip Communication – See the top-left section.
- Internal Clock Configuration – See the top-center section.
- Input (0,1,2,3) – See the left-middle section.
- PLL (A,B,C,D), Clock Switch and Lock Loss – See the center section.
- Output (0T,7,6,5,4,3,2,1,OB) – See the right-middle section.
- Bird's Eye view – See the bottom section.

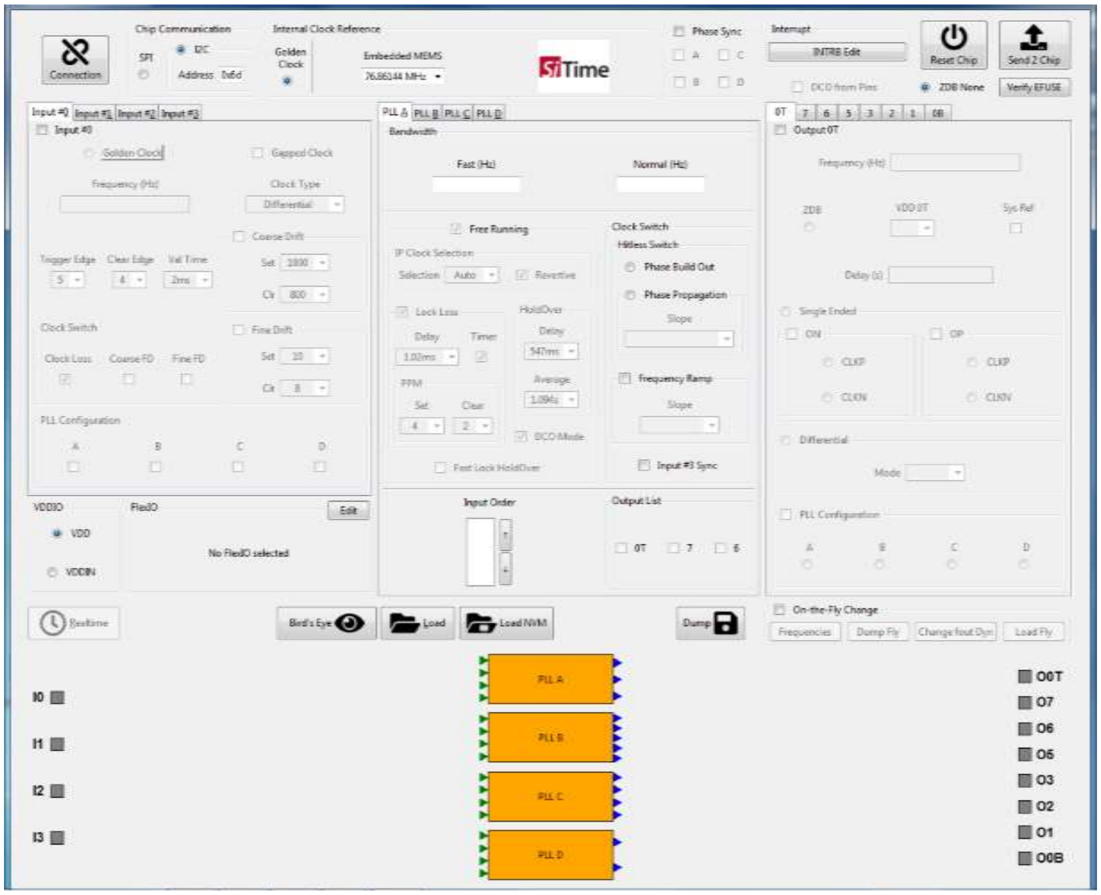


Figure 19: Sections in the GUI interface

6.2 Chip Communication and Interrupt

The **Chip Communication** option allows the user to specify how to connect to the chip.

The **Interrupt** section is used to configure interrupt related settings.

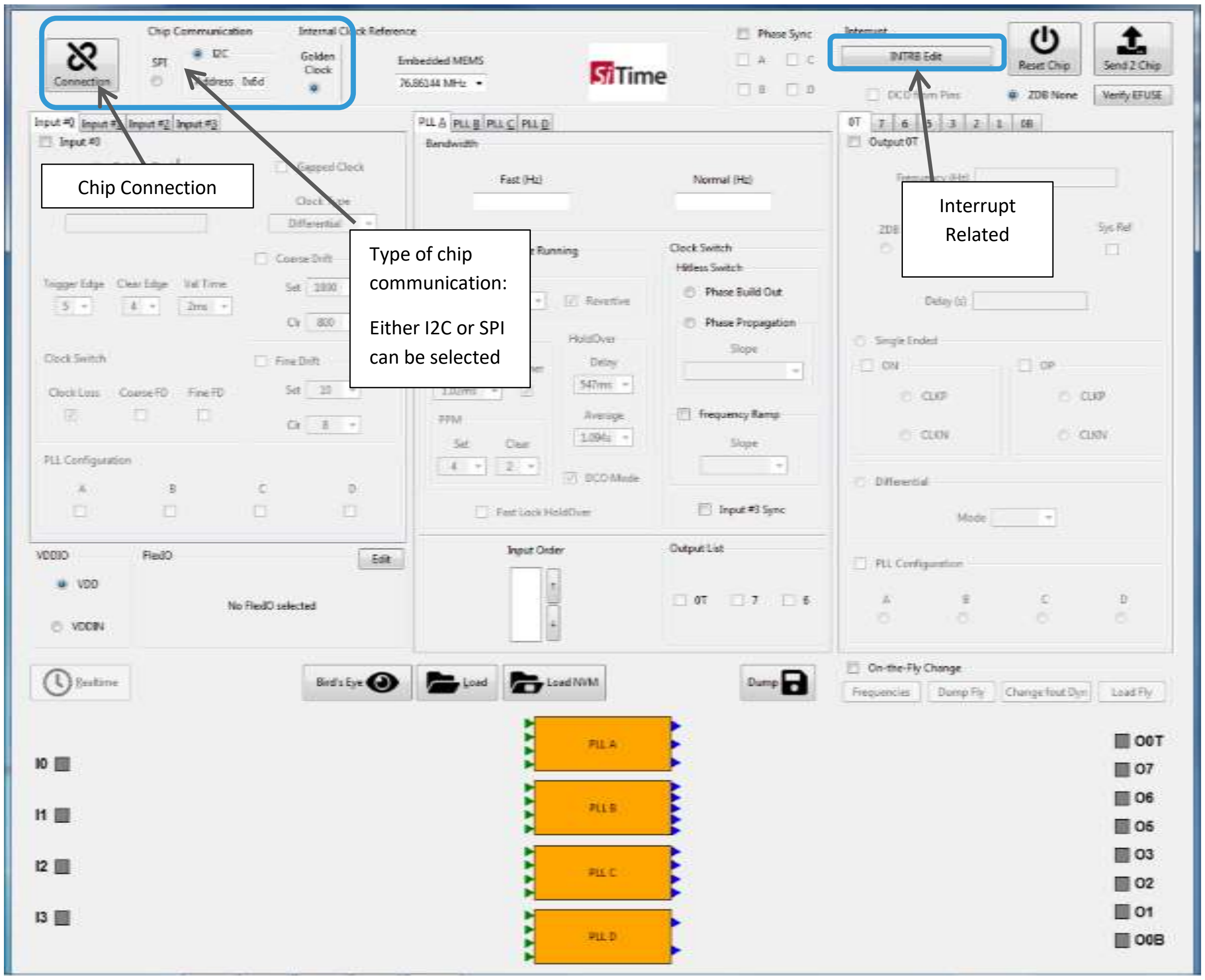
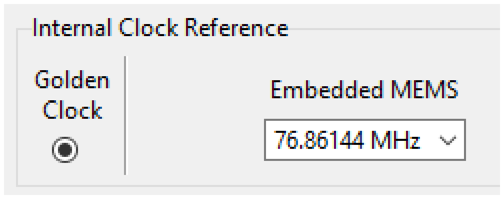


Figure 20: Communication and interrupt options

6.2.1 Input Clock Reference section

The **Input Clock Reference** section is used to configure:

- Golden clock (for frequency drift monitoring)
- Embedded MEMS frequency



Embedded MEMS: 76.86144 MHz

Figure 21: Input Clock Reference section

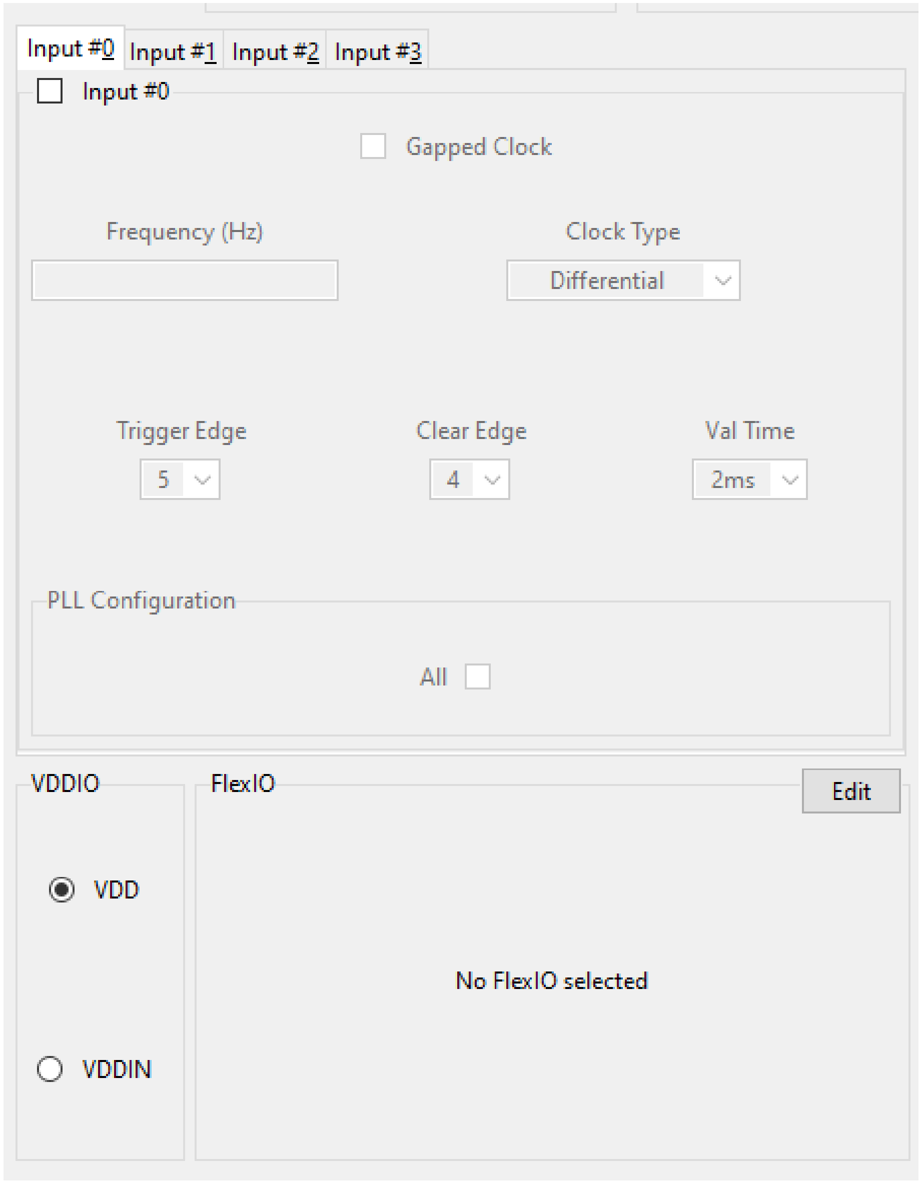
IMPORTANT:

Embedded MEMS *must* be set to 76.86144 MHz only. Operation in the 76.8 MHz setting is *not* generally supported and should *only* be used if advised to do so by SiTime Technical Support.

6.2.2 Input section

The **Input** section is divided into similar individual tabs for **Input #(0-3)** where the following parameters can be set:

- Frequency, clock type, and clock loss/frequency drift
- FlexIO, for setting **Clock Switch** fine and coarse frequency drift monitors for the input clocks
- Input assignment (or not) to all PLLs



☐ Input #0

☐ Gapped Clock

Clock Type: Differential

Trigger Edge: 5

Clear Edge: 4

Val Time: 2ms

All ☐

(● VDD) ○ VDDIN

No FlexIO selected

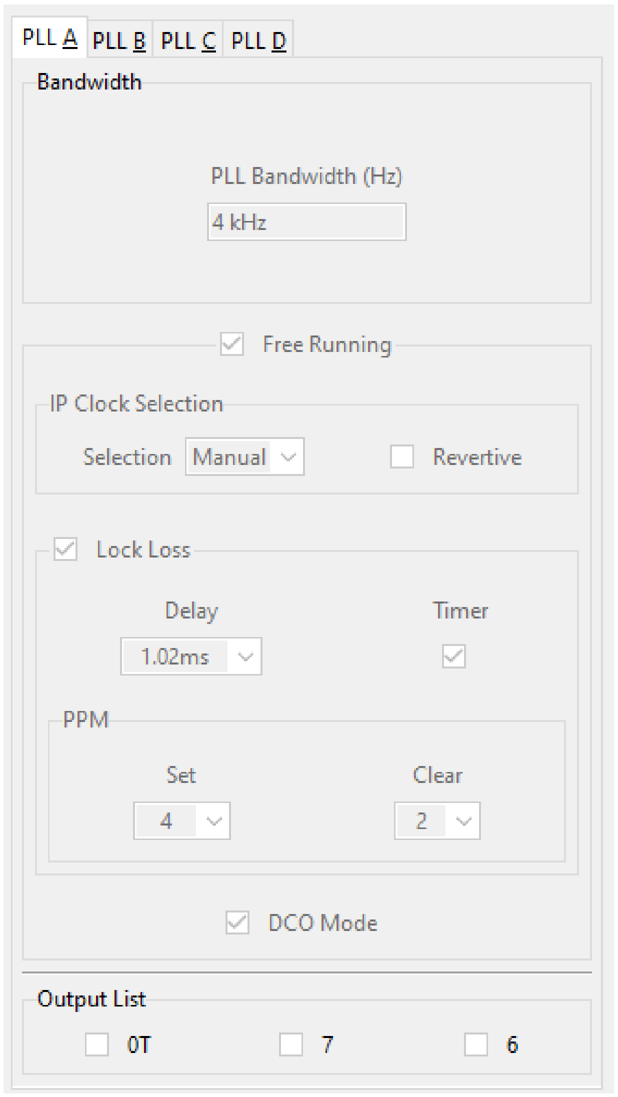
Figure 22: GUI Input section

NOTE: If no input is assigned to the PLL section, then the device's internal PLL oscillator is the primary clock source.

6.2.3 PLL section

The **PLL** section is used for setting all PLL related parameters including:

- PLL (A,B,C,D) – Up to four independently configurable PLL sections. Each PLL supports up to four clock inputs with Frac-N dividers, enabling flexible input to output frequency translation configurations. PLL input clock priority settings can be changed in Page 1h, registers 49h - 4Bh.
- Bandwidth
- Lock Loss
- Clock switching options (SiT95145 only)
- Input clock priorities



PLL Bandwidth (Hz): 4 kHz
✓ Free Running
Selection: Manual
☐ Revertive
✓ Lock Loss
Delay: 1.02ms
Timer: ✓
Set: 4
Clear: 2
✓ DCO Mode
☐ 0T ☐ 7 ☐ 6

Figure 23: GUI PLL section

6.2.4 Output section

The **Output** section is used to set the output standard type, voltage supply, and to assign outputs to the PLLs.

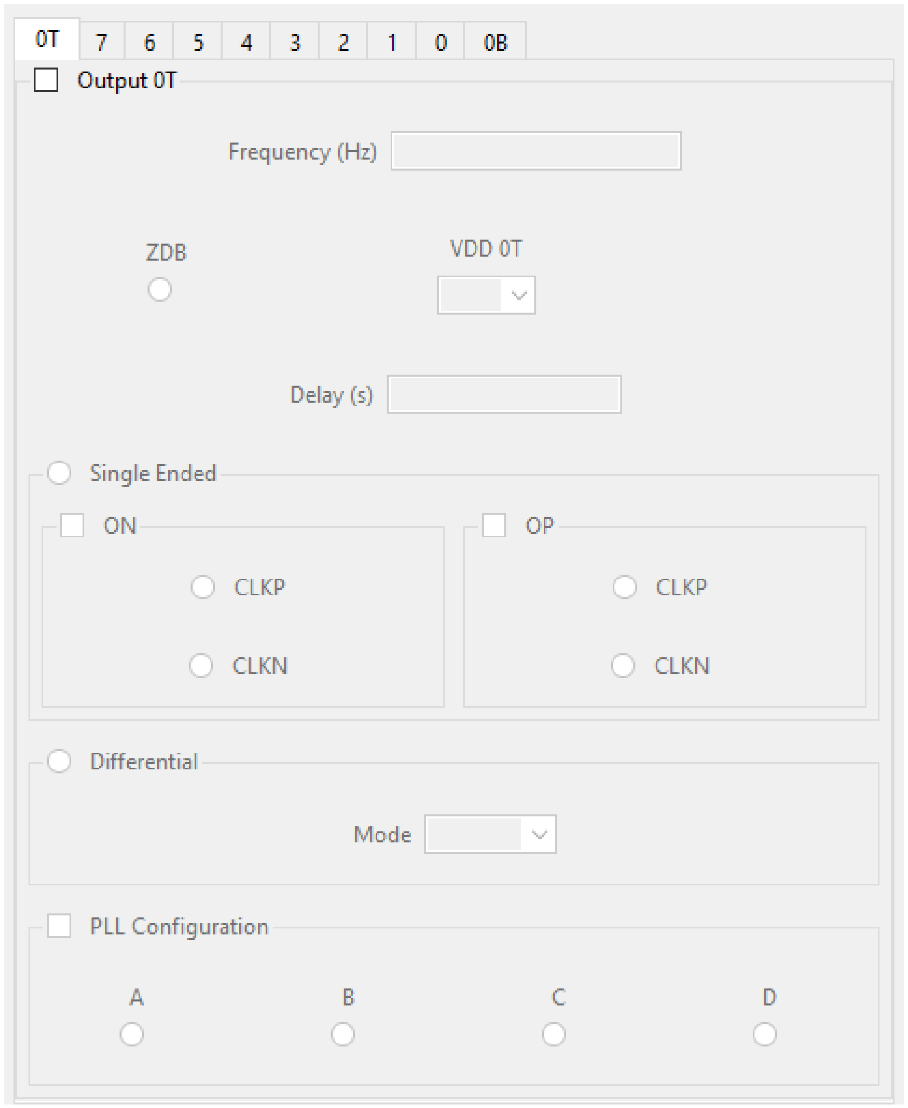


Figure 24: GUI Output section

Additionally, if the DCO mode is enabled in the **Output** section, the **Realtime** section can be used to move the output frequency in the DCO mode. Once the chip is programmed, the **Realtime** section can be used for observing the status of the clock loss monitors.

In the **Free running** mode, the DCO is enabled by default and is available in the **Realtime** section.

For more information, see TASK 4: Use Realtime to set output frequency with DCO or view clock monitor status.

6.2.5 Bird's Eye section

The **Bird's Eye** section can be used to visualize the resulting configuration of Inputs to PLLs to Outputs.

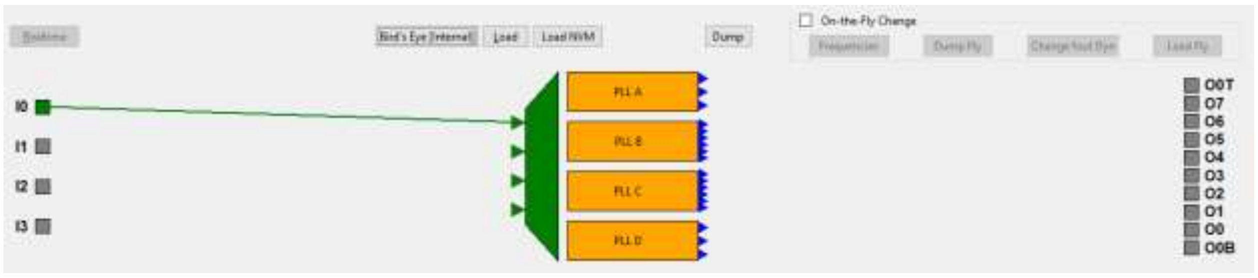


Figure 25: GUI Bird's Eye section

Click the **Bird's Eye** button to activate the Bird's Eye section, showing the configured signal paths. Click the **Bird's Eye** button again to open the Bird's Eye section in a separate window. Close that view to completely close the Bird's Eye view.

This lower section of the Cascade SiTime GUI is also used to:

- Dump (save) and Load (open) device configuration files
- Specify On-the-Fly frequencies
- Change fout parameters
- Dump and Load On-the-Fly configuration parameters
- Select Realtime status of the device

The **Realtime** button opens the Realtime window that displays the detailed operational status of the programmed device based on the current configuration. It can be is used to monitor device operation and view the impact of the set configuration parameters. For an example, see *TASK 4: Use Realtime to set output frequency with DCO or view clock monitor status*.

7 SiT9514x device configuration tasks

An overall approach for using the Cascade SiTime GUI is described in the following tasks:

- TASK 1: Select inputs
- TASK 2: Set up PLL parameters
- TASK 3: Save or load UI configuration and program the SiT9514x
- TASK 4: Use Realtime to set output frequency with DCO or view clock monitor status

7.1 TASK 1: Select inputs

The four input clocks with frequencies **fin_ext*k*** translate to the PLL input clocks **fin*k*,** following division by the respective input dividers with fractional or integer frequency division ratios **DIVN1*k***, where the index ***k* ϵ {0, 1, 2, 3}**. Each of the PLLs are driven by one of the four divided input clocks **fin*k*** as its active input clock. Each PLL sets the priority for up to three spare clocks from the remaining three input clocks, if required for switching to a redundant input, see Figure 26 and Figure 27.

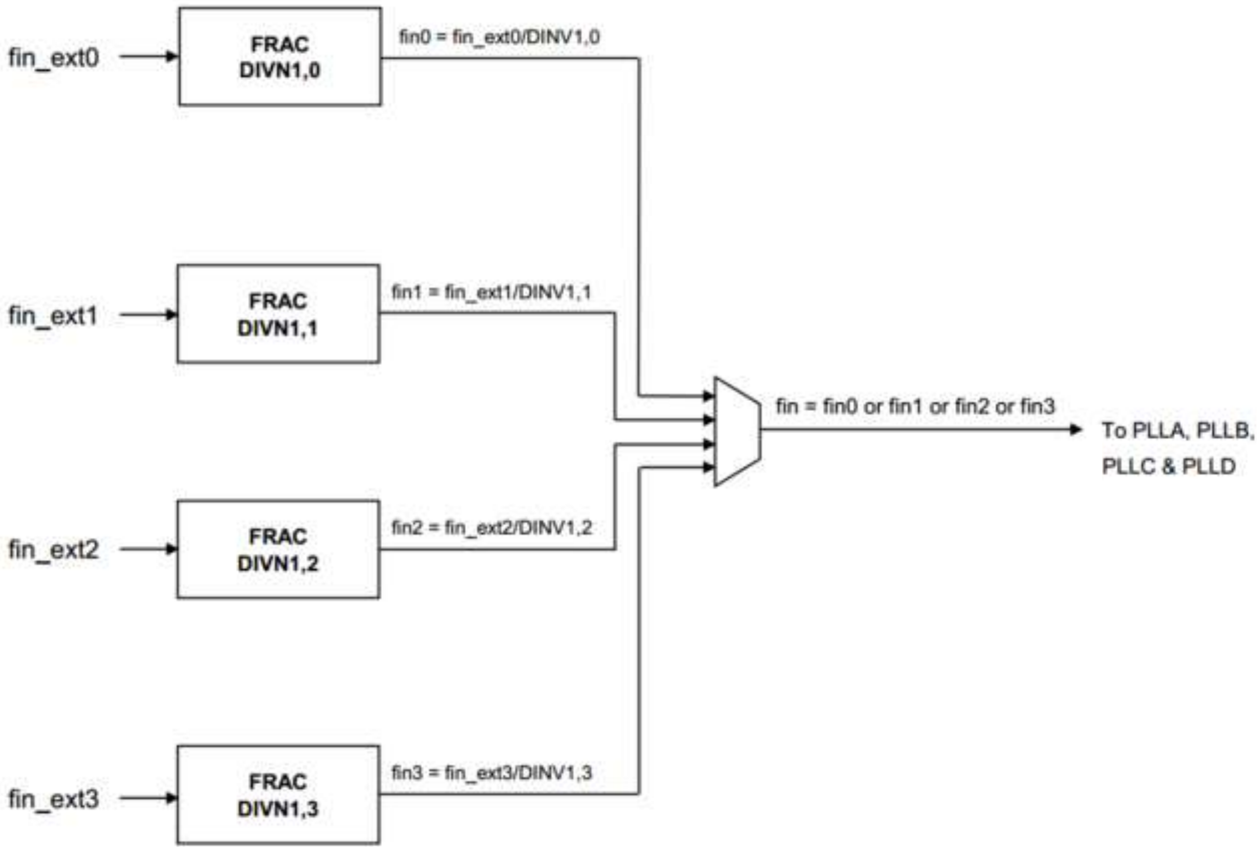


Figure 26: SiT95141 and SiT95145 input clock selection

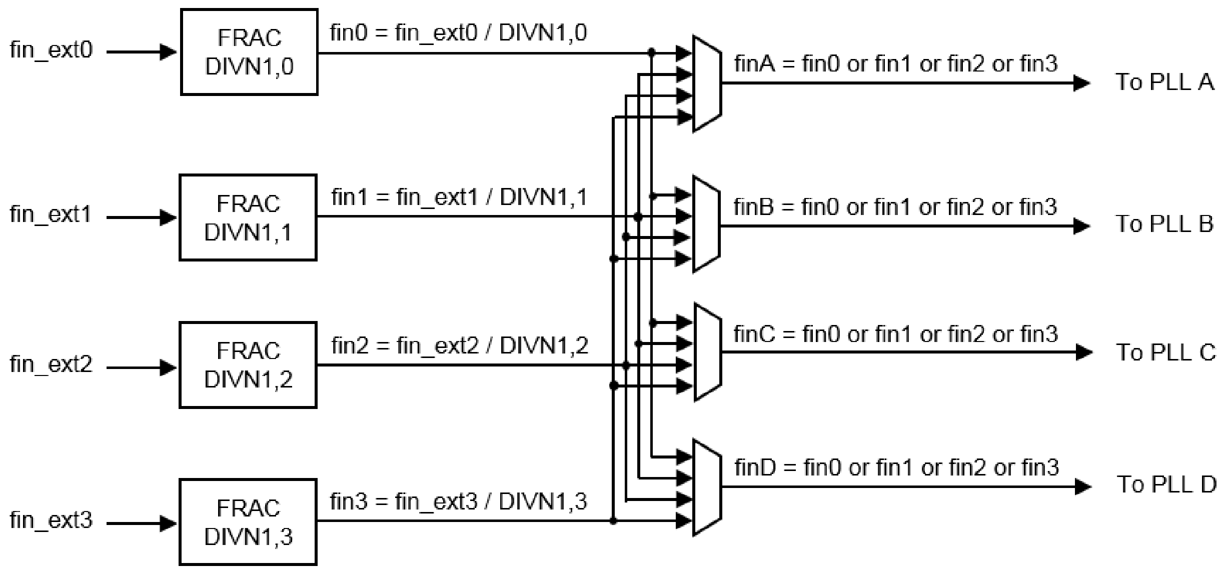

fin_ext0
FRAC DIVN1,0
fin0 = fin_ext0 / DIVN1,0
fin_ext1
FRAC DIVN1,1
fin1 = fin_ext1 / DIVN1,1
fin_ext2
FRAC DIVN1,2
fin2 = fin_ext2 / DIVN1,2
fin_ext3
FRAC DIVN1,3
fin3 = fin_ext3 / DIVN1,3
finA = fin0 or fin1 or fin2 or fin3
To PLL A
finB = fin0 or fin1 or fin2 or fin3
To PLL B
finC = fin0 or fin1 or fin2 or fin3
To PLL C
finD = fin0 or fin1 or fin2 or fin3
To PLL D

Figure 27: SiT95147 and SiT95148 input clock distribution

The **DIVN1** input dividers are internally computed by the Cascade SiTime GUI software. The **Input** section is used to set up the input frequencies and clock loss status as well as directing each input to a particular **PLL**. Further, the frequency drift monitors (with respect to the inputs) are set in this section. An example of setting the pathways for the **Input #0** clock is shown in Figure 28.

7.2 SiT95141, SiT95145 input configuration

When an input is selected, it is assigned by default to all the PLLs as input. It is added to the input order of the PLLs.

For example, if **Input #0** is selected, then **Input #0** goes to *all* the PLLs as input, which is also shown in the **Bird's Eye** section, see Figure 28.

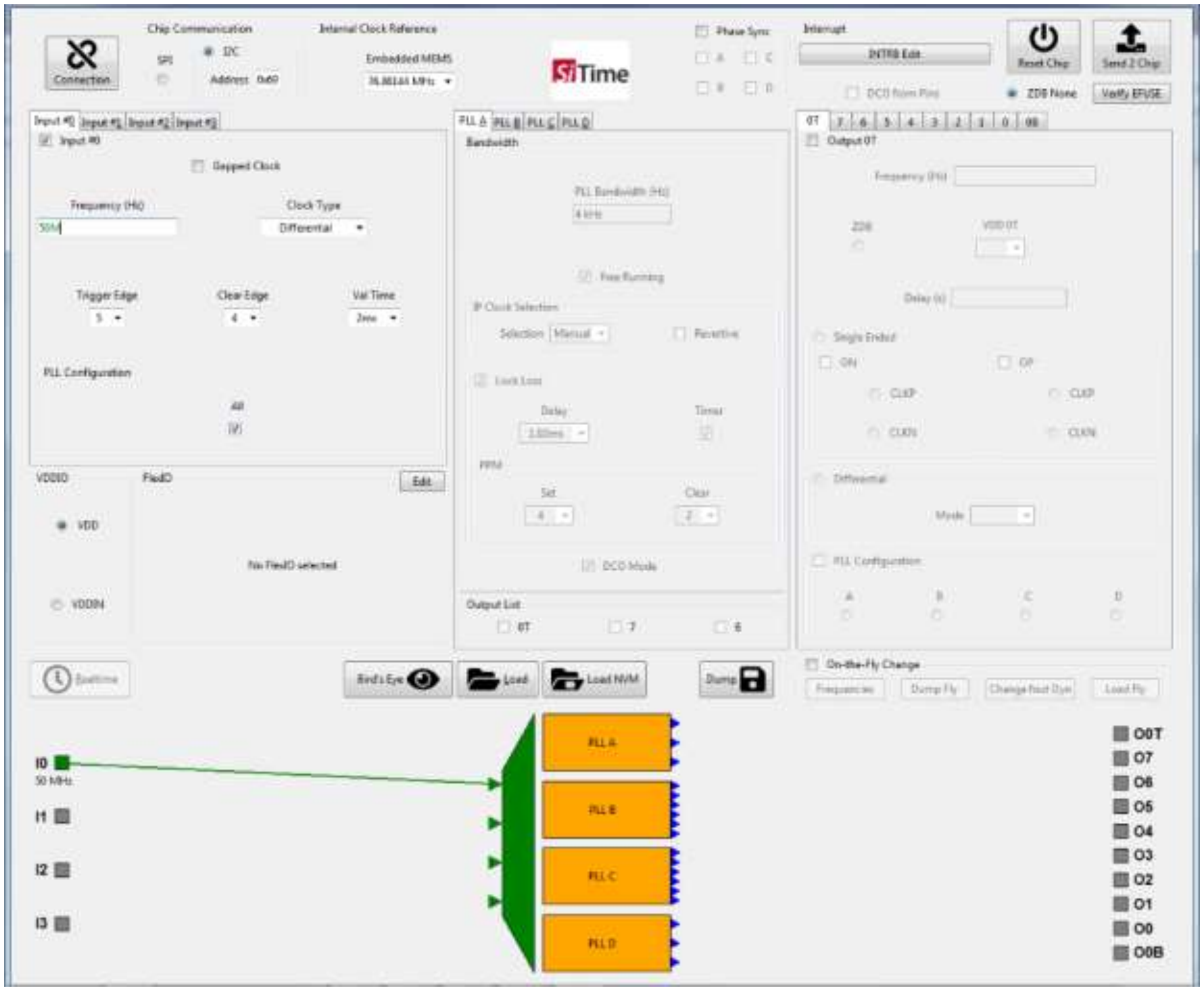


Figure 28: SiT95141 Input #0 assigned to all PLLs

7.2.1 SiT95147, SiT95148 input configuration

When an input is selected, that input is assigned to a particular PLL (A,B,C,D) as input and is added to the **Input Order** of that PLL. To select an input, click the tab for the selected **Input #(0,1,2,3)**, and then check the box.

For example, if the box for **Input #0** is checked, it becomes the selected input for **PLL A** as shown in the **Bird's Eye** view, see Figure 29.

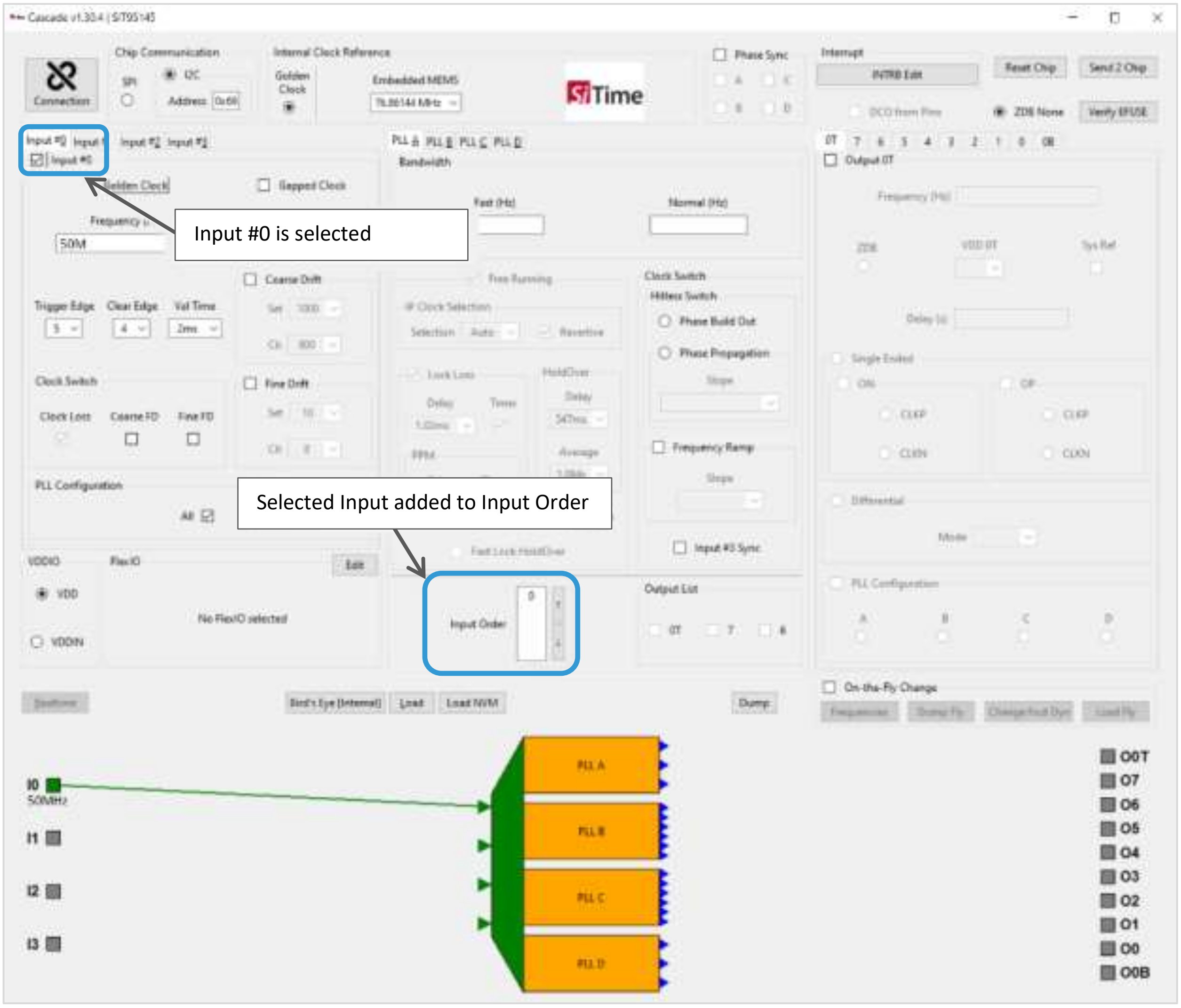
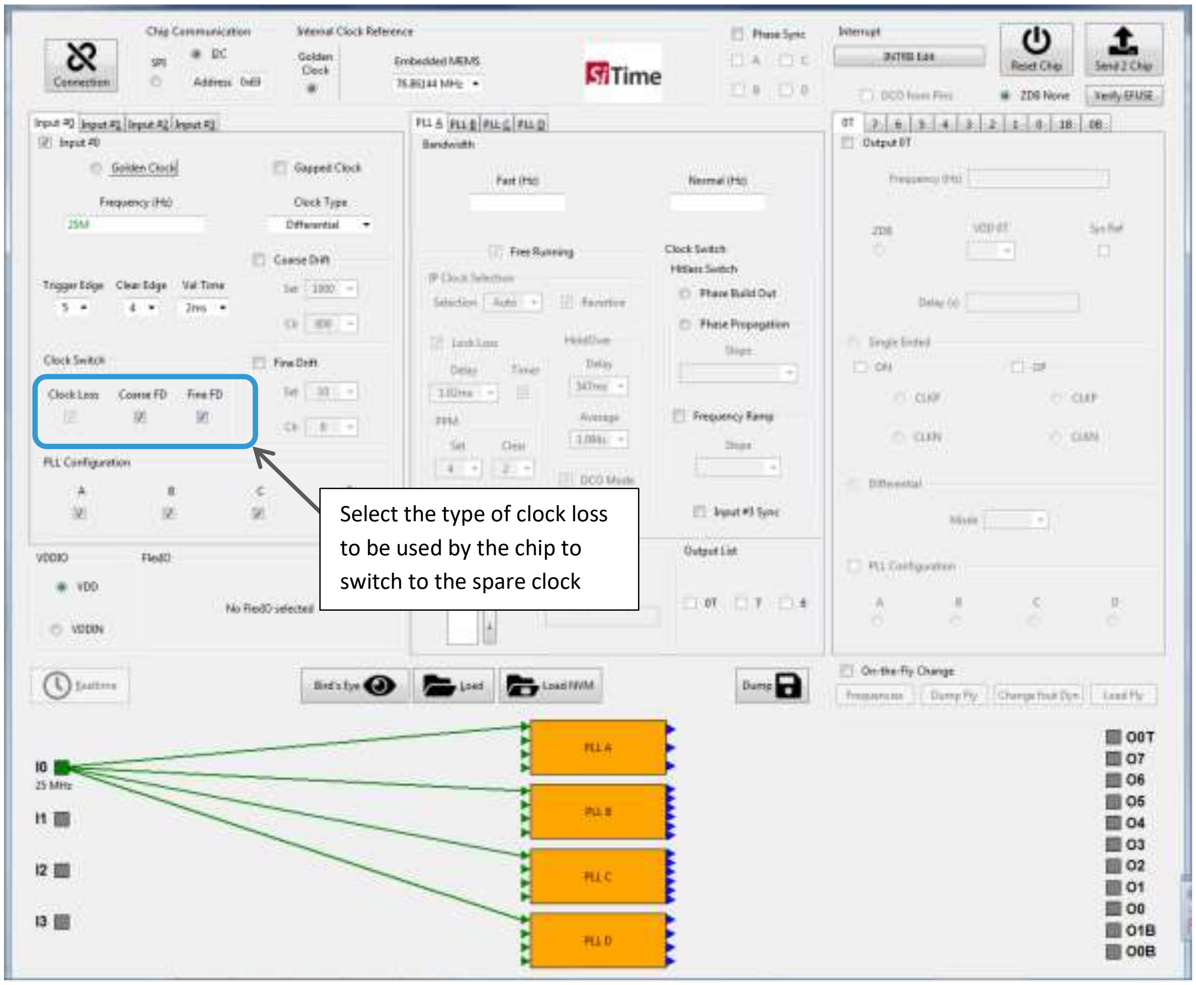


Figure 29: SiT95147 Assigning Input #0 to PLL A as input

7.2.2 SiT95148 clock loss configuration

One of four different clock loss conditions can be selected to initiate clock switchover for the PLL, as shown in Figure 30. The PLL uses the selected information from the clock loss monitors to determine whether to enter holdover or switch to another input.



Select the type of clock loss to be used by the chip to switch to the spare clock

Figure 30: SiT95148 clock loss configuration

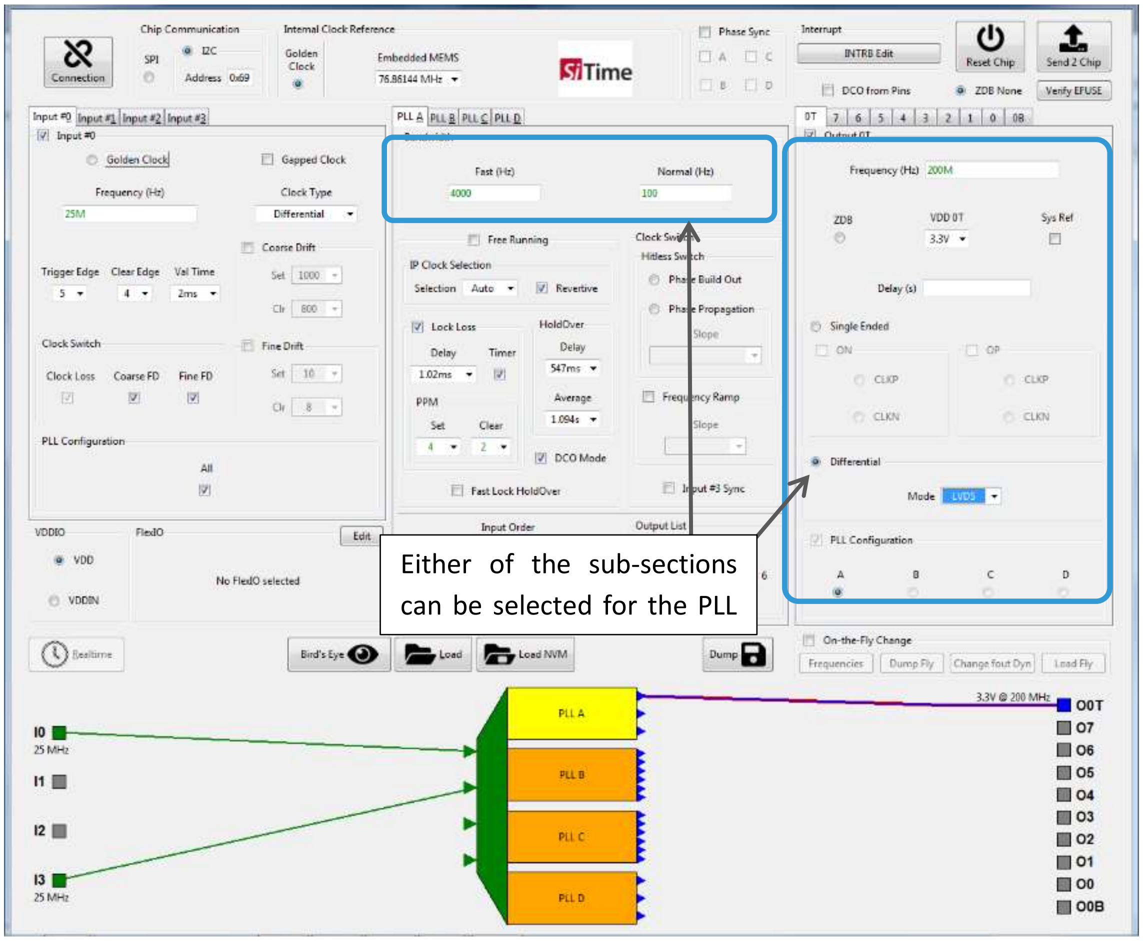
7.3 TASK 2: Set up PLL parameters

Click on a PLL and select its outputs. Then choose the PLL bandwidth, output frequency, output format (differential, LVDS mode should be selected for the SiTime evaluation board used, see Table 1). Choose the fast lock bandwidth and the regular bandwidth for the PLL. If you plan to use the DCO later, the DCO should be enabled when configuring the PLL.

Once the list of input clocks to a PLL is finalized, the input clock priority for the PLL can be changed in this section by dragging the clock selections up or down in the **Input Order** box (in the lower left area of the PLL section). The priority of the input clocks is listed in order with the highest priority clock at the top of the list and the lowest priority clock at the bottom.

For SiT95145 *only*, the input clock switching can be set to either **Manual** or **Auto** in the **Selection** drop-down menu in the **IP Clock Selection** area of the PLL section. For the **Auto** setting for the input clock switching, the revertive switching can be enabled or disabled (non-revertive) by checking the **Revertive** checkbox.

Manual clock switching is supported by changing the respective **IN_SEL1/0** pins on the SiTime SiT6503EB evaluation board, see Figure 31.


Either of the sub-sections can be selected for the PLL

Chip Communication: I2C, Address 0x69
Internal Clock Reference: Embedded MEMS 76.8144 MHz
Input #0: Frequency (Hz) 25M, Clock Type Differential, Trigger Edge 5, Clear Edge 4, Val Time 2ms
Coarse Drift: Set 1000, Clr 800
Fine Drift: Set 10, Clr 8
PLL A Bandwidth: Fast (Hz) 4000, Normal (Hz) 100
IP Clock Selection: Selection Auto, Revertive ✓
Lock Loss ✓: Delay 1.02ms, PPM Set 4, Clear 2
HoldOver: Delay 547ms, Average 1.094s
DCO Mode ✓
Output 0T: Frequency (Hz) 200M, VDD 0T 3.3V, Differential, Mode LVDS
VDDIO: VDD
FlexIO: No FlexIO selected
3.3V @ 200 MHz

Figure 31: Manual clock switching for SiT95145

The lock loss and output frequency (Fout) settings can be set in the respective sections. Go to the **Lock Loss** sub-section (in the center of the PLL section) to select the holdover (HO) history and to enable the lock loss monitor for each PLL. Note that the holdover delay determines the time back in history that the PLL goes for averaging holdover. In other words, the PLL ignores HO delay seconds of the most recent history before the clock loss event. The HO average is the time over which the frequency is averaged to arrive at the holdover frequency, see Figure 32.

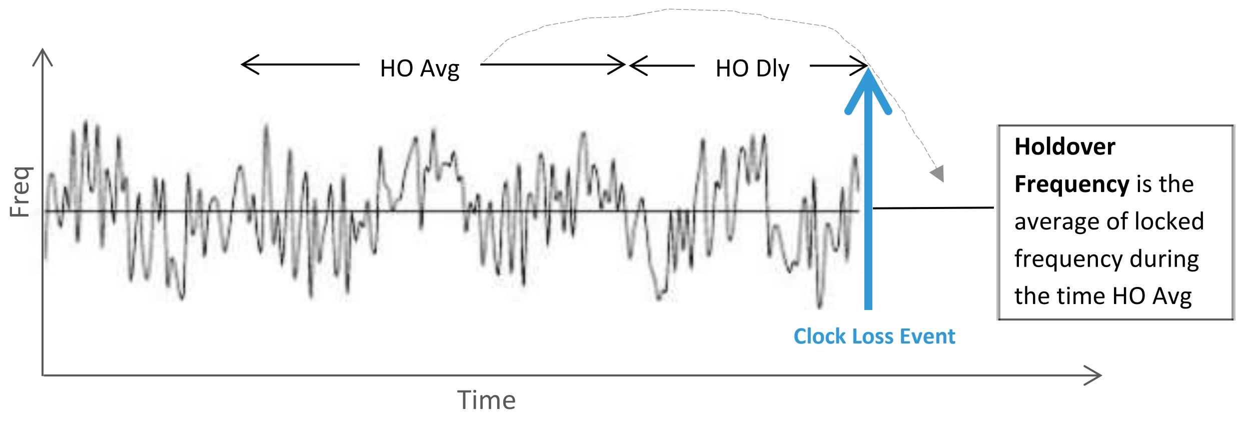

HO Avg
HO Dly
Freq
Holdover Frequency is the average of locked frequency during the time HO Avg
Clock Loss Event
Time

Figure 32: Example of the holdover history plot

Important note about the PLL bandwidth

The PLL bandwidth for both fast lock and normal bandwidth is normally recommended to be less than 1/100th and 1/500th of the PLL input frequency, respectively. Since the PLL input frequency is determined by the Cascade SiTime GUI based on the setting for the input dividers (DIVN1), the PLL bandwidth should be configured based on the internal input frequency of the PLL. The input frequency for each PLL is displayed in the background **Realtime** terminal that runs along with the Cascade SiTime GUI. Additionally, a pop-up warning message appears if a PLL bandwidth is chosen such that fast lock and normal bandwidth are larger than the normally recommended bound of 1/100th and 1/500th of the PLL input frequency.

7.4 TASK 3: Save or load UI configuration and program the SiT9514x device

7.4.1 Saving and loading the UI configuration file

Press the **Send 2 Chip** button (top-right) to save the UI configuration file. The GUI will prompt you to save the UI configuration file in .py format to a user specified directory, see Figure 33. With the **Load** button, this file can be used to load the saved configuration later as necessary.

This is the preferred method to save the configuration profile.

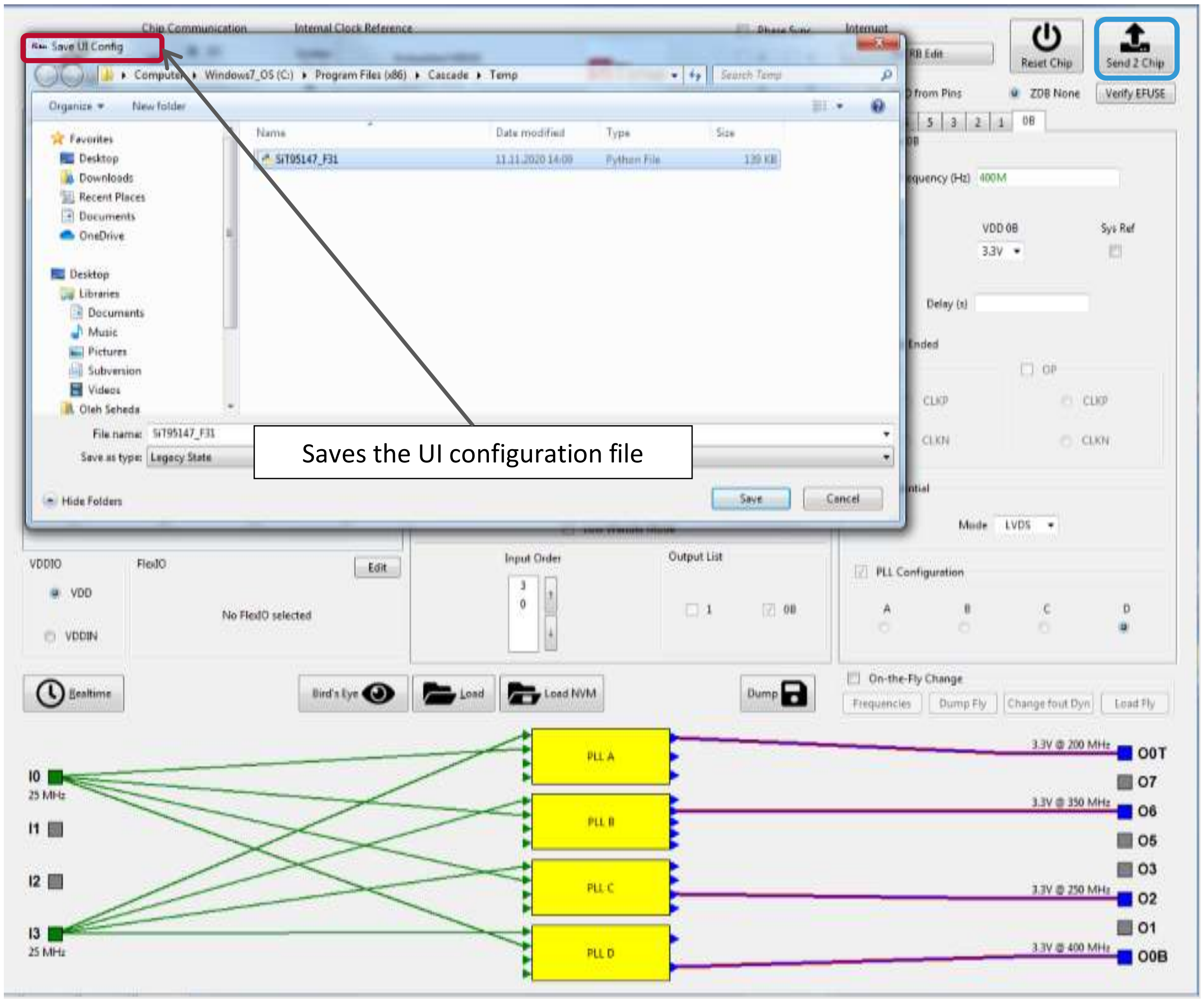
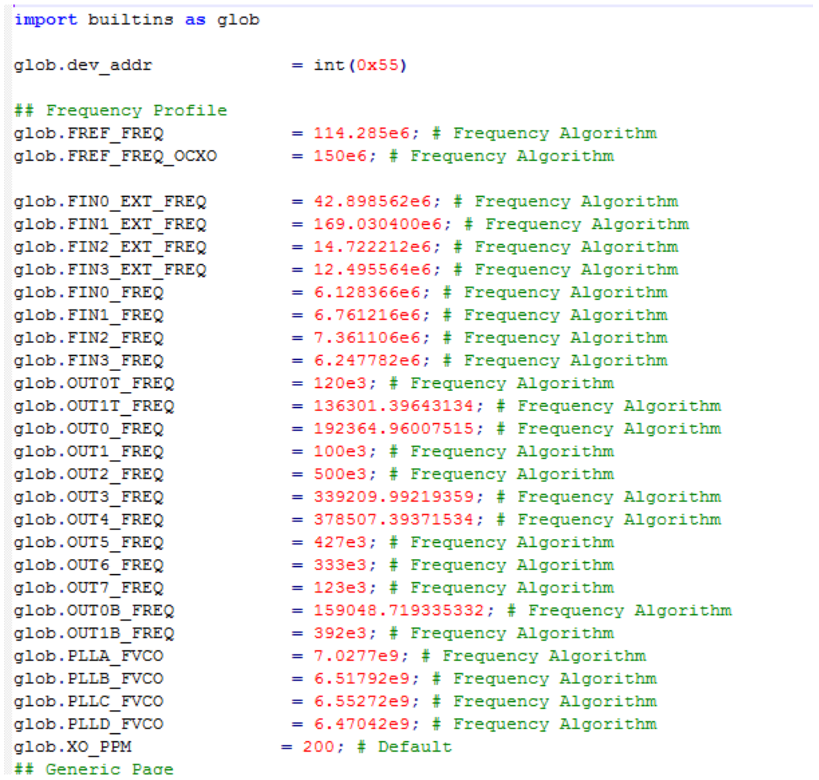


Figure 33: Saving the UI configuration profile file

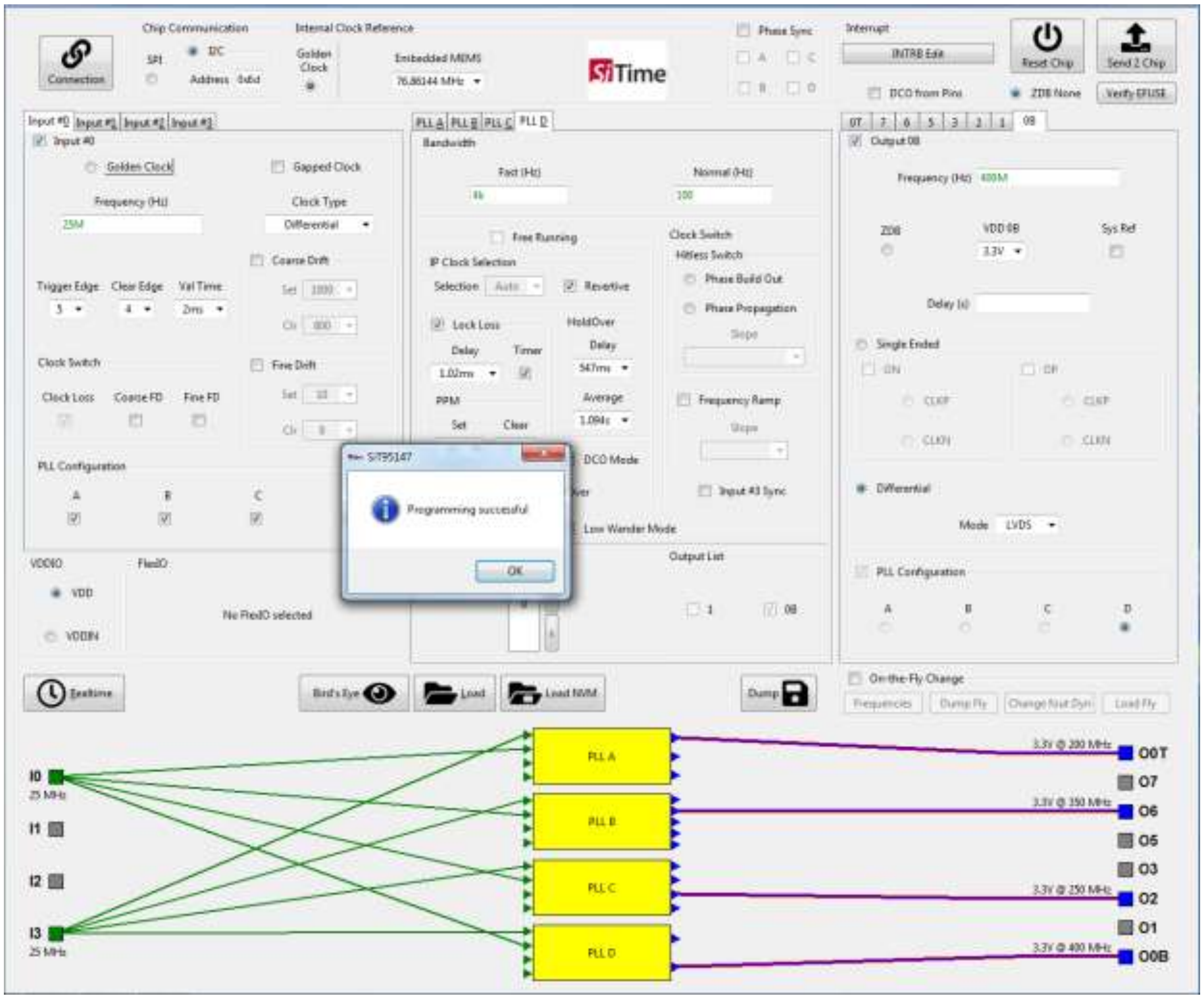
Figure 29 shows an example of a Cascade SiTime GUI configuration profile file.



```
import builtins as glob

glob.dev_addr              = int(0x55)

## Frequency Profile
glob.FREF_FREQ             = 114.285e6; # Frequency Algorithm
glob.FREF_FREQ_OCXO        = 150e6; # Frequency Algorithm

glob.FIN0_EXT_FREQ         = 42.898562e6; # Frequency Algorithm
glob.FIN1_EXT_FREQ         = 169.030400e6; # Frequency Algorithm
glob.FIN2_EXT_FREQ         = 14.722212e6; # Frequency Algorithm
glob.FIN3_EXT_FREQ         = 12.495564e6; # Frequency Algorithm
glob.FIN0_FREQ             = 6.128366e6; # Frequency Algorithm
glob.FIN1_FREQ             = 6.761216e6; # Frequency Algorithm
glob.FIN2_FREQ             = 7.361106e6; # Frequency Algorithm
glob.FIN3_FREQ             = 6.247782e6; # Frequency Algorithm
glob.OUT0T_FREQ            = 120e3; # Frequency Algorithm
glob.OUT1T_FREQ            = 136301.39643134; # Frequency Algorithm
glob.OUT0_FREQ             = 192364.96007515; # Frequency Algorithm
glob.OUT1_FREQ             = 100e3; # Frequency Algorithm
glob.OUT2_FREQ             = 500e3; # Frequency Algorithm
glob.OUT3_FREQ             = 339209.99219359; # Frequency Algorithm
glob.OUT4_FREQ             = 378507.39371534; # Frequency Algorithm
glob.OUT5_FREQ             = 427e3; # Frequency Algorithm
glob.OUT6_FREQ             = 333e3; # Frequency Algorithm
glob.OUT7_FREQ             = 123e3; # Frequency Algorithm
glob.OUT0B_FREQ            = 159048.719335332; # Frequency Algorithm
glob.OUT1B_FREQ            = 392e3; # Frequency Algorithm
glob.PLLA_FVCO             = 7.0277e9; # Frequency Algorithm
glob.PLLB_FVCO             = 6.51792e9; # Frequency Algorithm
glob.PLLC_FVCO             = 6.55272e9; # Frequency Algorithm
glob.PLLD_FVCO             = 6.47042e9; # Frequency Algorithm
glob.XO_PPM              = 200; # Default
## Generic Page
```

Figure 34: Example content of the configuration profile file

Click the **Send 2 Chip** button (top-right) to program the chip with the current configuration or a previously loaded configuration profile, see Figure 35.



SiT95147

Programming successful

OK

Figure 35: Example of successful programming of a SiT95147

7.4.2 Using the dump function

The dump function can be used to save the NVM (or I2C/SPI writes) file containing the sequence of register writes for any profile.

NOTE: Depending on the **Chip Communication** selected, **I2C** or **SPI**, one of the following NVM file types is created:

- If **I2C** is selected, then the NVM (I2C write) file is created.
- If **SPI** is selected, then the NVM (SPI write) file is created.

To dump the NVM (or I2C/SPI writes) file, the file must first be activated using the **Send 2 Chip** button, even if the SiTime evaluation board is *not* connected. This runs the algorithms that optimize the internal configuration based on the required inputs and outputs. After this, the **Dump** button can be used to save the list of register writes, see Figure 36.

Note, the **Send 2 Chip** process *also* provides the option of saving the UI configuration profile file, but this does not need to be saved *if* the user is only interested in the NVM files.

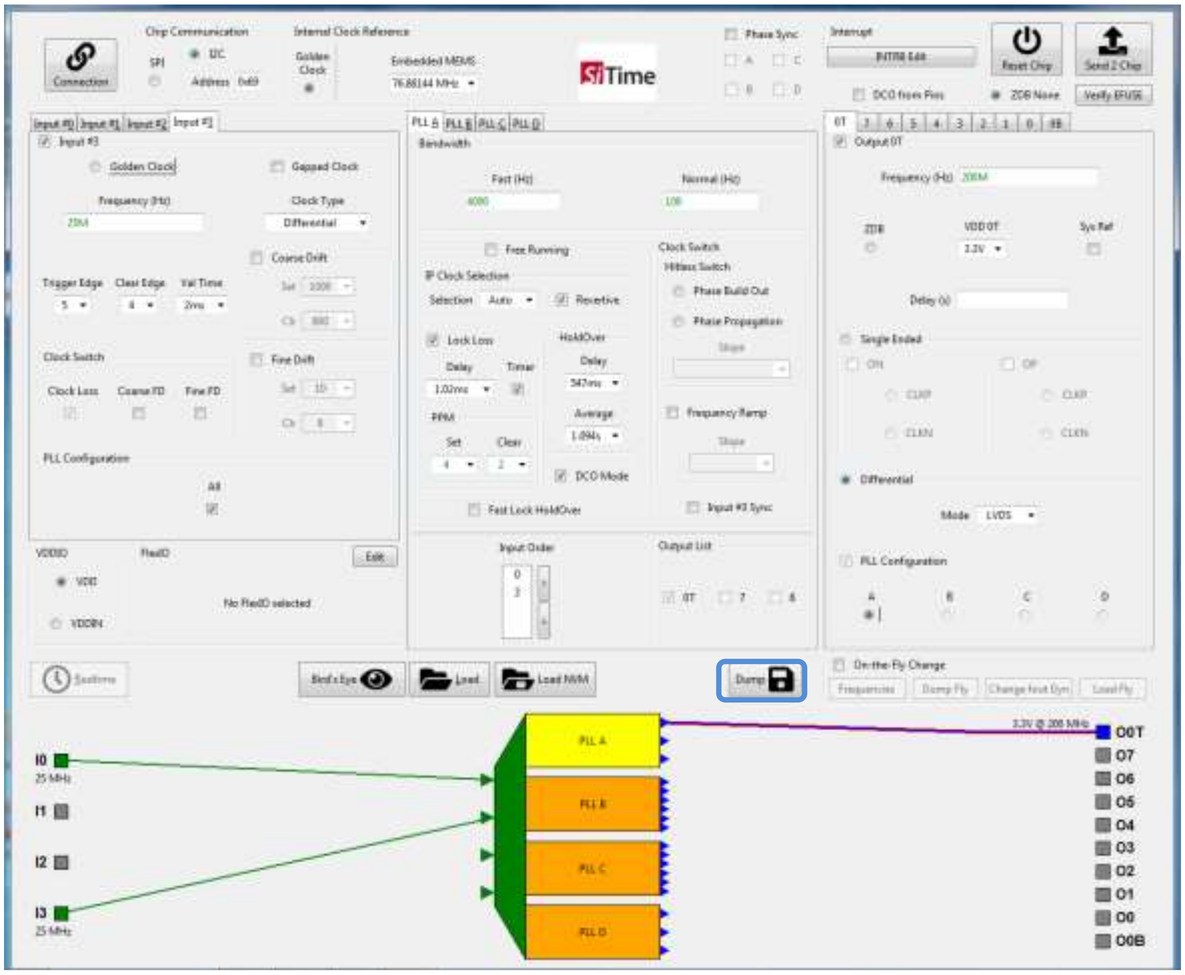


Figure 36: The “Dump” button can be used to save the list of register writes

When the **Dump** button is clicked, the user is prompted to save the configuration to a file. Select the NVM (I2C/SPI write) option from the drop down. Use the **Legacy State [NVM] (*.NVM)** option in this case, which allows the user to save the file to any directory location needed, see Figure 37.

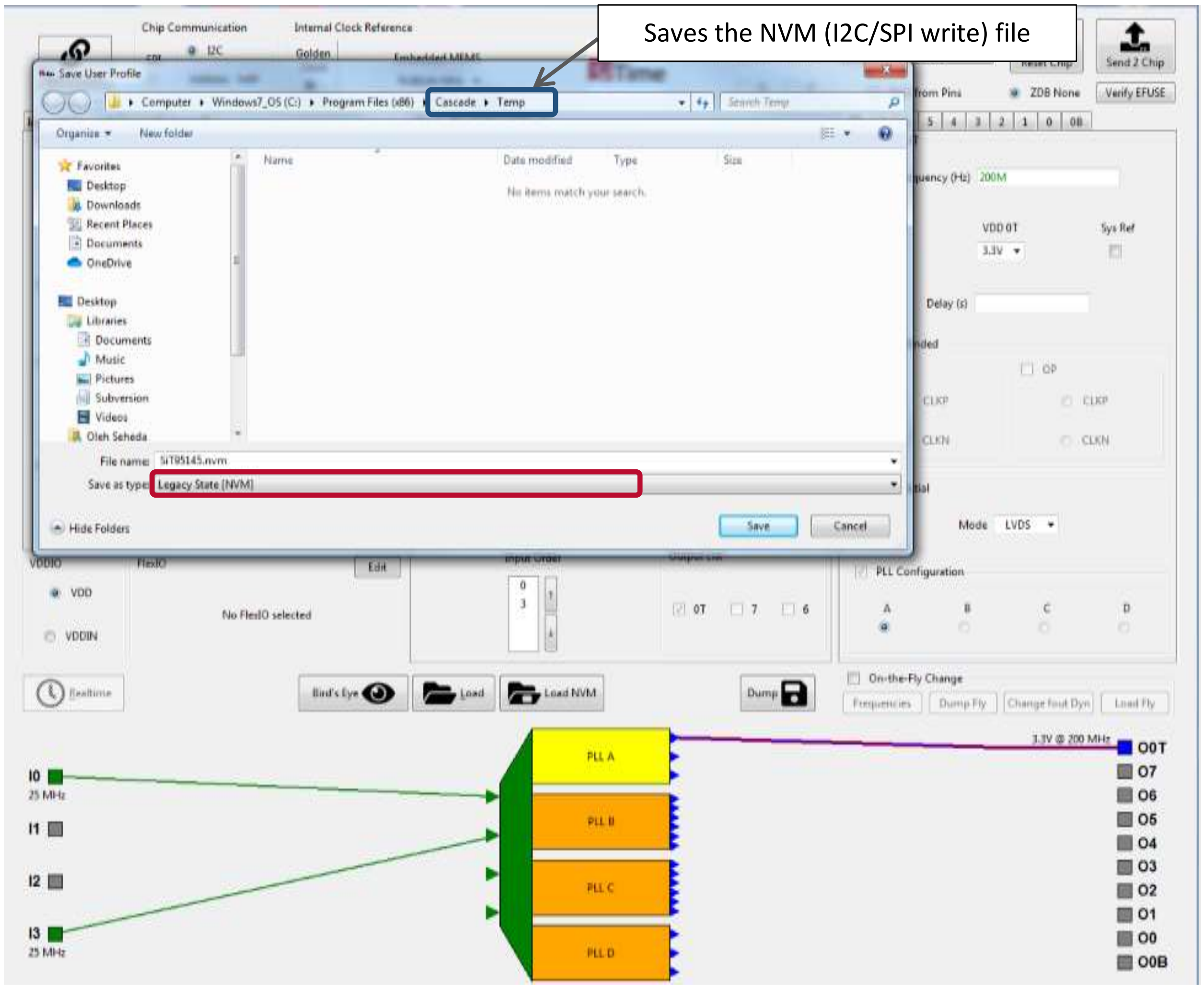
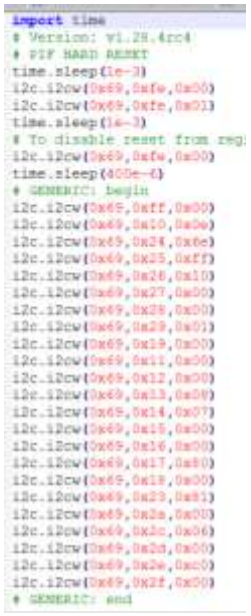


Figure 37: Saving the NVM

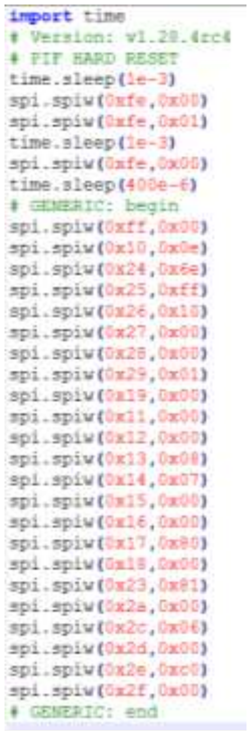
Figure 38 shows an example of an NVM (I2C write) file.



```
import time
# Version: v1.28.4rc4
# PIF HARD RESET
time.sleep(1e-3)
i2c.i2cw(0x69,0xfe,0x00)
i2c.i2cw(0x69,0xfe,0x01)
time.sleep(1e-3)
# To disable reset from reg
i2c.i2cw(0x69,0xfe,0x00)
time.sleep(400e-6)
# GENERIC: begin
i2c.i2cw(0x69,0xff,0x00)
i2c.i2cw(0x69,0x10,0x0e)
i2c.i2cw(0x69,0x24,0x6e)
i2c.i2cw(0x69,0x25,0xff)
i2c.i2cw(0x69,0x26,0x10)
i2c.i2cw(0x69,0x27,0x00)
i2c.i2cw(0x69,0x28,0x00)
i2c.i2cw(0x69,0x29,0x01)
i2c.i2cw(0x69,0x19,0x00)
i2c.i2cw(0x69,0x11,0x00)
i2c.i2cw(0x69,0x12,0x00)
i2c.i2cw(0x69,0x13,0x08)
i2c.i2cw(0x69,0x14,0x07)
i2c.i2cw(0x69,0x15,0x00)
i2c.i2cw(0x69,0x16,0x00)
i2c.i2cw(0x69,0x17,0x80)
i2c.i2cw(0x69,0x18,0x00)
i2c.i2cw(0x69,0x23,0x81)
i2c.i2cw(0x69,0x2a,0x00)
i2c.i2cw(0x69,0x2c,0x06)
i2c.i2cw(0x69,0x2d,0x00)
i2c.i2cw(0x69,0x2e,0xc0)
i2c.i2cw(0x69,0x2f,0x00)
# GENERIC: end
```

Figure 38: Example NVM (I2C write) file

Figure 39 shows an example of an NVM (SPI write).



```
import time
# Version: v1.28.4rc4
# PIF HARD RESET
time.sleep(1e-3)
spi.spiw(0xfe,0x00)
spi.spiw(0xfe,0x01)
time.sleep(1e-3)
spi.spiw(0xfe,0x00)
time.sleep(400e-6)
# GENERIC: begin
spi.spiw(0xff,0x00)
spi.spiw(0x10,0x0e)
spi.spiw(0x24,0x6e)
spi.spiw(0x25,0xff)
spi.spiw(0x26,0x10)
spi.spiw(0x27,0x00)
spi.spiw(0x28,0x00)
spi.spiw(0x29,0x01)
spi.spiw(0x19,0x00)
spi.spiw(0x11,0x00)
spi.spiw(0x12,0x00)
spi.spiw(0x13,0x08)
spi.spiw(0x14,0x07)
spi.spiw(0x15,0x00)
spi.spiw(0x16,0x00)
spi.spiw(0x17,0x80)
spi.spiw(0x18,0x00)
spi.spiw(0x23,0x81)
spi.spiw(0x2a,0x00)
spi.spiw(0x2c,0x06)
spi.spiw(0x2d,0x00)
spi.spiw(0x2e,0xc0)
spi.spiw(0x2f,0x00)
# GENERIC: end
```

Figure 39: Example NVM (SPI write) file

The NVM file can be directly written to the chip using the **Load NVM** button, but it cannot be used to load the UI configuration profile.

7.4.3 Saving efuse.NVM.py (I2C/SPI) files

Select the NVM (I2C/SPI write) option from the drop down. Use the **EFUSE Locking (*efuse.NVM.py)** option for this case. This allows the user to save the file to any directory, see Figure 40.

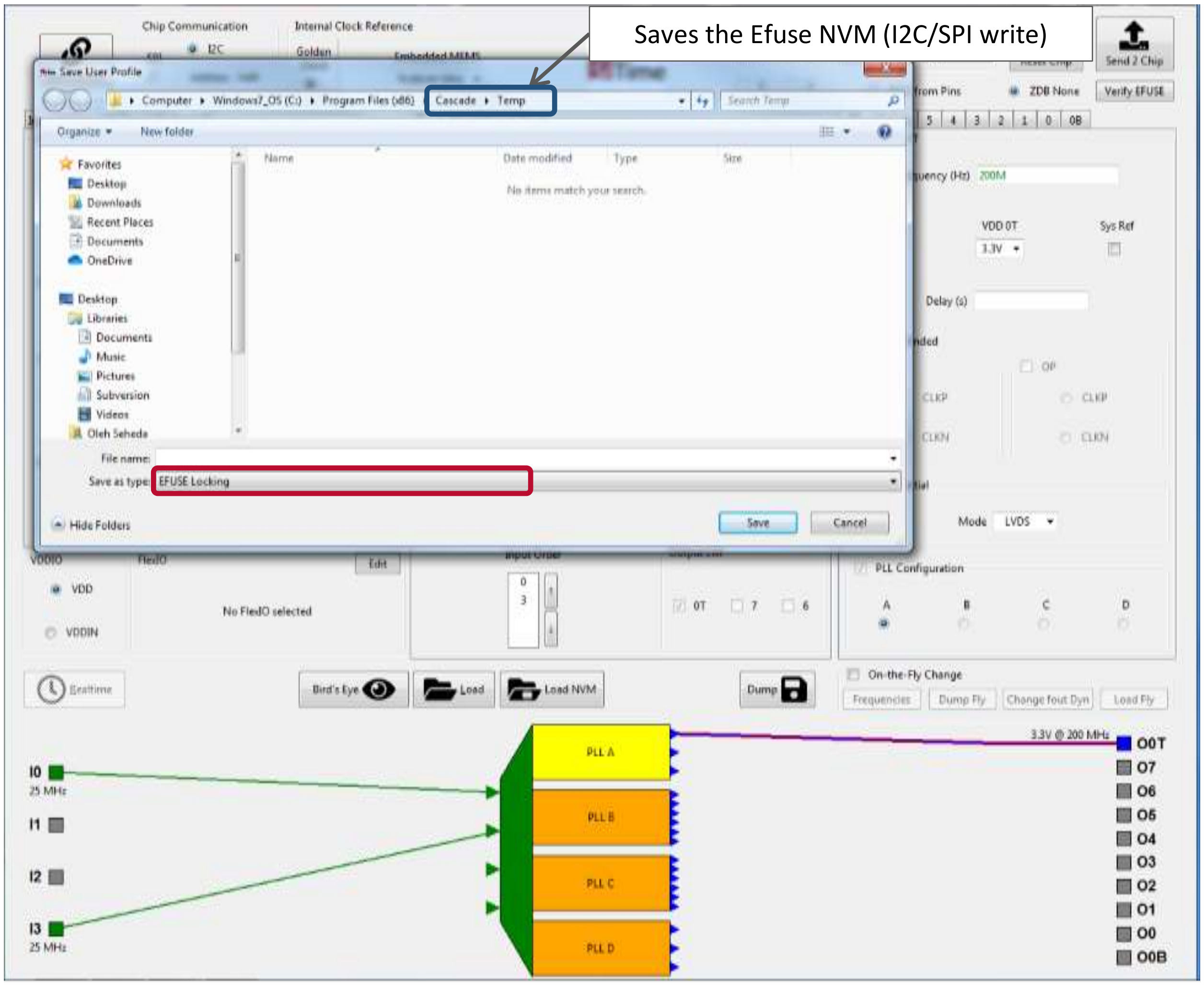
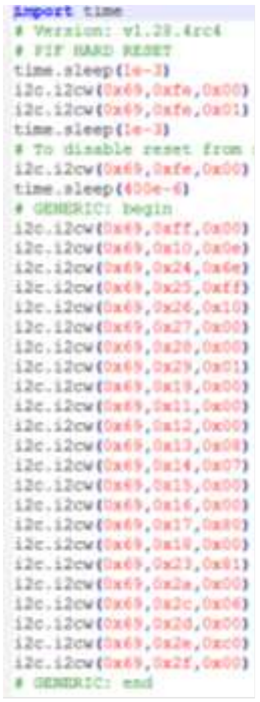


Figure 40: Saving Efuse NVM(I2C/SPI) files

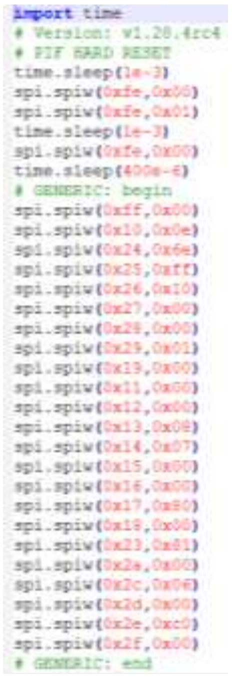
Figure 41 shows an example of an Efuse NVM (I2C write) file.



```
import time
# Version: v1.28.4rc4
# PIF HARD RESET
time.sleep(1e-3)
i2c.i2cw(0x69,0xfe,0x00)
i2c.i2cw(0x69,0xfe,0x01)
time.sleep(1e-3)
# To disable reset from
i2c.i2cw(0x69,0xfe,0x00)
time.sleep(400e-6)
# GENERIC: begin
i2c.i2cw(0x69,0xff,0x00)
i2c.i2cw(0x69,0x10,0x0e)
i2c.i2cw(0x69,0x24,0x6e)
i2c.i2cw(0x69,0x25,0xff)
i2c.i2cw(0x69,0x26,0x10)
i2c.i2cw(0x69,0x27,0x00)
i2c.i2cw(0x69,0x28,0x00)
i2c.i2cw(0x69,0x29,0x01)
i2c.i2cw(0x69,0x19,0x00)
i2c.i2cw(0x69,0x11,0x00)
i2c.i2cw(0x69,0x12,0x00)
i2c.i2cw(0x69,0x13,0x08)
i2c.i2cw(0x69,0x14,0x07)
i2c.i2cw(0x69,0x15,0x00)
i2c.i2cw(0x69,0x16,0x00)
i2c.i2cw(0x69,0x17,0x80)
i2c.i2cw(0x69,0x18,0x00)
i2c.i2cw(0x69,0x23,0x81)
i2c.i2cw(0x69,0x2a,0x00)
i2c.i2cw(0x69,0x2c,0x06)
i2c.i2cw(0x69,0x2d,0x00)
i2c.i2cw(0x69,0x2e,0xc0)
i2c.i2cw(0x69,0x2f,0x00)
# GENERIC: end
```

Figure 41: Example Efuse NVM(I2C write) file

Figure 42 shows an example of an Efuse NVM (SPI write) file.



```
import time
# Version: v1.28.4rc4
# PIF HARD RESET
time.sleep(1e-3)
spi.spiw(0xfe,0x00)
spi.spiw(0xfe,0x01)
time.sleep(1e-3)
spi.spiw(0xfe,0x00)
time.sleep(400e-6)
# GENERIC: begin
spi.spiw(0xff,0x00)
spi.spiw(0x10,0x0e)
spi.spiw(0x24,0x6e)
spi.spiw(0x25,0xff)
spi.spiw(0x26,0x10)
spi.spiw(0x27,0x00)
spi.spiw(0x28,0x00)
spi.spiw(0x29,0x01)
spi.spiw(0x19,0x00)
spi.spiw(0x11,0x00)
spi.spiw(0x12,0x00)
spi.spiw(0x13,0x08)
spi.spiw(0x14,0x07)
spi.spiw(0x15,0x00)
spi.spiw(0x16,0x00)
spi.spiw(0x17,0x80)
spi.spiw(0x18,0x00)
spi.spiw(0x23,0x81)
spi.spiw(0x2a,0x00)
spi.spiw(0x2c,0x06)
spi.spiw(0x2d,0x00)
spi.spiw(0x2e,0xc0)
spi.spiw(0x2f,0x00)
# GENERIC: end
```

Figure 42: Example Efuse NVM (SPI write) file

7.4.4 Saving the Cascade SiTime GUI state

The state of the Cascade SiTime GUI (i.e. the values of the widgets) can be saved as a **.json** file.

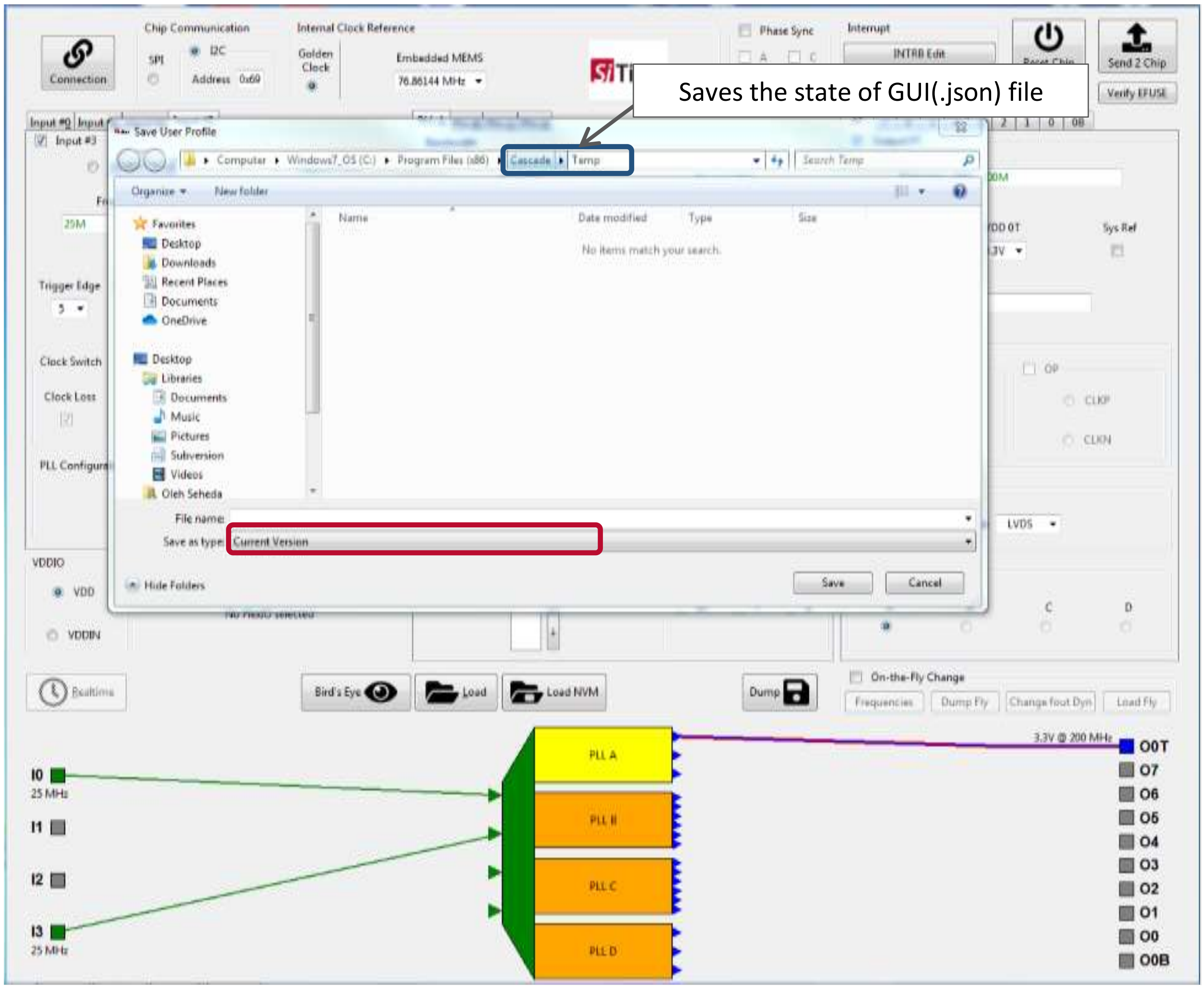


Figure 43: Saving the state of the Cascade SiTime GUI

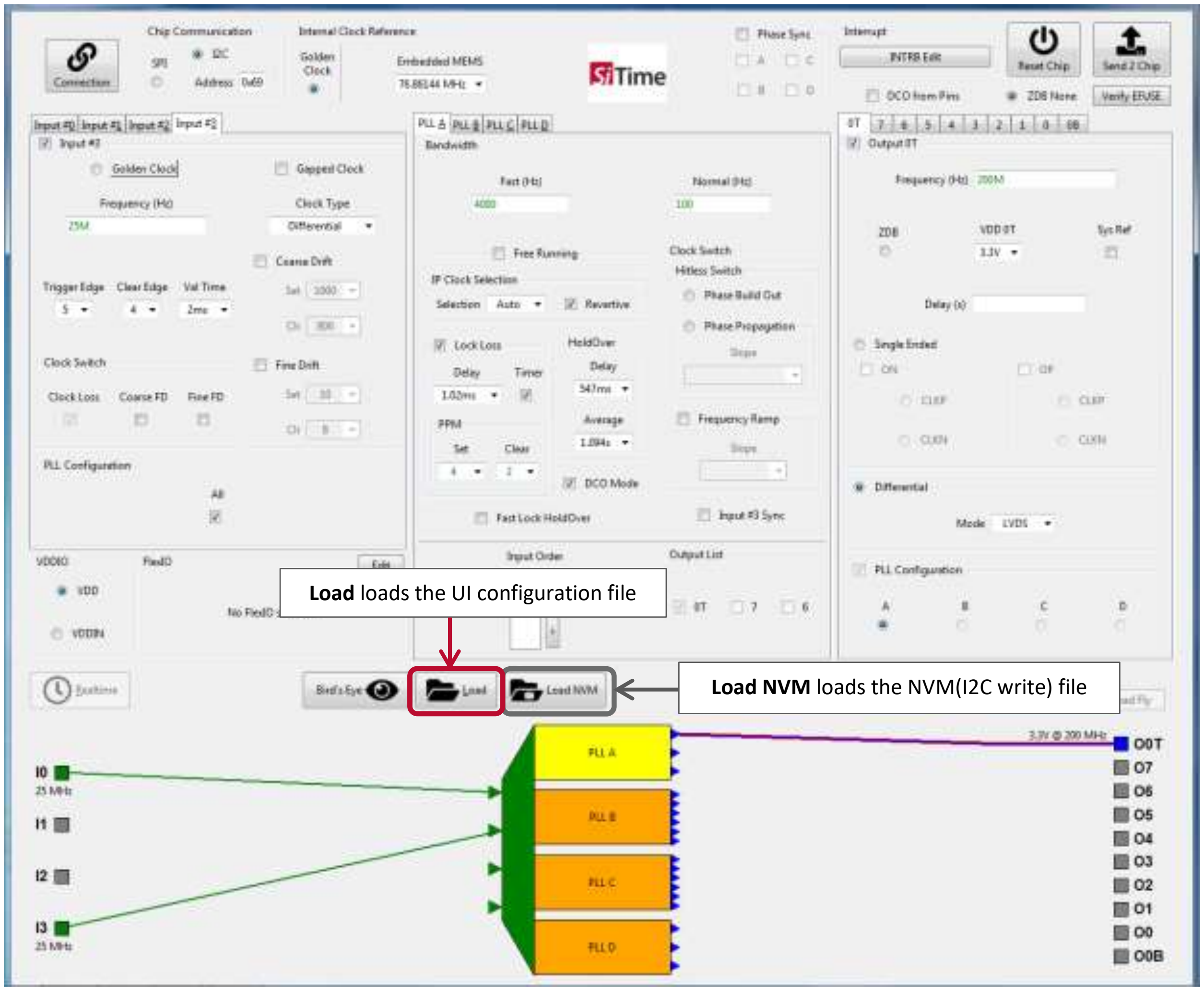


Figure 44: Loading files using the Cascade SiTime GUI

7.4.5 Using the load NVM function

When the **Load** button is pressed, an open file dialog window pops up. Navigate to select the Cascade SiTime GUI configuration file, see Figure 45.

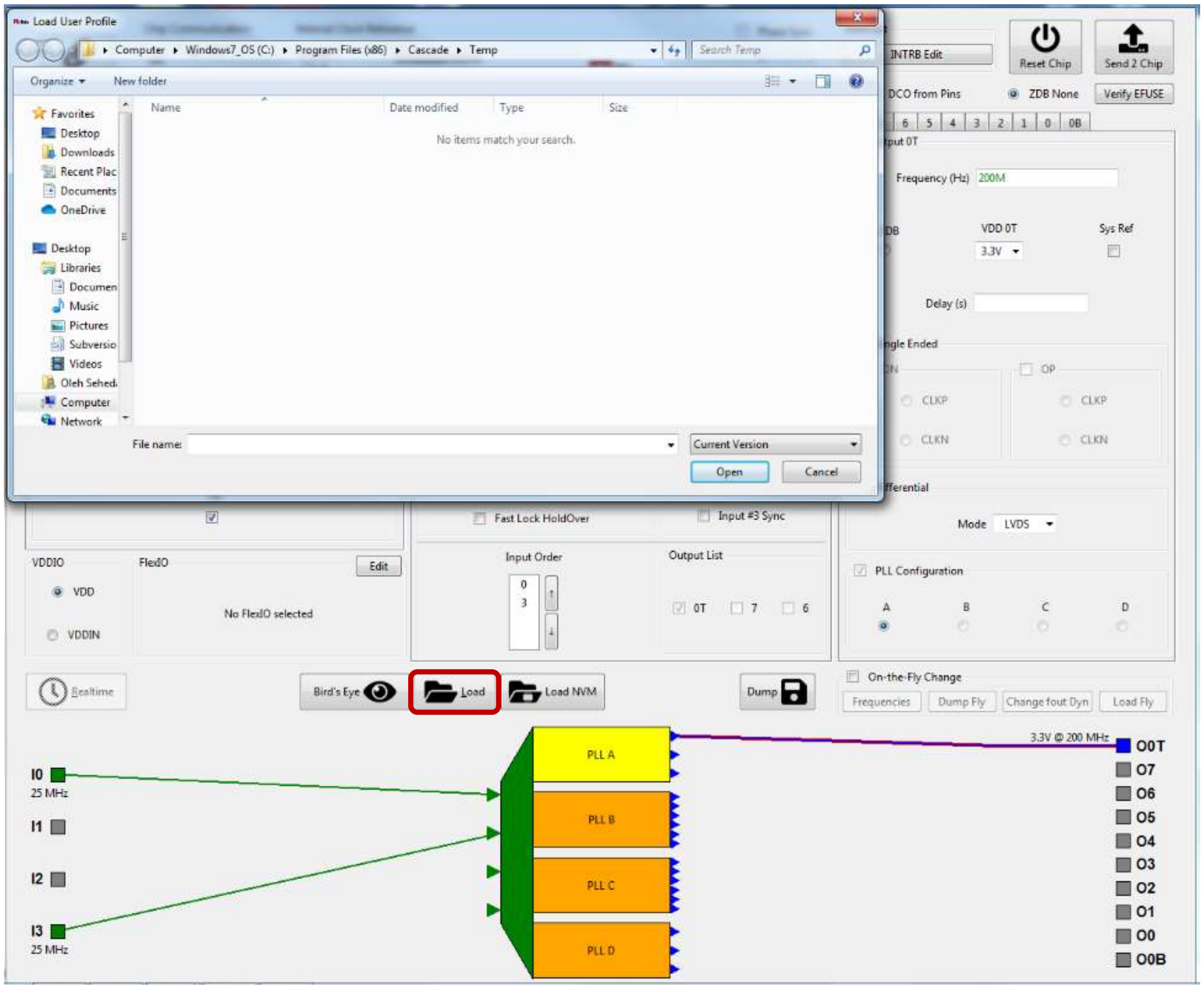


Figure 45: Using the Load function

Note, the state of the Cascade SiTime GUI (.json) file can be loaded using the **Load** button.

When a Cascade SiTime GUI configuration file is loaded, the programmed configuration is shown in the GUI.

See Figure 46 for an example showing a basic SiT95141 variant configuration. Examples for other SiT9514x variants are shown in the subsequent figures.

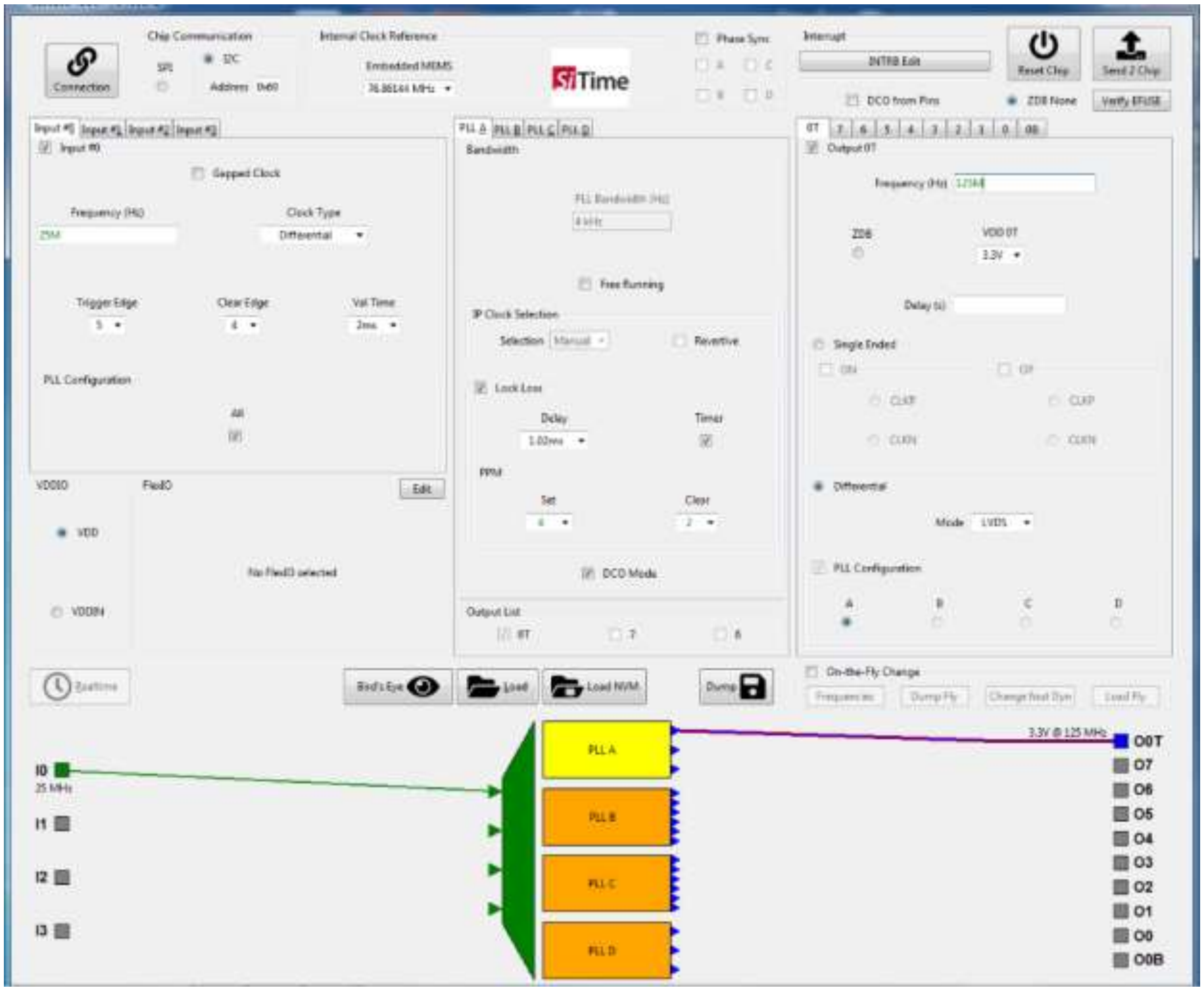

Chip Communication: SPI, I2C (selected), Address 0x68
Internal Clock Reference: Embedded MEMS, 76.8 MHz
Input #0: Frequency (Hz) 25M, Clock Type Differential, Trigger Edge 5, Clear Edge 4, Val Time 2ms
VDDIO: VDD (selected), VDDIN
No FlexIO selected
PLL A: PLL Bandwidth (Hz) 4 kHz, Selection Manual, Lock Loss: Delay 1.02ms, PPM Set 4, Clear 2, DCO Mode
Output 0T: Frequency (Hz) 125M, VDD 0T 3.3V, Differential, Mode LVDS, PLL Configuration A
ZDB None
3.3V @ 125 MHz

Figure 46: Showing the programmed configuration for SiT95141

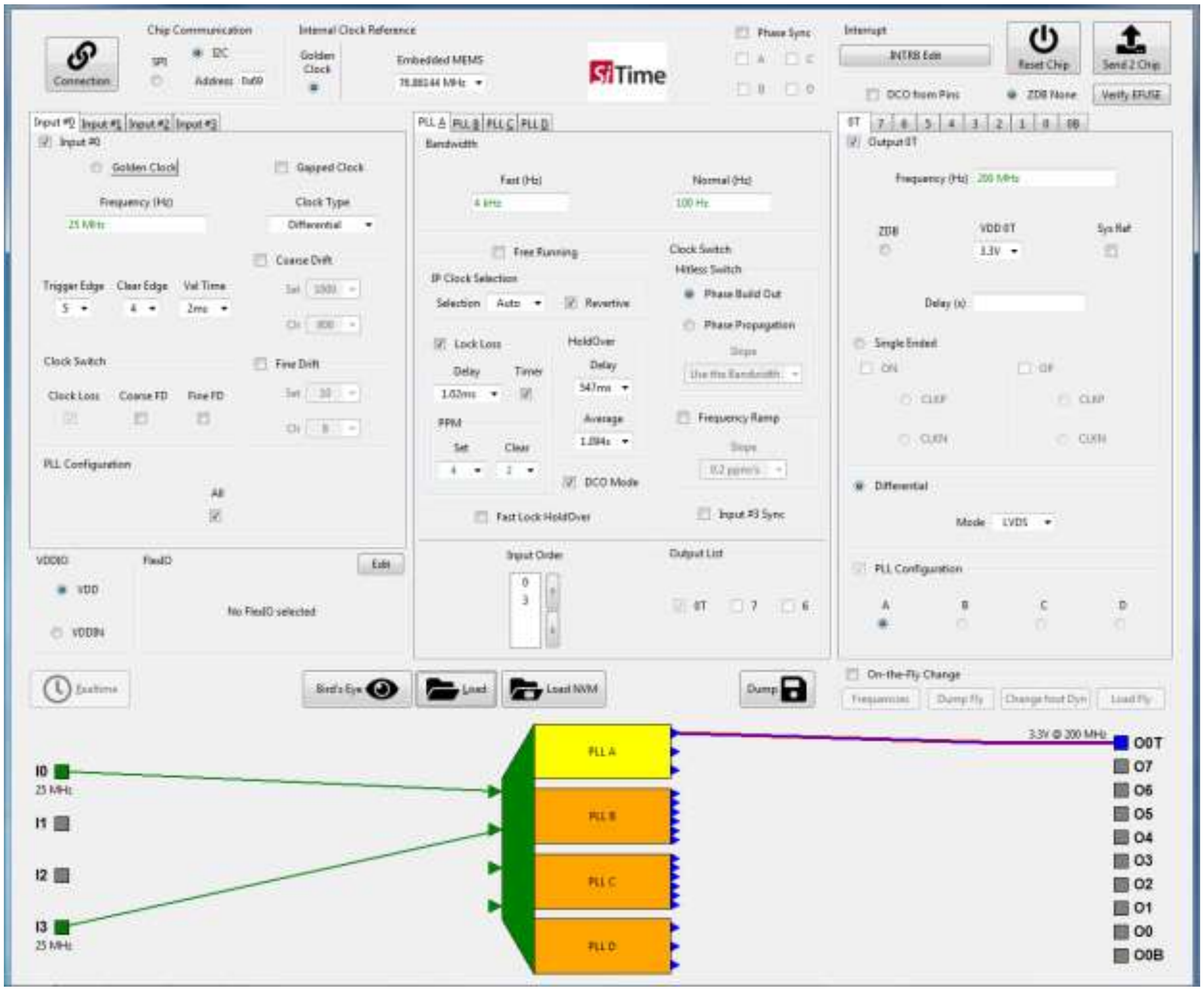
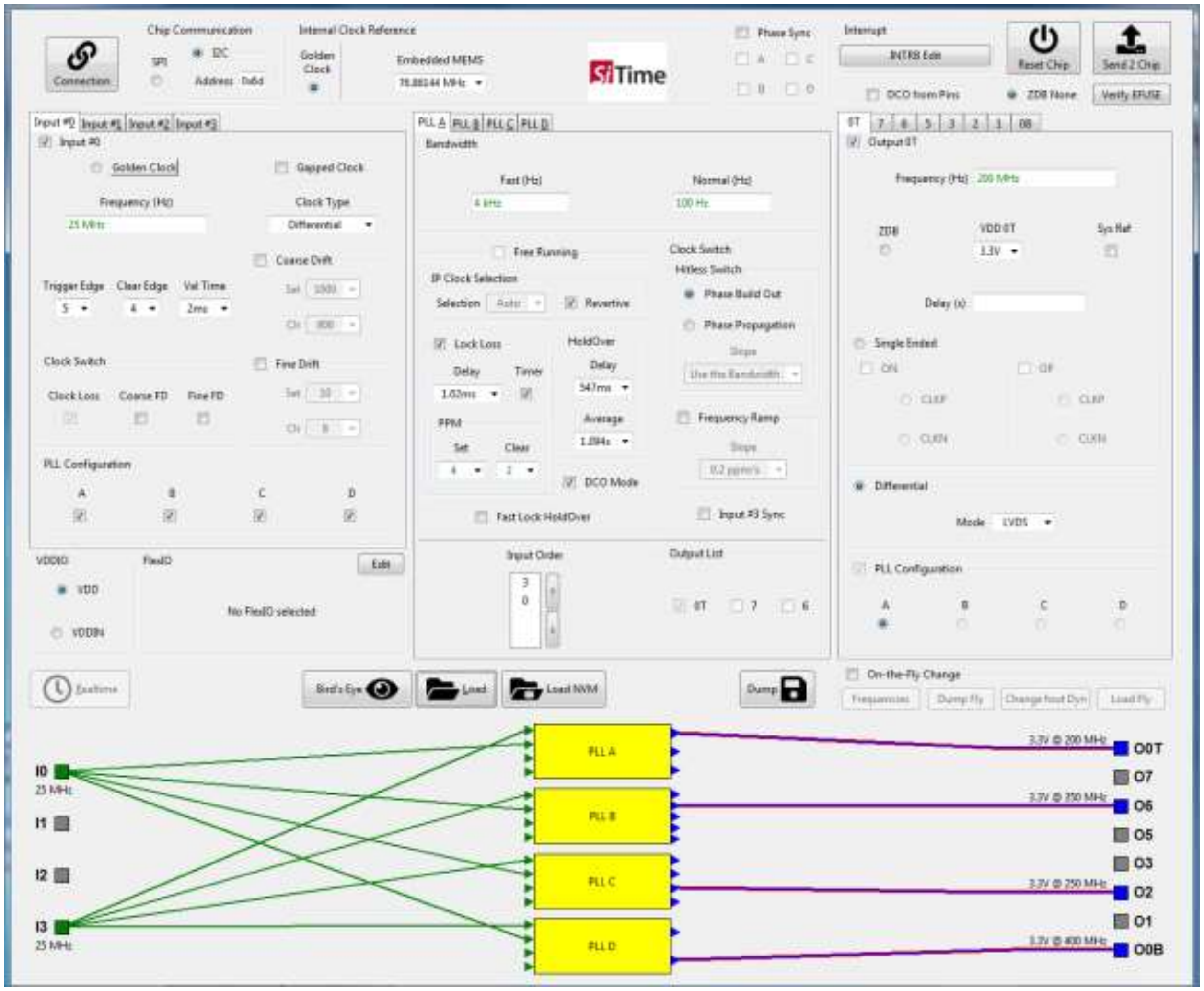


Figure 47: Showing the programmed configuration for SiT95145



Chip Communication: SPI, [✓ I2C], Address: 0x6d
Internal Clock Reference: Golden Clock [✓], Embedded MEMS: 78.88144 MHz
Phase Sync: A, C, B, D
Interrupt
DCO from Pins
[✓ ZDB None]

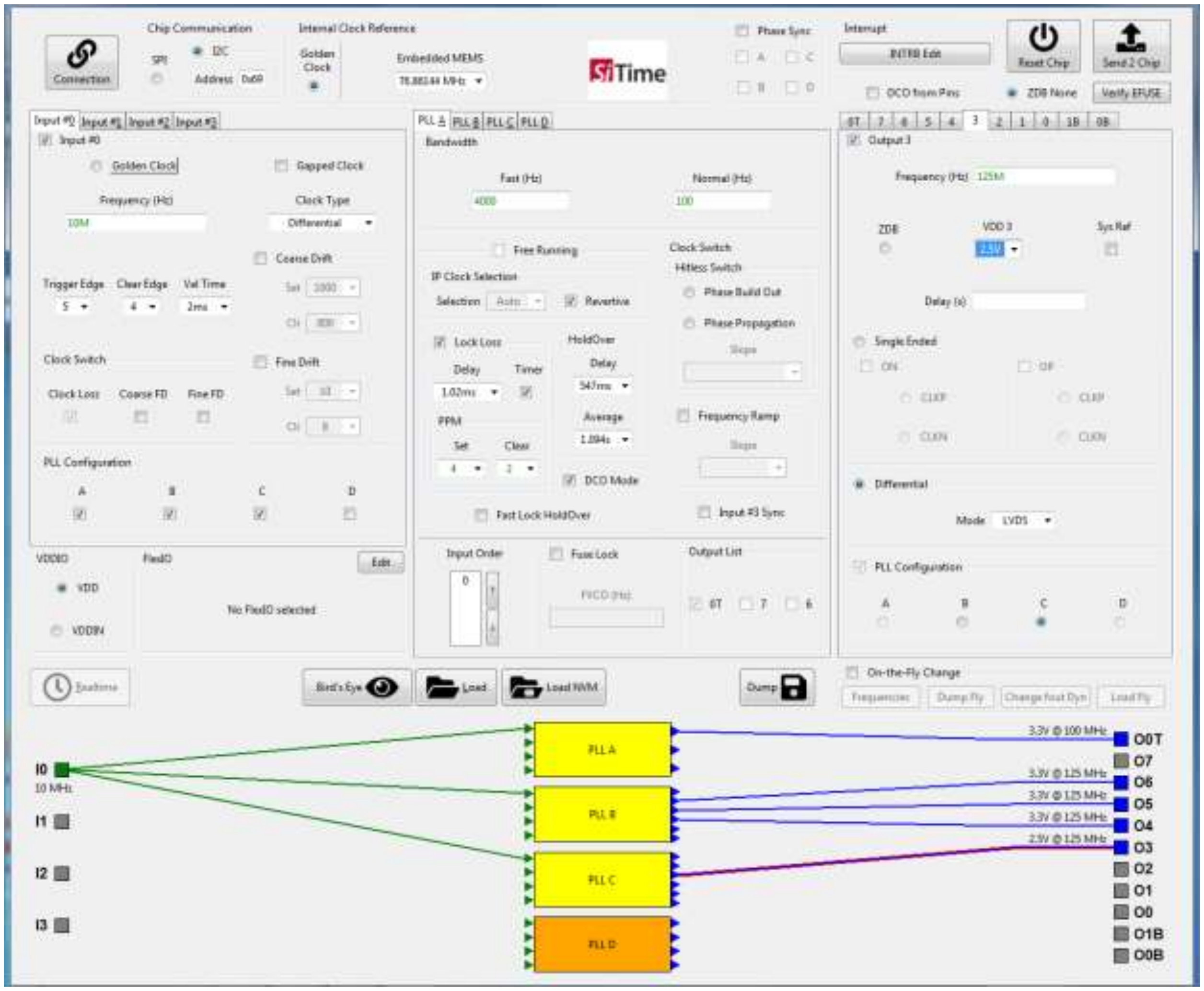
Input #0 [✓]
Golden Clock
Gapped Clock
Frequency (Hz): 25 MHz
Clock Type: Differential
Coarse Drift: Set 1000, Clr 800
Trigger Edge: 5
Clear Edge: 4
Val Time: 2ms
Clock Switch
Clock Loss, Coarse FD, Fine FD
Fine Drift: Set 10, Clr 8
PLL Configuration: A [✓], B [✓], C [✓], D [✓]
VDDIO: [✓ VDD], VDDIN
FlexIO: No FlexIO selected

PLL A
Bandwidth
Fast (Hz): 4 kHz
Normal (Hz): 100 Hz
Free Running
IP Clock Selection
Selection: Auto
Revertive [✓]
Lock Loss [✓]
Delay: 1.02ms
Timer [✓]
PPM
Set: 4
Clear: 2
HoldOver
Delay: 547ms
Average: 1.094s
DCO Mode [✓]
Fast Lock HoldOver
Clock Switch
Hitless Switch
[✓ Phase Build Out]
Phase Propagation
Slope: Use the Bandwidth
Frequency Ramp
Slope: 0.2 ppm/s
Input #3 Sync
Input Order: 3, 0
Output List: 0T, 7, 6

Output 0T [✓]
Frequency (Hz): 200 MHz
ZDB
VDD 0T: 3.3V
Sys Ref
Delay (s):
Single Ended
ON: CLKP, CLKN
OP: CLKP, CLKN
[✓ Differential]
Mode: LVDS
PLL Configuration: [✓ A], B, C, D
On-the-Fly Change

I0 25 MHz
I1
I2
I3 25 MHz
PLL A
PLL B
PLL C
PLL D
3.3V @ 200 MHz O0T
O7
3.3V @ 350 MHz O6
O5
O3
3.3V @ 250 MHz O2
O1
3.3V @ 400 MHz O0B

Figure 48: Showing the programmed configuration for SiT95147



Chip Communication: SPI [✓ I2C], Address 0x68

Internal Clock Reference: [✓ Golden Clock]; Embedded MEMS 78.88344 MHz

Phase Sync: ☐ A ☐ B ☐ C ☐ D

Interrupt

☐ DCO from Pins [✓ ZDB None]

Input #0 | Input #1 | Input #2 | Input #3

☑ Input #0

Golden Clock ☐ Gapped Clock

Frequency (Hz): 10M Clock Type: Differential

☐ Coarse Drift Set 1000 Clr 800

Trigger Edge 5 Clear Edge 4 Val Time 2ms

Clock Switch: Clock Loss ☑ Coarse FD ☐ Fine FD ☐

☐ Fine Drift Set 10 Clr 8

PLL Configuration: A ☑ B ☑ C ☑ D ☐

VDDIO: [✓ VDD] VDDIN

FlexIO: No FlexIO selected

PLL A | PLL B | PLL C | PLL D

Bandwidth: Fast (Hz) 4000 Normal (Hz) 100

☐ Free Running

IP Clock Selection: Selection Auto ☑ Revertive

☑ Lock Loss: Delay 1.02ms Timer ☑

PPM: Set 4 Clear 2

HoldOver: Delay 547ms Average 1.094s

☑ DCO Mode

☐ Fast Lock HoldOver

Clock Switch: Hitless Switch: Phase Build Out / Phase Propagation; Slope

☐ Frequency Ramp Slope

☐ Input #3 Sync

Input Order: 0

☐ Fuse Lock FVCO (Hz)

Output List: ☑ 0T ☐ 7 ☐ 6

0T | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 1B | 0B

☑ Output 3

Frequency (Hz): 125M

ZDB VDD 3: 2.5V Sys Ref ☐

Delay (s):

Single Ended: ☐ ON (CLKP, CLKN) ☐ OP (CLKP, CLKN)

[✓ Differential] Mode: LVDS

PLL Configuration: A B [✓ C] D

☐ On-the-Fly Change

Figure 49: Showing the programmed configuration for SiT95148

7.4.6 Using the load NVM function

Load NVM loads the NVM (I2C sequence of writes) file.

When the **Load NVM** button is clicked, it prompts the user to load an NVM (I2C write) file, see Figure 50.

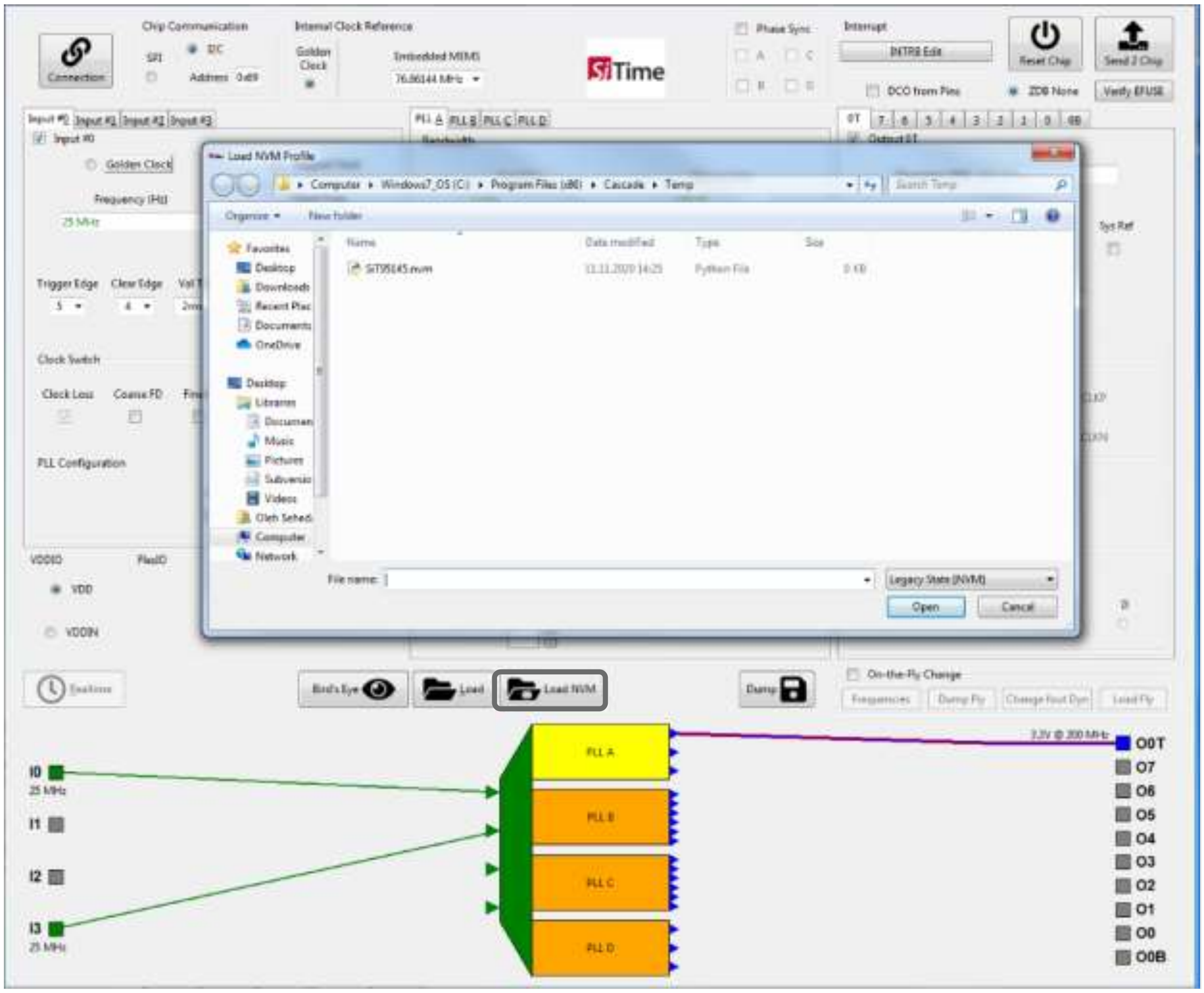


Figure 50: Loading the NVM (I2C sequence of writes) file

Once the file is loaded and after the chip is programmed, the **Realtime** section can be used, see Figure 51.

NOTE: If the SiTime evaluation board is not connected, then none of the on-the-fly functions will work. But the Realtime window can be useful to check the estimated current consumption for the selected configuration, even with the SiTime evaluation board disconnected.

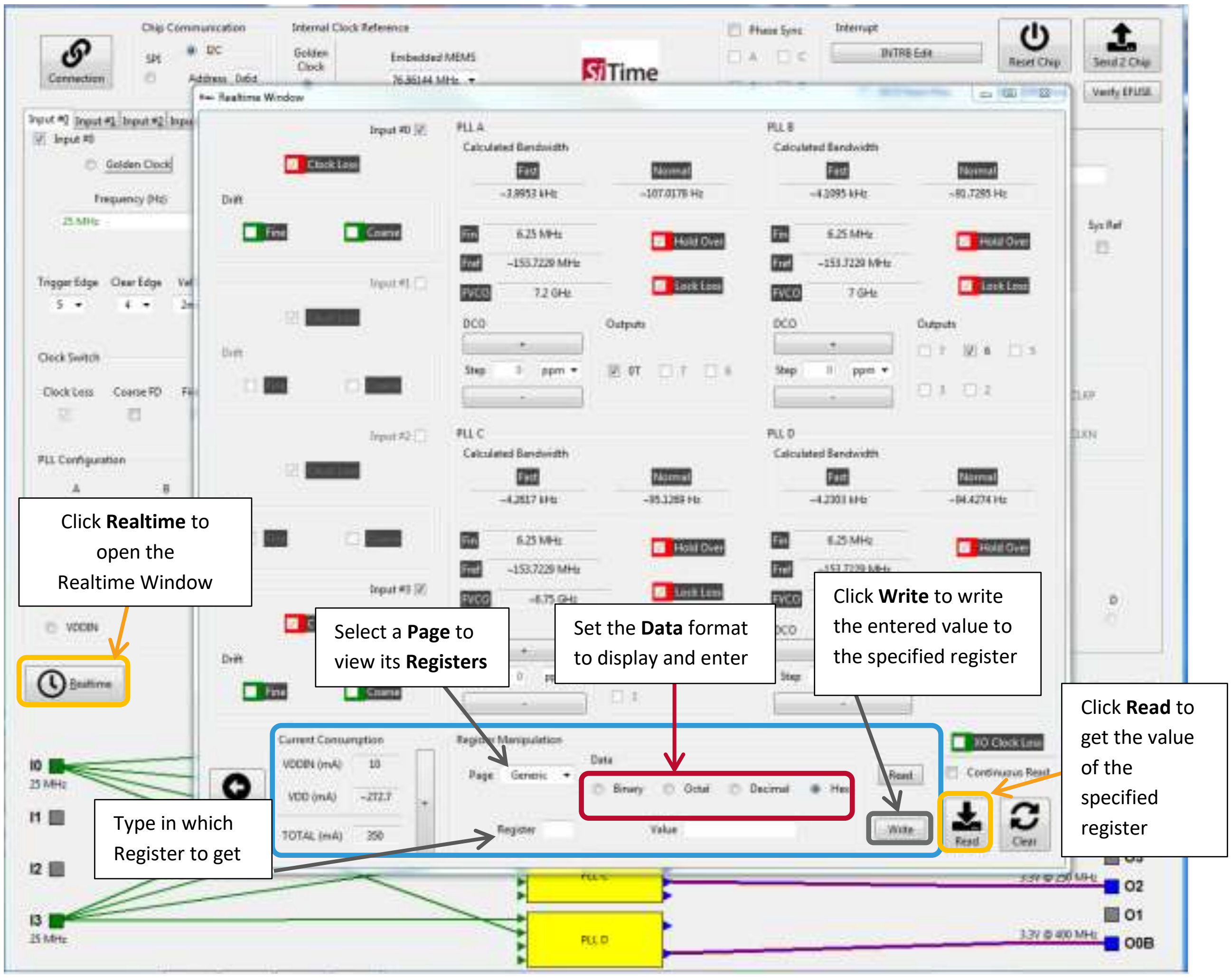

Click **Realtime** to open the Realtime Window
Select a **Page** to view its **Registers**
Set the **Data** format to display and enter
Click **Write** to write the entered value to the specified register
Click **Read** to get the value of the specified register
Type in which Register to get
Realtime Window
Current Consumption
Register Manipulation
Page
Generic
Binary
Octal
Decimal
Hex
Register
Value
Read
Write
Clear
Connection
Reset Chip
Send 2 Chip
Verify EFUSE
PLL A
PLL B
PLL C
PLL D

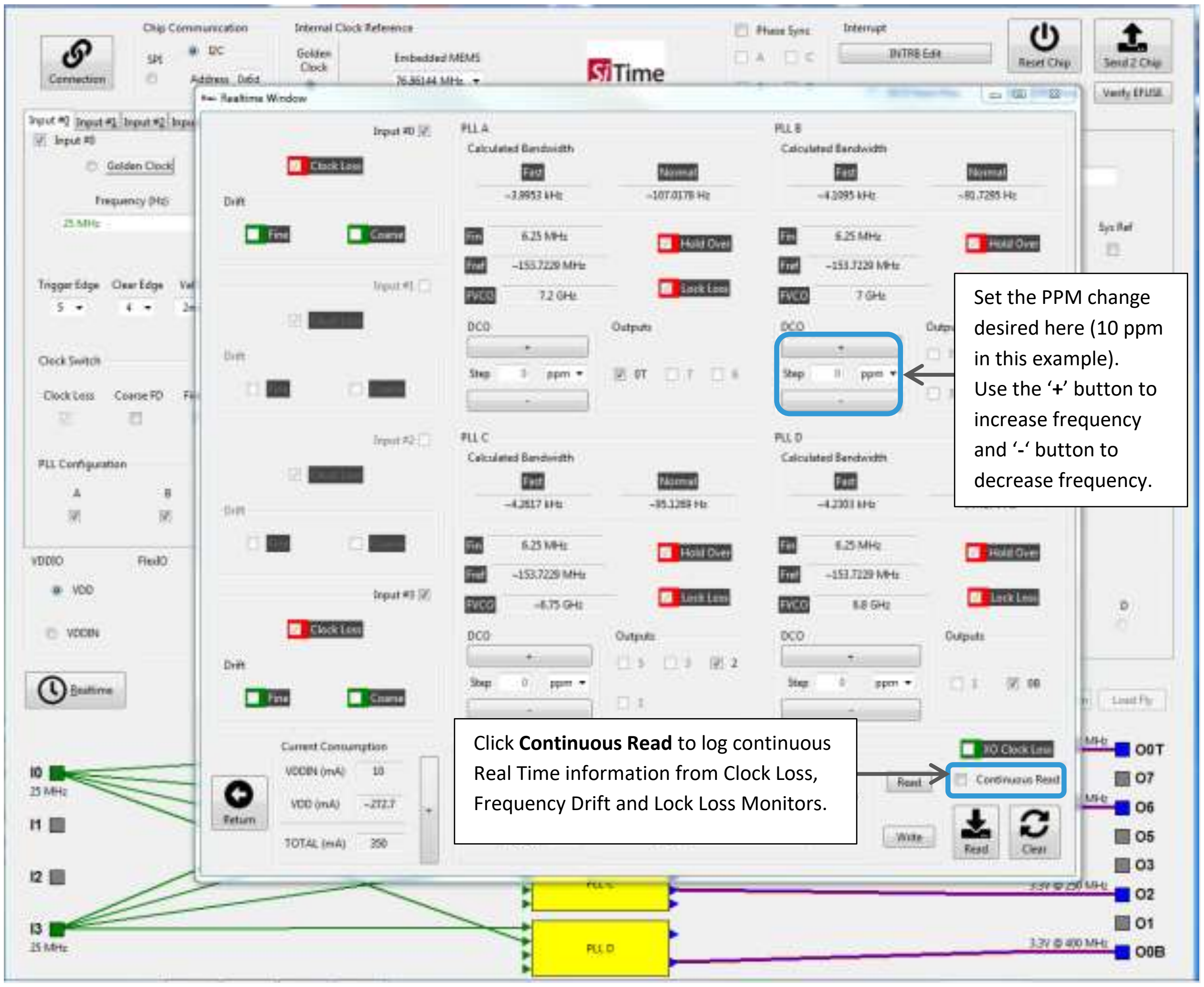
Figure 51: Descriptions of the Realtime section

When the **Return** button is clicked, the Realtime Window closes, and focus returns to the main window.

The **Register Manipulation** sub-section allows the user to read or write to a specified register on the selected page, see Figure 51.

7.5 TASK 4: Use Realtime to set output frequency with DCO or view clock monitor status

After the chip is programmed, the **Realtime** view can be used to change the output frequency using the DCO feature, or to check the status of all clock monitors, see Figure 52.



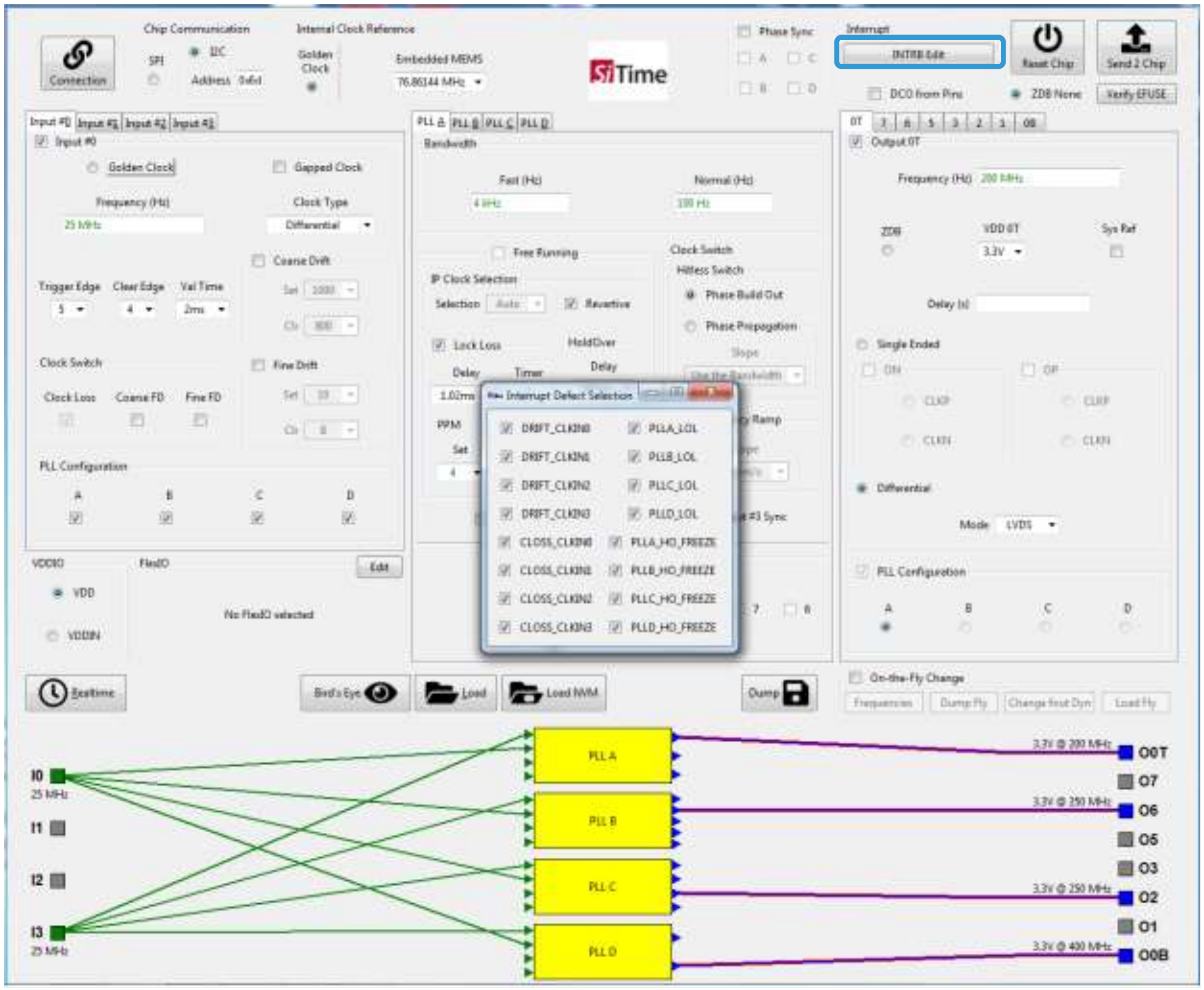
Set the PPM change desired here (10 ppm in this example).
Use the '+' button to increase frequency and '-' button to decrease frequency.

Click **Continuous Read** to log continuous Real Time information from Clock Loss, Frequency Drift and Lock Loss Monitors.

Figure 52: Viewing the Realtime section after the chip is programmed

7.5.1 Interrupts

Click the **INTRB EDIT** button to open the **Interrupt Defect Selection** panel to enable or disable sticky notifications on the **INTRB** pin, see Figure 53.



Interrupt Defect Selection

✓ DRIFT_CLKIN0	✓ PLLA_LOL
✓ DRIFT_CLKIN1	✓ PLLB_LOL
✓ DRIFT_CLKIN2	✓ PLLC_LOL
✓ DRIFT_CLKIN3	✓ PLLD_LOL
✓ CLOSS_CLKIN0	✓ PLLA_HO_FREEZE
✓ CLOSS_CLKIN1	✓ PLLB_HO_FREEZE
✓ CLOSS_CLKIN2	✓ PLLC_HO_FREEZE
✓ CLOSS_CLKIN3	✓ PLLD_HO_FREEZE

Figure 53: Selecting the interrupt defects to enable sticky notifications on the INTRB pin

Click the **Clear** button to clear all sticky notifications on the chip, see Figure 54 and clear the INTRB pin (whose state should now be '1', assuming no defect exists now among the selected list).

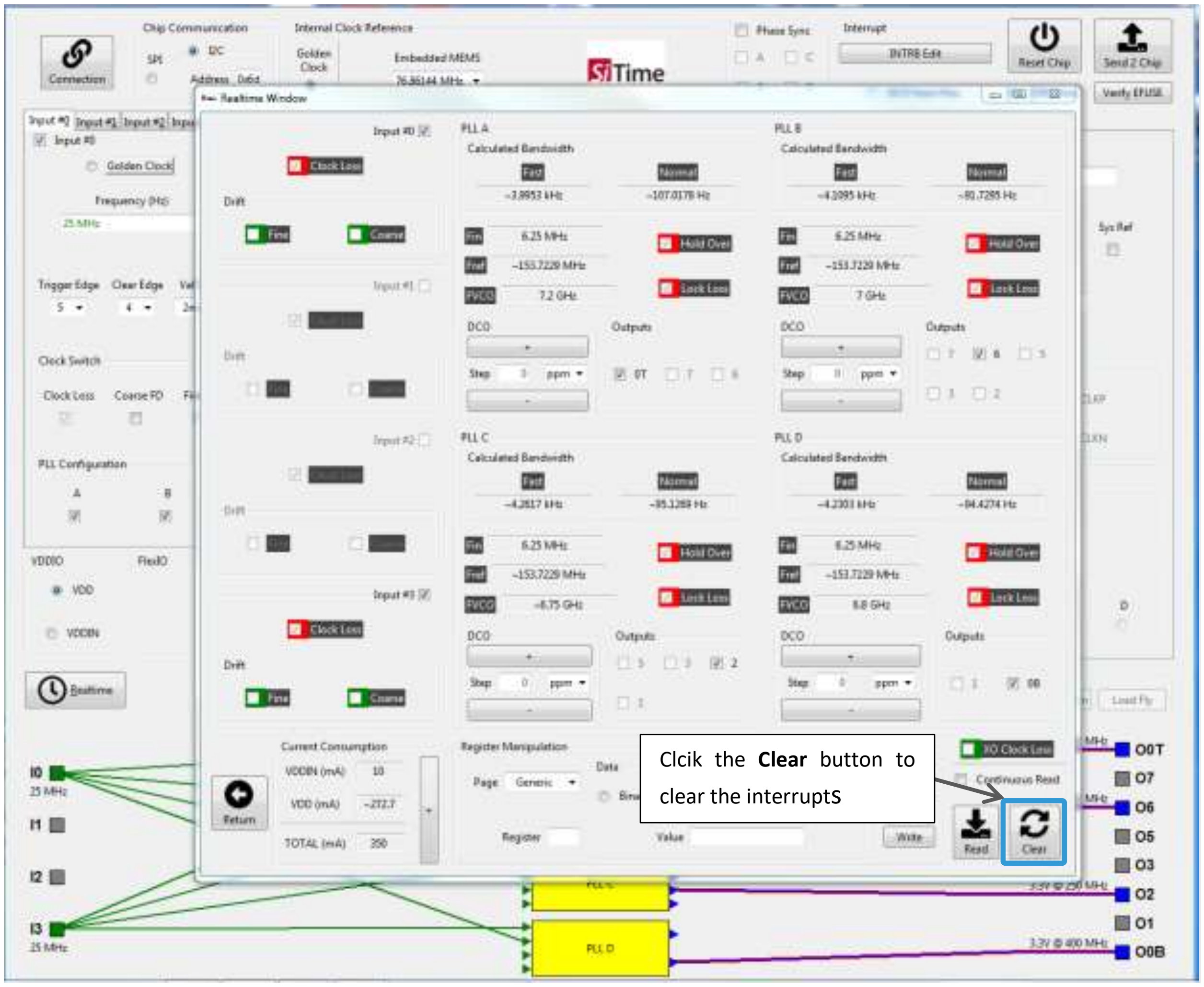


Figure 54: Using the Clear button

7.5.2 On-the-fly change

Select the **On-the-fly Change** checkbox to enable the on-the-fly change feature allowing the user to enter frequencies.

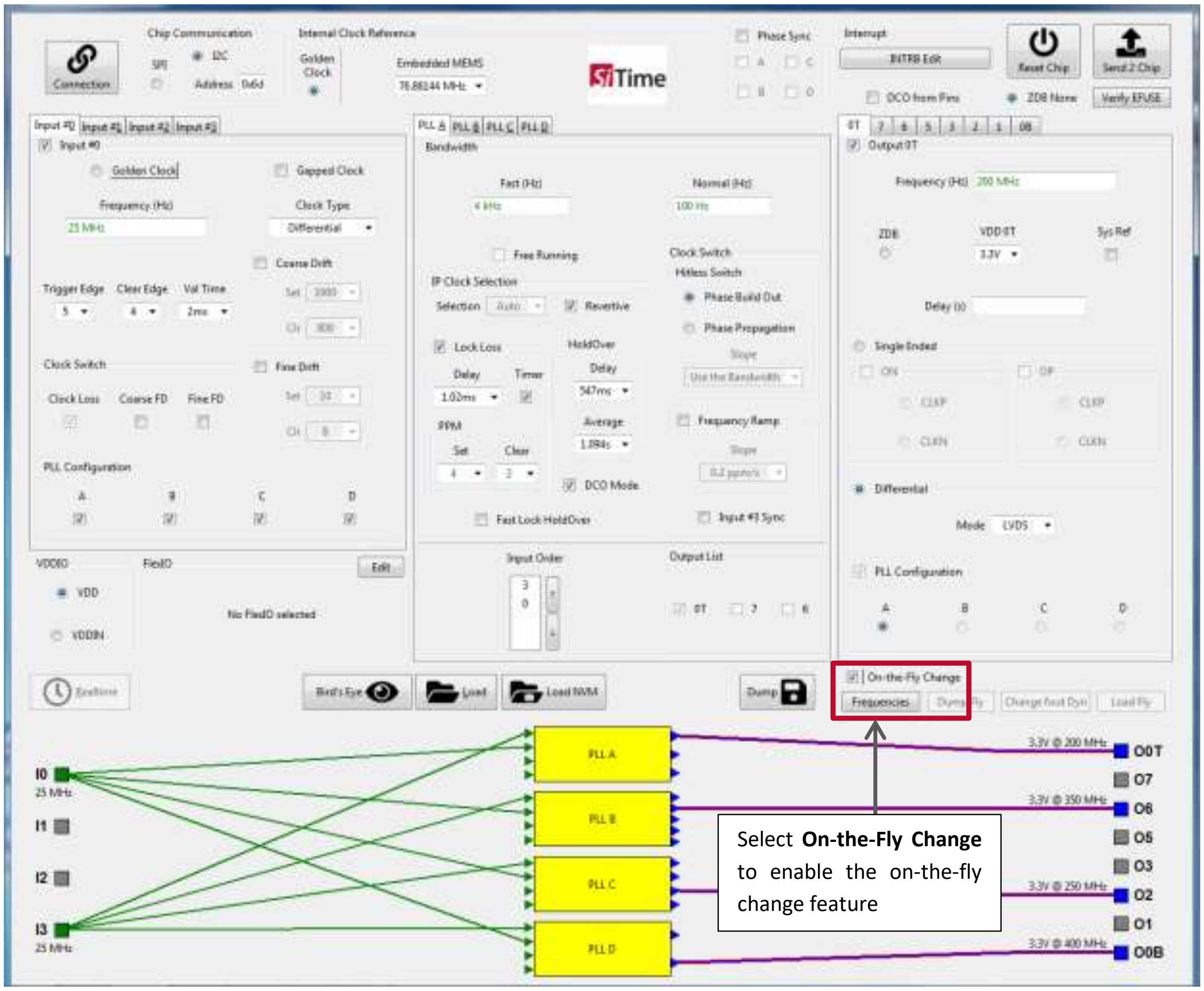


Figure 55: Enabling on-the-fly change feature

7.5.3 Managing on-the-fly frequencies

Click the **On-the-Fly** section **Frequencies** button to open the panel where you can enter and manage the on-the-fly frequencies. Use the two buttons at the top to clear one or all selected frequencies.

Near the bottom of the panel is the **New (Hz)** field where you can enter a new frequency and then click the **+** button (right of the field) to add the new frequency to the list, see Figure 56.

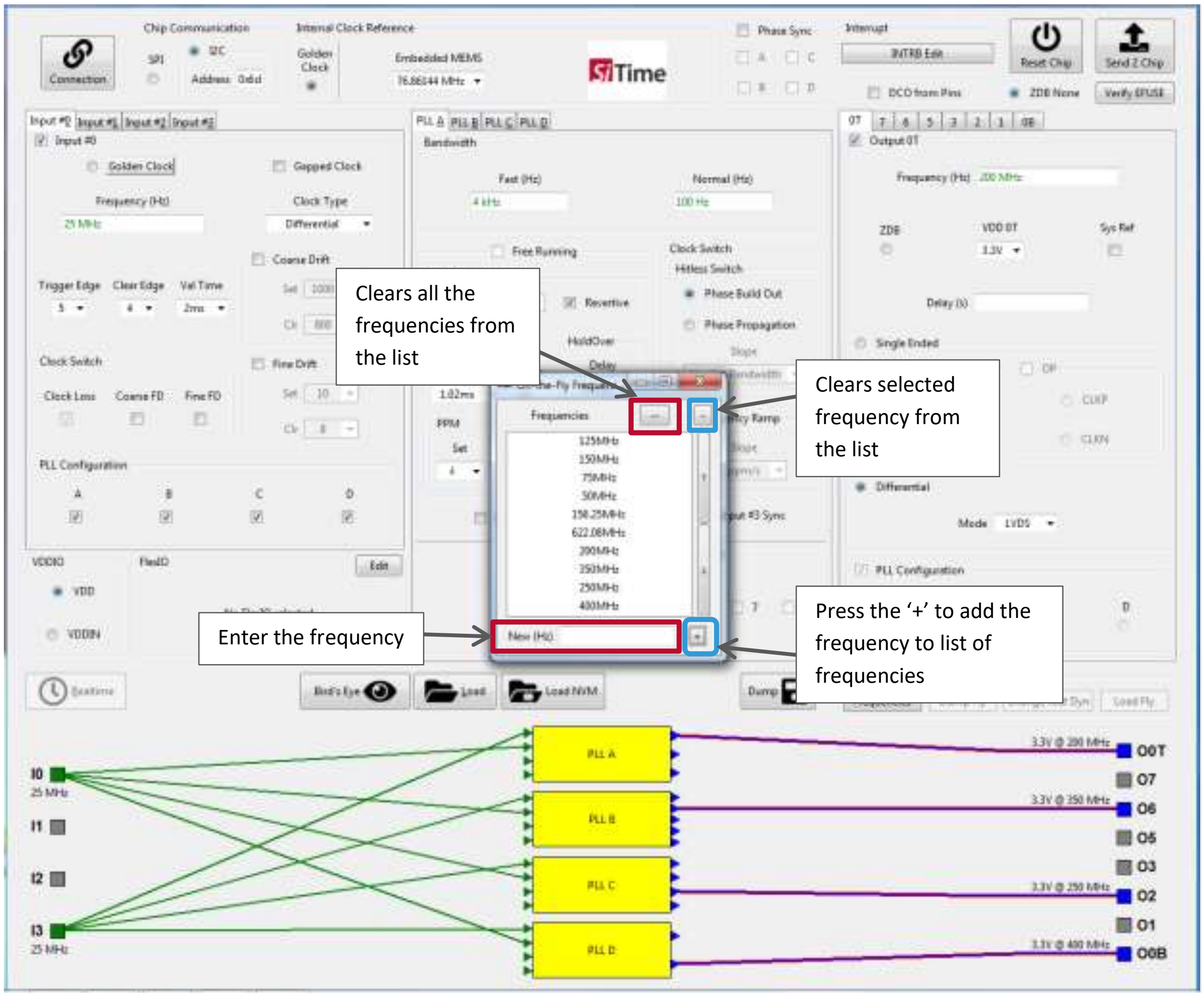


Figure 56: Using the On-the-Fly Frequency pop up

After the chip is programmed, the dump fly and load fly functions are enabled.

Note, if the connection to the chip is not active, you will *not* be able to operate this feature.

7.5.4 Using the dump fly function

The **Dump Fly** function allows the user to dump the NVM (I2C write) file, see Figure 57.

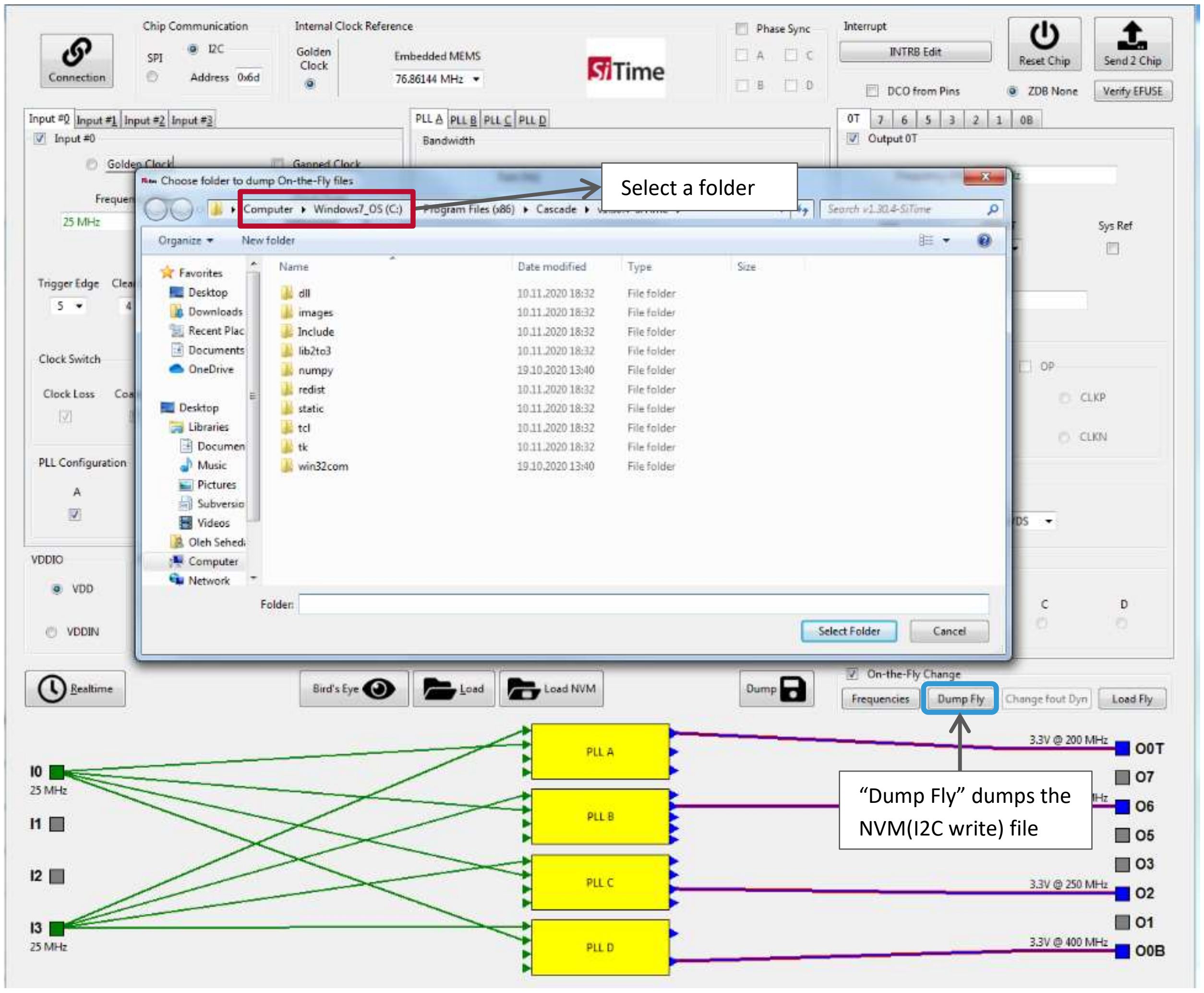

Select a folder
Computer ▸ Windows7_OS (C:) ▸ Program Files (x86) ▸ Cascade
Choose folder to dump On-the-Fly files
Organize
New folder
Select Folder
Cancel
Dump Fly
"Dump Fly" dumps the NVM(I2C write) file
PLL A
PLL B
PLL C
PLL D
I0
I1
I2
I3
O0T
O7
O6
O5
O3
O2
O1
O0B

Figure 57: Click the Dump Fly button to save the NVM(I2C write) file

7.5.4.1 Static profile on-the-fly with single output per PLL

When the **Dump Fly** button is clicked, it prompts the user to dump the NVM (I2C write) file, see Figure 57.

Note, the number of files created are 4*N (N- number of frequencies entered). The created file will have fixed file name.

Example: PllX_pin_fout_otf_NVM.py

```
X    - A, B, C, D
pin  - Output name (Example: if OUT6 it will be displayed as 6)
fout - Frequency that are entered in the Frequency box
If the frequency entered was "125MHz", then the file created will be follows:
Single output       : PllA_T_125000000p0_otf_NVM.py
```

7.5.4.2 Selecting static profile on-the-fly with multiple outputs with same frequency

There is no special change if multiple outputs are needed from the same PLL at the same frequency. However, the outputs need to be selected and defined appropriately, see Figure 58.

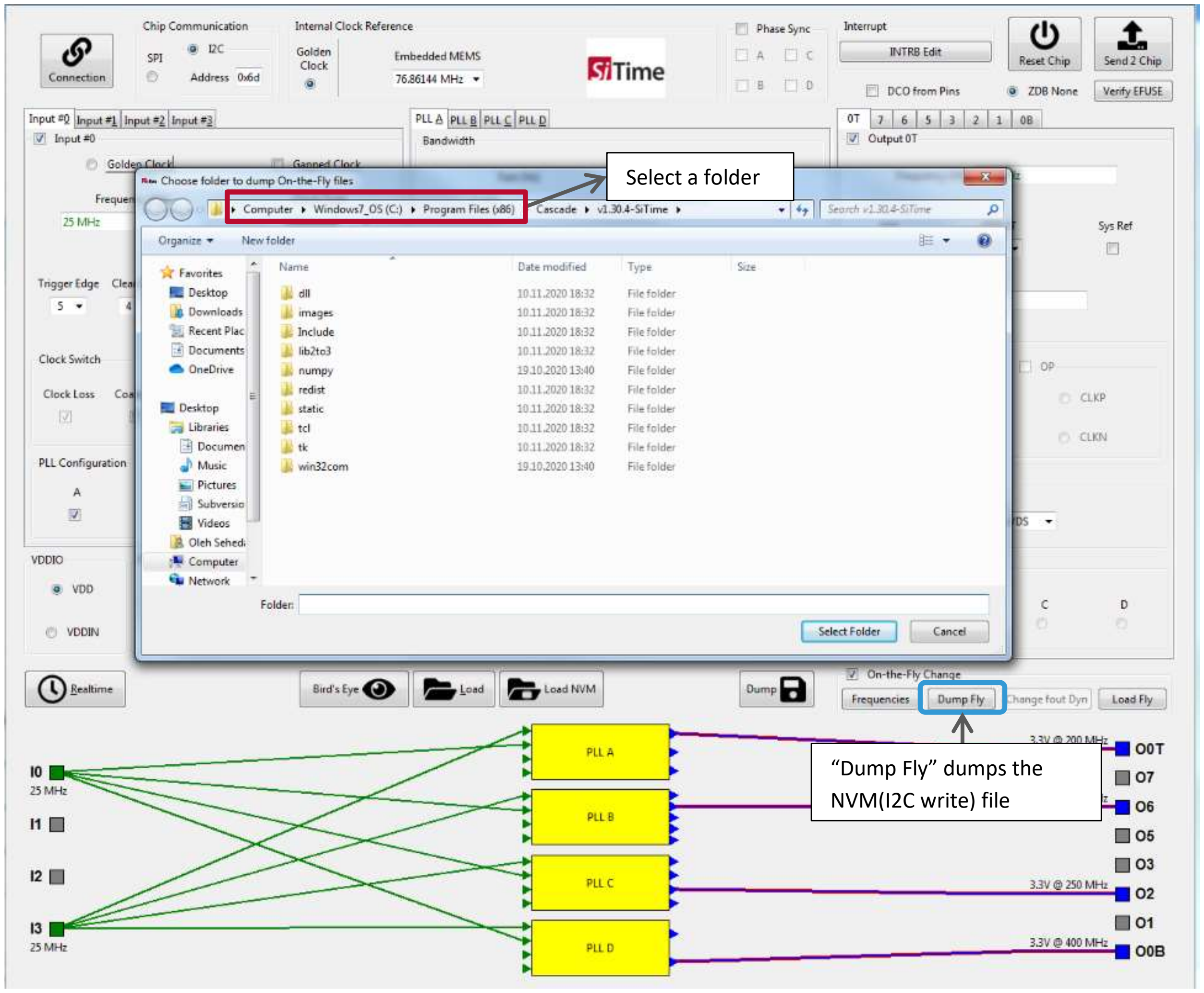

Select a folder
"Dump Fly" dumps the NVM(I2C write) file
Choose folder to dump On-the-Fly files
Computer ▸ Windows7_OS (C:) ▸ Program Files (x86) ▸ Cascade ▸ v1.30.4-SiTime
Select Folder
Cancel
Dump Fly
PLL A
PLL B
PLL C
PLL D
I0 25 MHz
I1
I2
I3 25 MHz
O0T
O7
O6
O5
O3
O2
O1
O0B

Figure 58: Selecting static profile on-the-fly with multiple same frequency outputs

The output files will look different from the single output files as shown below.

Example: PllX_pin1_fout_pin2_fout_...otf_NVM.py

```
X     - A,B,C,D
pin1/2…- output names (Example: if OUT6 it will be displayed as 6)
fout - frequency that are entered in the Frequency box
```

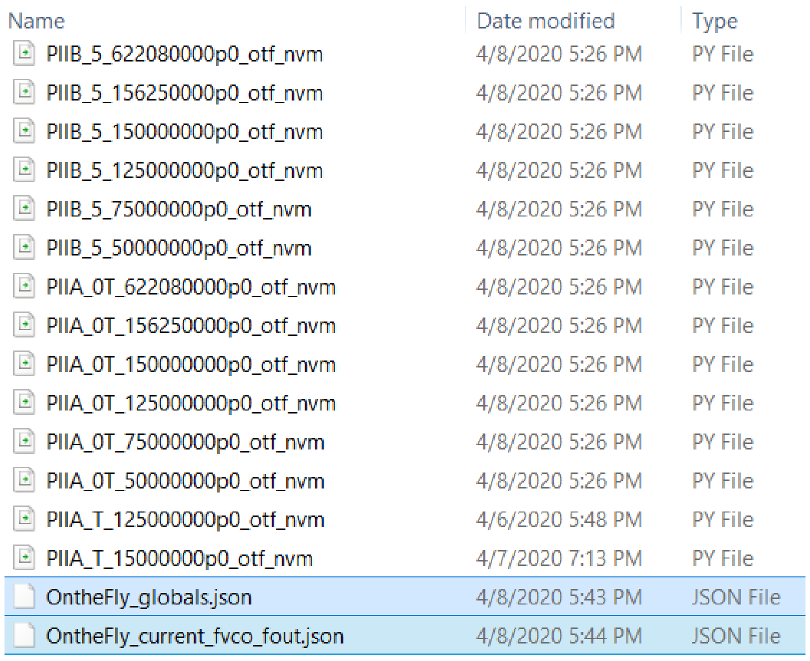
If the frequency entered was 125 MHz, then the file created will be like the following example.

```
Single output: PllA_T_125000000p0_otf_NVM
```

7.5.5 Dynamic profile on-the-fly with single output per PLL

Two additional files will be created to support the dynamic embedded algorithm for frequency change on-the-fly, see Figure 59:

- OntheFly_current_fvco_fout.json
- OntheFly_globals.json



Name	Date modified	Type
PllB_5_622080000p0_otf_nvm	4/8/2020 5:26 PM	PY File
PllB_5_156250000p0_otf_nvm	4/8/2020 5:26 PM	PY File
PllB_5_150000000p0_otf_nvm	4/8/2020 5:26 PM	PY File
PllB_5_125000000p0_otf_nvm	4/8/2020 5:26 PM	PY File
PllB_5_75000000p0_otf_nvm	4/8/2020 5:26 PM	PY File
PllB_5_50000000p0_otf_nvm	4/8/2020 5:26 PM	PY File
PllA_0T_622080000p0_otf_nvm	4/8/2020 5:26 PM	PY File
PllA_0T_156250000p0_otf_nvm	4/8/2020 5:26 PM	PY File
PllA_0T_150000000p0_otf_nvm	4/8/2020 5:26 PM	PY File
PllA_0T_125000000p0_otf_nvm	4/8/2020 5:26 PM	PY File
PllA_0T_75000000p0_otf_nvm	4/8/2020 5:26 PM	PY File
PllA_0T_50000000p0_otf_nvm	4/8/2020 5:26 PM	PY File
PllA_T_125000000p0_otf_nvm	4/6/2020 5:48 PM	PY File
PllA_T_15000000p0_otf_nvm	4/7/2020 7:13 PM	PY File
OntheFly_globals.json	4/8/2020 5:43 PM	JSON File
OntheFly_current_fvco_fout.json	4/8/2020 5:44 PM	JSON File

Figure 59: Dynamic Profile on-the-fly with single output per PLL

To run the dynamic change function, a separate folder is provided with the following functions.

The src directory has the following files:

- dyn_change_out_fout.py
- flymode_freq_change_latest_multi.py

The main function is in the file named **dyn_change_out_fout.py**. If this python file is opened in an editor, at the end of the file, the **dyn_change_out_fout** function is called with three inputs, as follows:

1) DIR – Path to the directory where the following two files were created:
   - OntheFly_globals.json
   - OntheFly_current_fvco_fout.json

2) Output pin name e.g. **6** (for output O6).

3) Fout frequency *desired* as a string that matches the original specified list in the GUI, described in the following subsection.

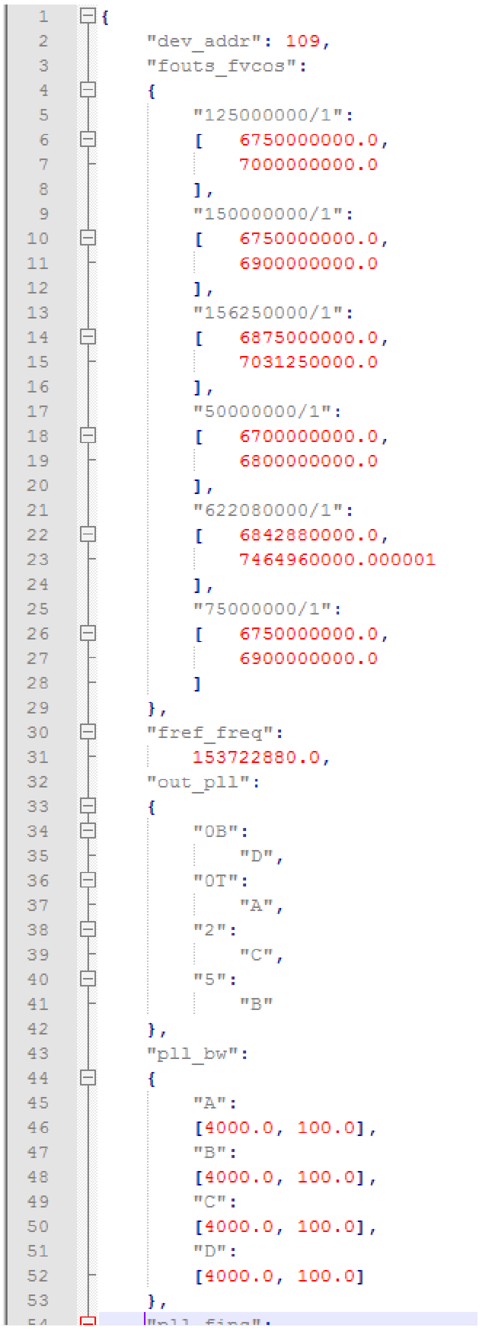
To receive the relevant frequencies, the function can be run with 0 inputs and the report with the appropriate inputs specified.

7.5.5.1 Creating a dynamic profile that uses on-the-fly frequencies

To create a dynamic profile that uses on-the-fly frequencies, first create the list of the frequencies required, and then save it (this should be done before chip programing). Then create additional files for each frequency defined in the initial list that separates files with the appropriate file names (e.g. PllA_0T_125000000p0_otf_nvm as the file name for output 0T; connected to PLLA with frequency 125 MHz).

7.5.5.2 Description of Dynamic State Files

OntheFly_globals.json: This file contains the variables set for dynamic changes. (Note, the file is formatted for easy reading, see Figure 60).

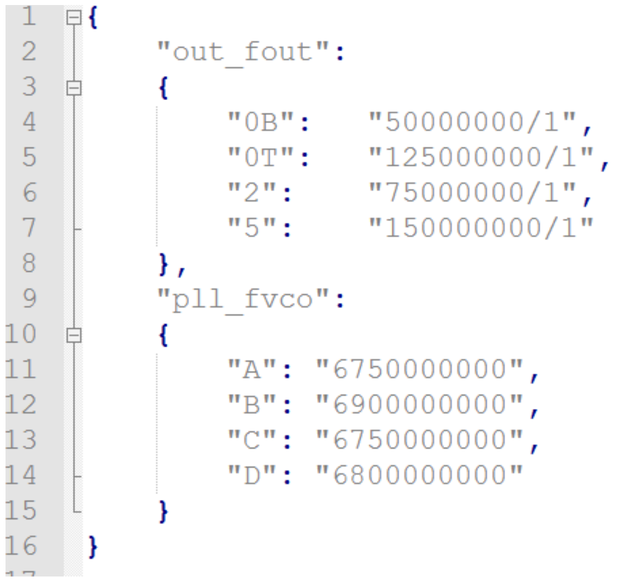


```
{
    "dev_addr": 109,
    "fouts_fvcos":
    {
        "125000000/1":
        [   6750000000.0,
            7000000000.0
        ],
        "150000000/1":
        [   6750000000.0,
            6900000000.0
        ],
        "156250000/1":
        [   6875000000.0,
            7031250000.0
        ],
        "50000000/1":
        [   6700000000.0,
            6800000000.0
        ],
        "622080000/1":
        [   6842880000.0,
            7464960000.000001
        ],
        "75000000/1":
        [   6750000000.0,
            6900000000.0
        ]
    },
    "fref_freq":
        153722880.0,
    "out_pll":
    {
        "0B":
            "D",
        "0T":
            "A",
        "2":
            "C",
        "5":
            "B"
    },
    "pll_bw":
    {
        "A":
        [4000.0, 100.0],
        "B":
        [4000.0, 100.0],
        "C":
        [4000.0, 100.0],
        "D":
        [4000.0, 100.0]
    },
```

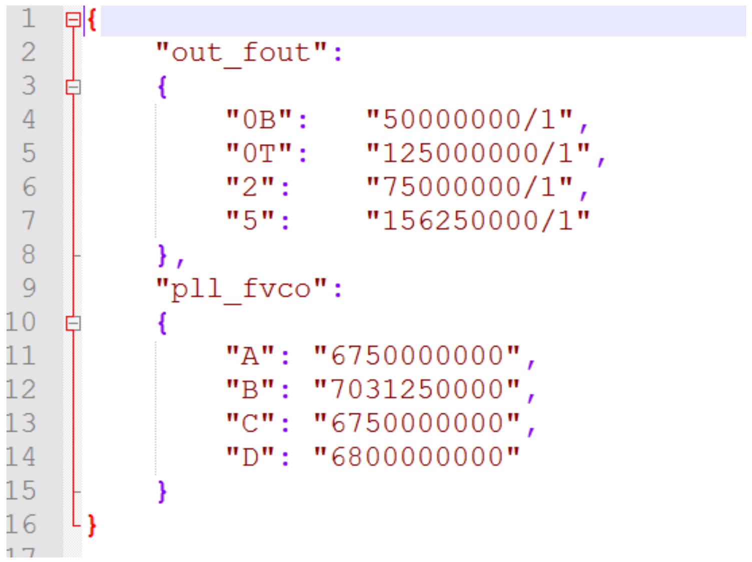
Figure 60: OntheFly_globals.json

OntheFly_current_fvco_fout.json: This file has the current FOUTs and FVCOs, which get updated when the dynamic function is run.

After Step 1, the outputs that need to be changed would have updated frequency, in this example “5” (out_fout[“5”]) that comes from PLL B being changed, and therefore is the frequency of PLL B (pll_fvco[“N”]) as set in Task 1, see Figure 61.



```
{
    "out_fout":
    {
        "0B":   "50000000/1",
        "0T":   "125000000/1",
        "2":    "75000000/1",
        "5":    "150000000/1"
    },
    "pll_fvco":
    {
        "A": "6750000000",
        "B": "6900000000",
        "C": "6750000000",
        "D": "6800000000"
    }
}
```

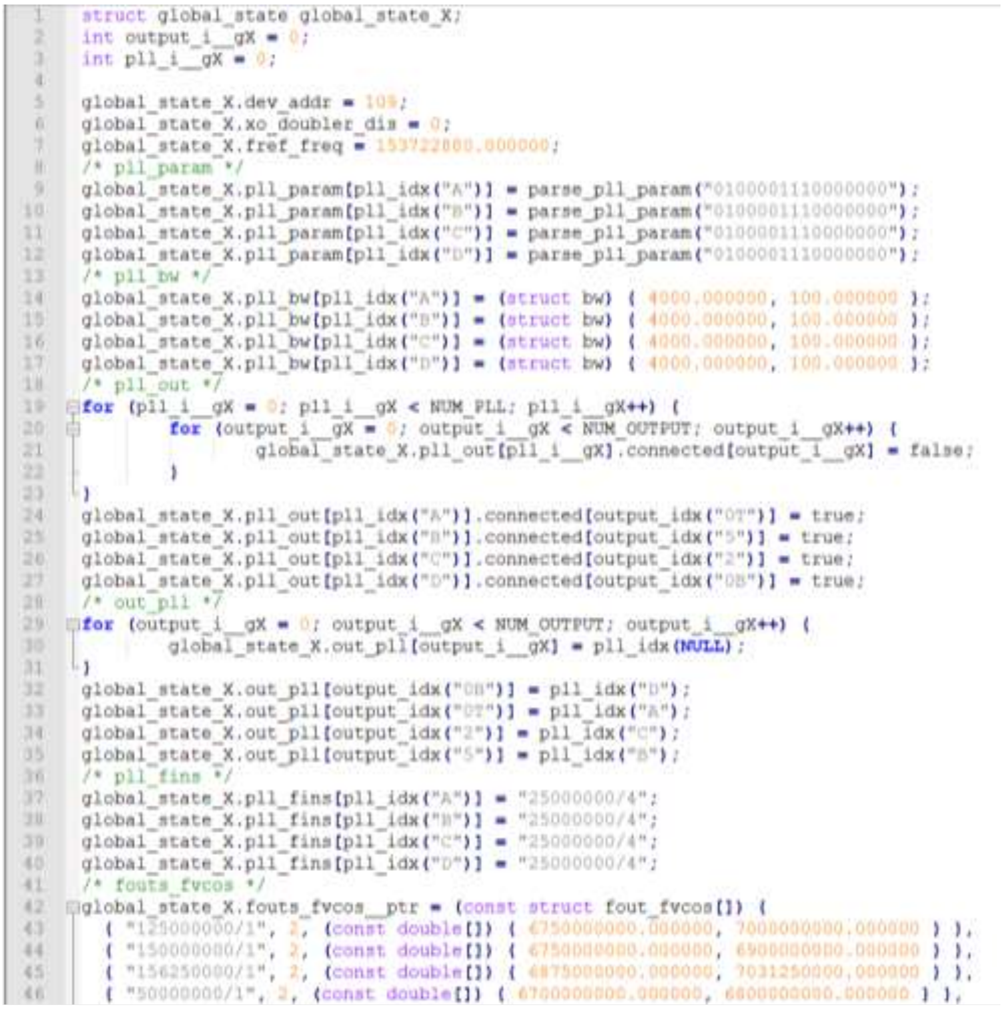


```
{
    "out_fout":
    {
        "0B":   "50000000/1",
        "0T":   "125000000/1",
        "2":    "75000000/1",
        "5":    "156250000/1"
    },
    "pll_fvco":
    {
        "A": "6750000000",
        "B": "7031250000",
        "C": "6750000000",
        "D": "6800000000"
    }
}
```

Figure 61: OntheFly_current_fvco_fout.json

7.5.6 Description of dynamic header files

OntheFly_globals.h: Has the variables set for dynamic changes, see Figure 62.



```
struct global_state global_state_X;
int output_i__gX = 0;
int pll_i__gX = 0;

global_state_X.dev_addr = 108;
global_state_X.xo_doubler_dis = 0;
global_state_X.fref_freq = 153722880.000000;
/* pll_param */
global_state_X.pll_param[pll_idx("A")] = parse_pll_param("0100001110000000");
global_state_X.pll_param[pll_idx("B")] = parse_pll_param("0100001110000000");
global_state_X.pll_param[pll_idx("C")] = parse_pll_param("0100001110000000");
global_state_X.pll_param[pll_idx("D")] = parse_pll_param("0100001110000000");
/* pll_bw */
global_state_X.pll_bw[pll_idx("A")] = (struct bw) { 4000.000000, 100.000000 };
global_state_X.pll_bw[pll_idx("B")] = (struct bw) { 4000.000000, 100.000000 };
global_state_X.pll_bw[pll_idx("C")] = (struct bw) { 4000.000000, 100.000000 };
global_state_X.pll_bw[pll_idx("D")] = (struct bw) { 4000.000000, 100.000000 };
/* pll_out */
for (pll_i__gX = 0; pll_i__gX < NUM_PLL; pll_i__gX++) {
        for (output_i__gX = 0; output_i__gX < NUM_OUTPUT; output_i__gX++) {
                global_state_X.pll_out[pll_i__gX].connected[output_i__gX] = false;
        }
}
global_state_X.pll_out[pll_idx("A")].connected[output_idx("0T")] = true;
global_state_X.pll_out[pll_idx("B")].connected[output_idx("5")] = true;
global_state_X.pll_out[pll_idx("C")].connected[output_idx("2")] = true;
global_state_X.pll_out[pll_idx("D")].connected[output_idx("0B")] = true;
/* out_pll */
for (output_i__gX = 0; output_i__gX < NUM_OUTPUT; output_i__gX++) {
        global_state_X.out_pll[output_i__gX] = pll_idx(NULL);
}
global_state_X.out_pll[output_idx("0B")] = pll_idx("D");
global_state_X.out_pll[output_idx("0T")] = pll_idx("A");
global_state_X.out_pll[output_idx("2")] = pll_idx("C");
global_state_X.out_pll[output_idx("5")] = pll_idx("B");
/* pll_fins */
global_state_X.pll_fins[pll_idx("A")] = "25000000/4";
global_state_X.pll_fins[pll_idx("B")] = "25000000/4";
global_state_X.pll_fins[pll_idx("C")] = "25000000/4";
global_state_X.pll_fins[pll_idx("D")] = "25000000/4";
/* fouts_fvcos */
global_state_X.fouts_fvcos__ptr = (const struct fout_fvcos[]) {
   { "125000000/1", 2, (const double[]) { 6750000000.000000, 7000000000.000000 } },
   { "150000000/1", 2, (const double[]) { 6750000000.000000, 6900000000.000000 } },
   { "156250000/1", 2, (const double[]) { 6875000000.000000, 7031250000.000000 } },
   { "50000000/1", 2, (const double[]) { 6700000000.000000, 6800000000.000000 } },
```

Figure 62: Description of dynamic header files

After dumping the static files in a folder, the **Change fout Dyn** button will be highlighted after dumping the static files in a folder.

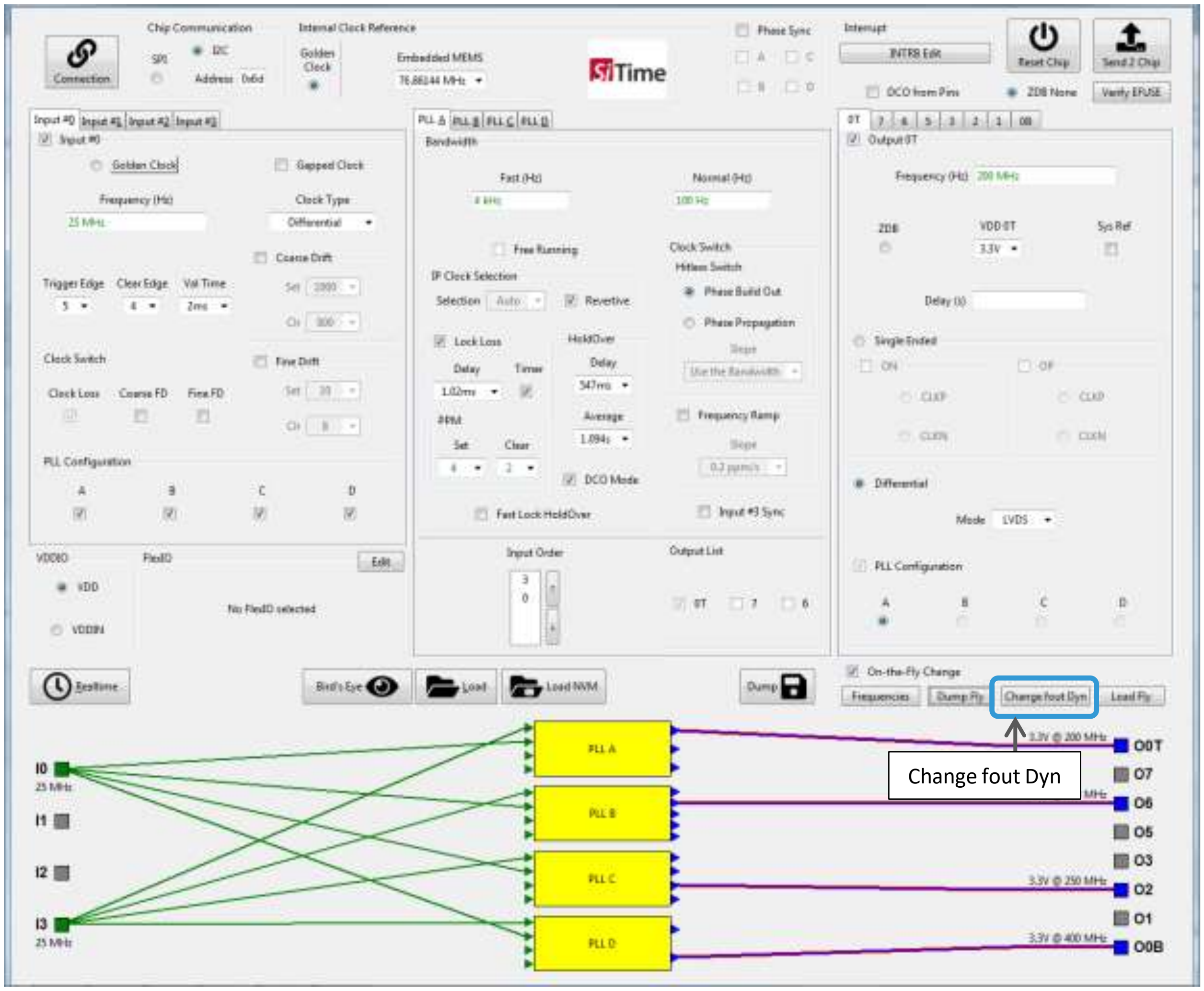


Figure 63: Using the Change fout Dyn button

When the **Change fout Dyn** button is pressed, the **On-the-fly Dynamic Frequency** window will pop-up, see Figure 64.

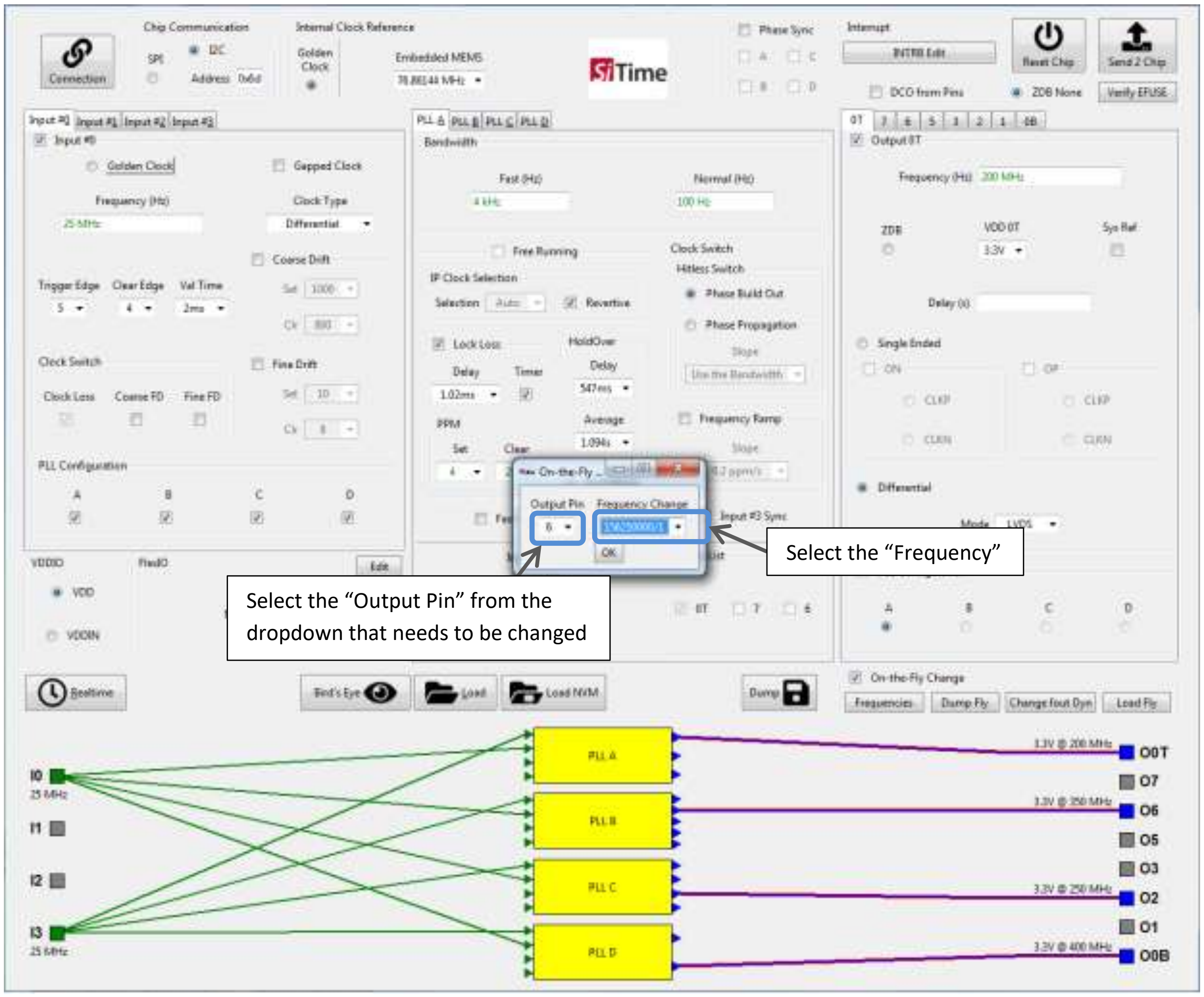


Figure 64: Specifying the On-the-fly Dynamic Frequency

After the **OK** button is pressed, the output file will be created in the exe directory.

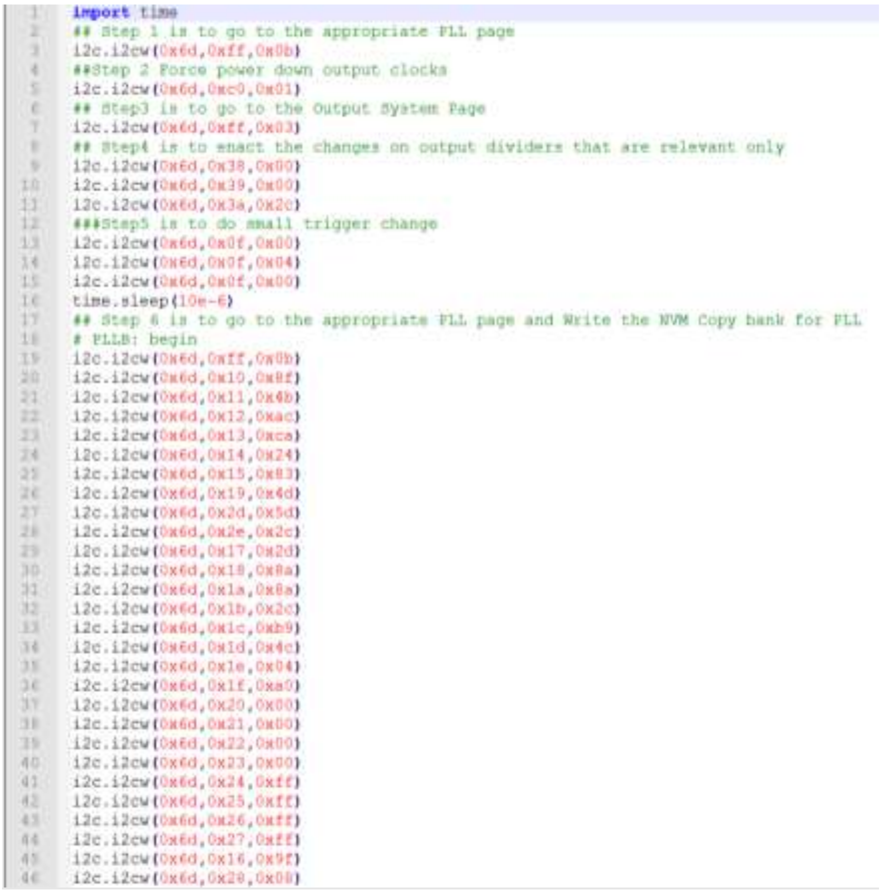
Example: fout_pin1_freq_pin2_freq_...otf_NVM.py

```
pin1/2…           -     output names (Example – if OUT0B it will displayed
as 0B)
fout -     frequency that is selected in the Frequency change dropdown.
```

If the frequency selected was 156250000/5, then the file created will be like the following:

- **fout__5_156250000.0.py**

The python output file would look like that in Figure 65:



```
import time
## Step 1 is to go to the appropriate PLL page
i2c.i2cw(0x6d,0xff,0x0b)
##Step 2 Force power down output clocks
i2c.i2cw(0x6d,0xc0,0x01)
## Step3 is to go to the Output System Page
i2c.i2cw(0x6d,0xff,0x03)
## Step4 is to enact the changes on output dividers that are relevant only
i2c.i2cw(0x6d,0x38,0x00)
i2c.i2cw(0x6d,0x39,0x00)
i2c.i2cw(0x6d,0x3a,0x2c)
###Step5 is to do small trigger change
i2c.i2cw(0x6d,0x0f,0x00)
i2c.i2cw(0x6d,0x0f,0x04)
i2c.i2cw(0x6d,0x0f,0x00)
time.sleep(10e-6)
## Step 6 is to go to the appropriate PLL page and Write the NVM Copy bank for PLL
# PLLB: begin
i2c.i2cw(0x6d,0xff,0x0b)
i2c.i2cw(0x6d,0x10,0x8f)
i2c.i2cw(0x6d,0x11,0x4b)
i2c.i2cw(0x6d,0x12,0xac)
i2c.i2cw(0x6d,0x13,0xca)
i2c.i2cw(0x6d,0x14,0x24)
i2c.i2cw(0x6d,0x15,0x83)
i2c.i2cw(0x6d,0x19,0x4d)
i2c.i2cw(0x6d,0x2d,0x5d)
i2c.i2cw(0x6d,0x2e,0x2c)
i2c.i2cw(0x6d,0x17,0x2d)
i2c.i2cw(0x6d,0x18,0x8a)
i2c.i2cw(0x6d,0x1a,0x8a)
i2c.i2cw(0x6d,0x1b,0x2c)
i2c.i2cw(0x6d,0x1c,0xb9)
i2c.i2cw(0x6d,0x1d,0x4c)
i2c.i2cw(0x6d,0x1e,0x04)
i2c.i2cw(0x6d,0x1f,0xa0)
i2c.i2cw(0x6d,0x20,0x00)
i2c.i2cw(0x6d,0x21,0x00)
i2c.i2cw(0x6d,0x22,0x00)
i2c.i2cw(0x6d,0x23,0x00)
i2c.i2cw(0x6d,0x24,0xff)
i2c.i2cw(0x6d,0x25,0xff)
i2c.i2cw(0x6d,0x26,0xff)
i2c.i2cw(0x6d,0x27,0xff)
i2c.i2cw(0x6d,0x16,0x9f)
i2c.i2cw(0x6d,0x28,0x08)
```

Figure 65: On-the-fly python output file example

7.6 Using the load fly function

When the **Load Fly** button is clicked, it prompts the user to load the NVM (I2C write) file created by the on-the-fly change feature, see Figure 66.

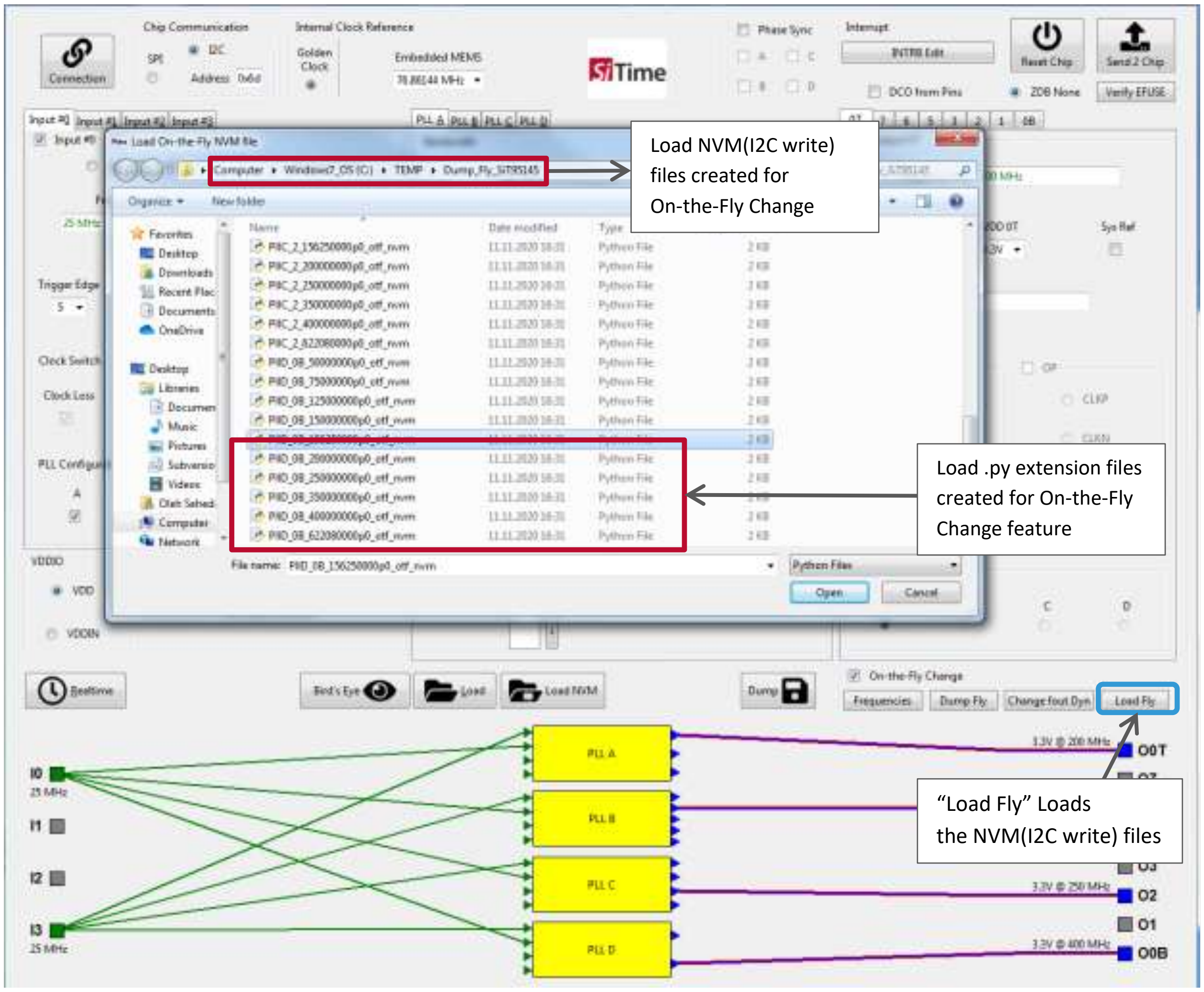

Load NVM(I2C write) files created for On-the-Fly Change
Load .py extension files created for On-the-Fly Change feature
"Load Fly" Loads the NVM(I2C write) files
Load On-the-Fly NVM file
Computer ▸ Windows7_OS (C:) ▸ TEMP ▸ Dump_Fly_SiT95145
File name: PllD_08_156250000p0_otf_nvm
Python Files
Open
Cancel
Load Fly
On-the-Fly Change
Frequencies
Dump Fly
Change fout Dyn
PLL A
PLL B
PLL C
PLL D
I0
25 MHz
I1
I2
I3
25 MHz
O0T
O2
O1
O0B

Figure 66: Using the Load Fly button

7.7 FlexIO

SiT9514x provides flexible input output pins to monitor the status of the chip. The monitoring options available on the main GUI include:

- Input clock status of defects
- Notify (sticky until cleared by user) of frequency drift (FD) in the input clocks
- Notify of the PLL defects

The **Edit** button in the **FlexIO** section will open a pop up to select these options. Five possible outputs can be assigned to any FlexION output, shown in Figure 67, as follows:

- Clock Monitoring Defect
- Clock Notify
- PLL Notify
- All Notify (both Clock Notify and PLL Notify)
- INTRB. INTRB

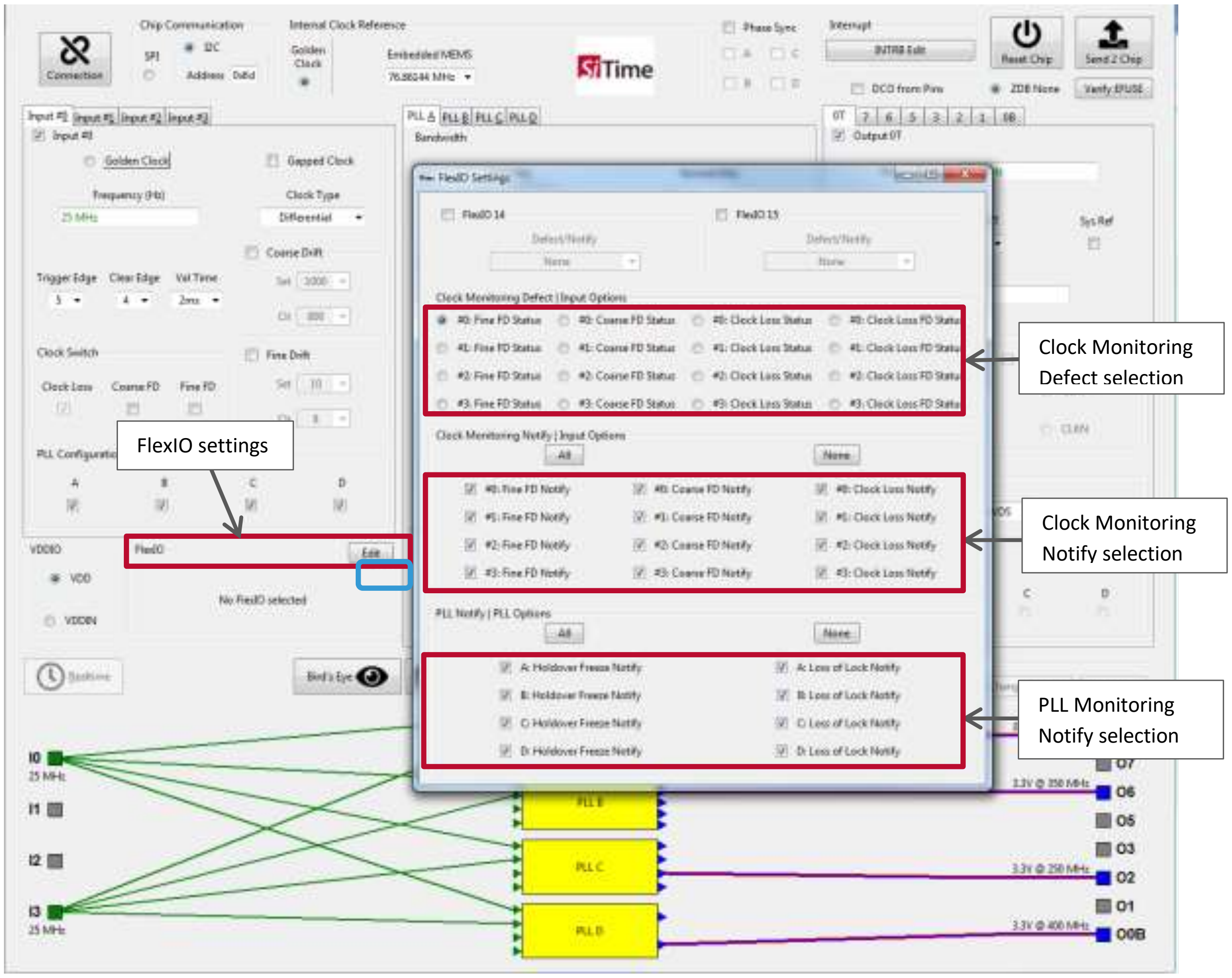
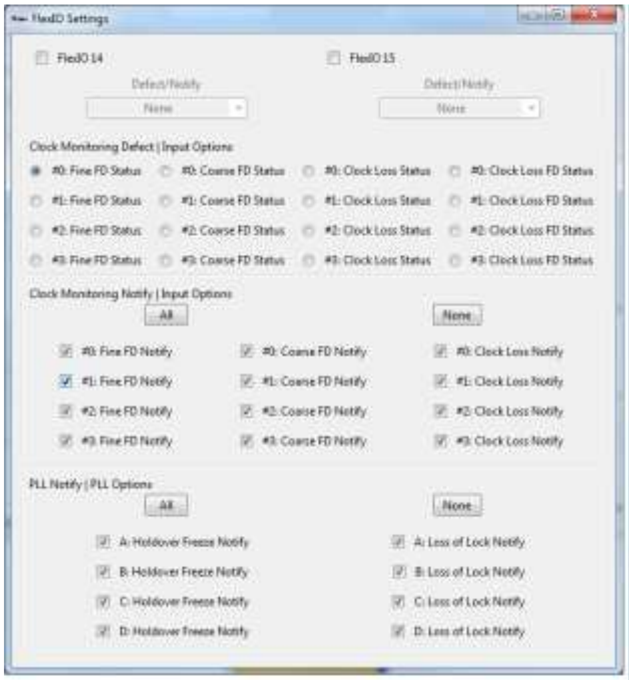


Figure 67: Using the GUI “FlexIO” widget for SiT95147

A summary of the selections made is available in the Flex IO section once Settings window is closed.

The **FlexIO Settings** panels for SiT9514x differ depending on the chip type. Figure 68 shows the **FlexIO Settings** panels for SiT95141, SiT95145, SiT95147 (panel A) and SiT95148 (panel B).



FlexIO 14 [ ] — None
FlexIO 15 [ ] — None

Clock Monitoring Defect | Input Options

[✓ #0: Fine FD Status]	#0: Coarse FD Status	#0: Clock Loss Status	#0: Clock Loss FD Status
#1: Fine FD Status	#1: Coarse FD Status	#1: Clock Loss Status	#1: Clock Loss FD Status
#2: Fine FD Status	#2: Coarse FD Status	#2: Clock Loss Status	#2: Clock Loss FD Status
#3: Fine FD Status	#3: Coarse FD Status	#3: Clock Loss Status	#3: Clock Loss FD Status

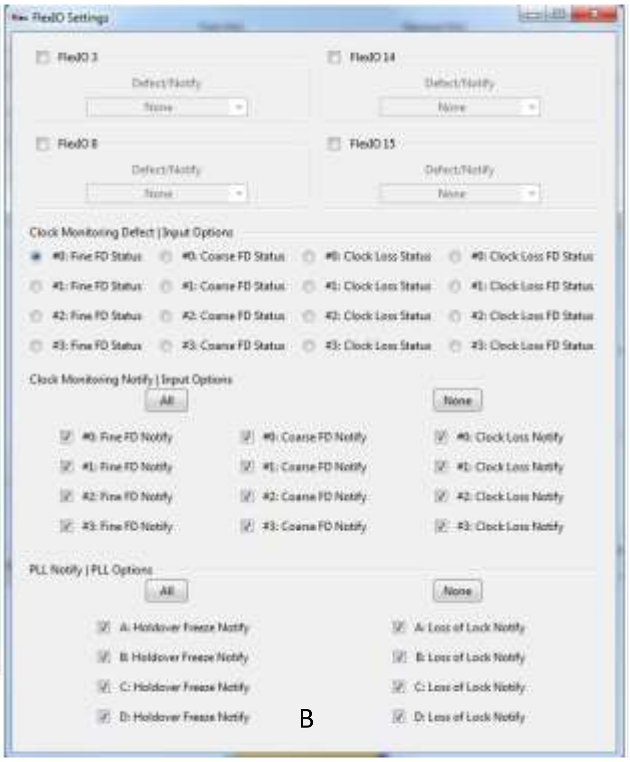
Clock Monitoring Notify | Input Options

[✓ #0: Fine FD Notify]	[✓ #0: Coarse FD Notify]	[✓ #0: Clock Loss Notify]
[✓ #1: Fine FD Notify]	[✓ #1: Coarse FD Notify]	[✓ #1: Clock Loss Notify]
[✓ #2: Fine FD Notify]	[✓ #2: Coarse FD Notify]	[✓ #2: Clock Loss Notify]
[✓ #3: Fine FD Notify]	[✓ #3: Coarse FD Notify]	[✓ #3: Clock Loss Notify]

PLL Notify | PLL Options

[✓ A: Holdover Freeze Notify]	[✓ A: Loss of Lock Notify]
[✓ B: Holdover Freeze Notify]	[✓ B: Loss of Lock Notify]
[✓ C: Holdover Freeze Notify]	[✓ C: Loss of Lock Notify]
[✓ D: Holdover Freeze Notify]	[✓ D: Loss of Lock Notify]

A



FlexIO 3 [ ] — None
FlexIO 14 [ ] — None
FlexIO 8 [ ] — None
FlexIO 15 [ ] — None

Clock Monitoring Defect | Input Options

[✓ #0: Fine FD Status]	#0: Coarse FD Status	#0: Clock Loss Status	#0: Clock Loss FD Status
#1: Fine FD Status	#1: Coarse FD Status	#1: Clock Loss Status	#1: Clock Loss FD Status
#2: Fine FD Status	#2: Coarse FD Status	#2: Clock Loss Status	#2: Clock Loss FD Status
#3: Fine FD Status	#3: Coarse FD Status	#3: Clock Loss Status	#3: Clock Loss FD Status

Clock Monitoring Notify | Input Options

[✓ #0: Fine FD Notify]	[✓ #0: Coarse FD Notify]	[✓ #0: Clock Loss Notify]
[✓ #1: Fine FD Notify]	[✓ #1: Coarse FD Notify]	[✓ #1: Clock Loss Notify]
[✓ #2: Fine FD Notify]	[✓ #2: Coarse FD Notify]	[✓ #2: Clock Loss Notify]
[✓ #3: Fine FD Notify]	[✓ #3: Coarse FD Notify]	[✓ #3: Clock Loss Notify]

PLL Notify | PLL Options

[✓ A: Holdover Freeze Notify]	[✓ A: Loss of Lock Notify]
[✓ B: Holdover Freeze Notify]	[✓ B: Loss of Lock Notify]
[✓ C: Holdover Freeze Notify]	[✓ C: Loss of Lock Notify]
[✓ D: Holdover Freeze Notify]	[✓ D: Loss of Lock Notify]

B

Figure 68: FlexIO settings differentiation depending on the device series type
***A* – FlexIO section for SiT95141, SiT95145, SiT95147; *B* – for SiT95148**

For example, for SiT95148, lets select FlexIO 3 for PLL A lock loss signal monitoring, the summary of the selections made is available in the Flex IO section, see Figure 69.

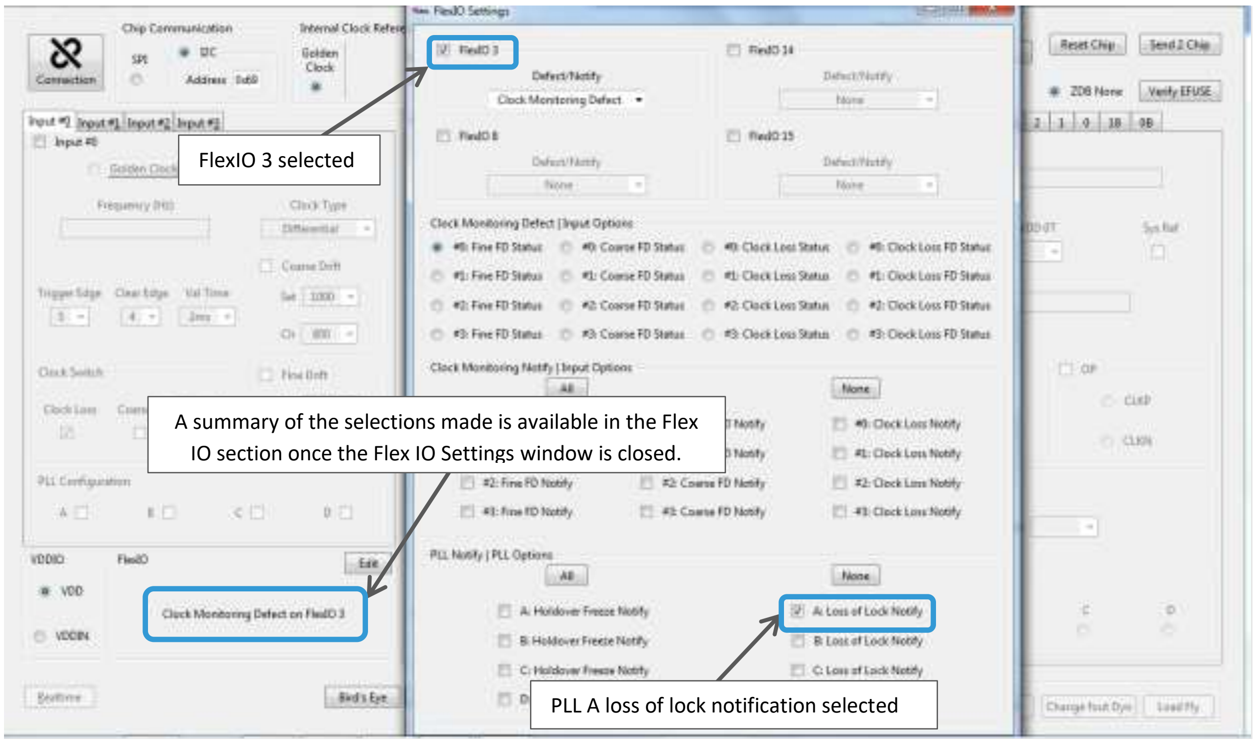

FlexIO Settings
FlexIO 3
Defect/Notify
Clock Monitoring Defect
FlexIO 14
FlexIO 8
FlexIO 15
None
Clock Monitoring Defect | Input Options
Clock Monitoring Notify | Input Options
PLL Notify | PLL Options
All
None
A: Holdover Freeze Notify
B: Holdover Freeze Notify
C: Holdover Freeze Notify
A: Loss of Lock Notify
B: Loss of Lock Notify
C: Loss of Lock Notify
Clock Monitoring Defect on FlexIO 3
FlexIO 3 selected
A summary of the selections made is available in the Flex IO section once the Flex IO Settings window is closed.
PLL A loss of lock notification selected

Figure 69: Example using the FlexIO settings for SiT95148

7.8 Phase sync feature

In this feature, the outputs from different PLLs will maintain the same relative phase difference, even in holdover when all input clocks are lost. This feature is provided by a PLL arrangement where PLLA determines the PLL dynamics and PLLB,C,D (one or more of B,C,D) are used as subordinate PLLs that work on an internal XO reference that is derived from the output of PLLA. Using this feature is recommended only for cases where the output phases are expected to stay in sync across PLLs even with loss of the input clock. For all other cases, using this is not recommended.

Figure 70 shows the PLL phase sync section in the GUI. Once the **Phase Sync** check box is clicked, you can choose the PLLs to be synced. PLLA is always set to sync because it is the internal refence PLL (even if it has no explicit output).

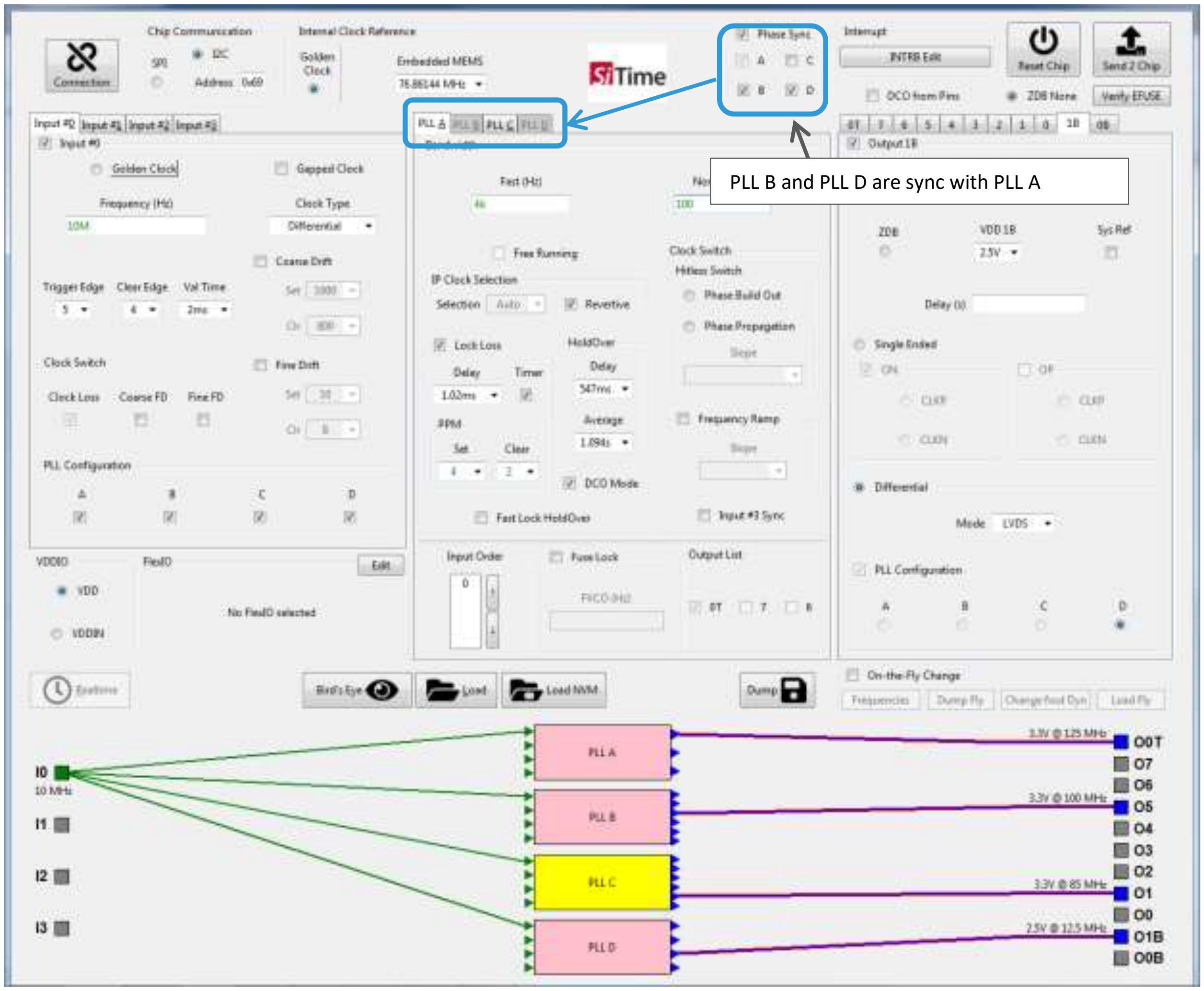


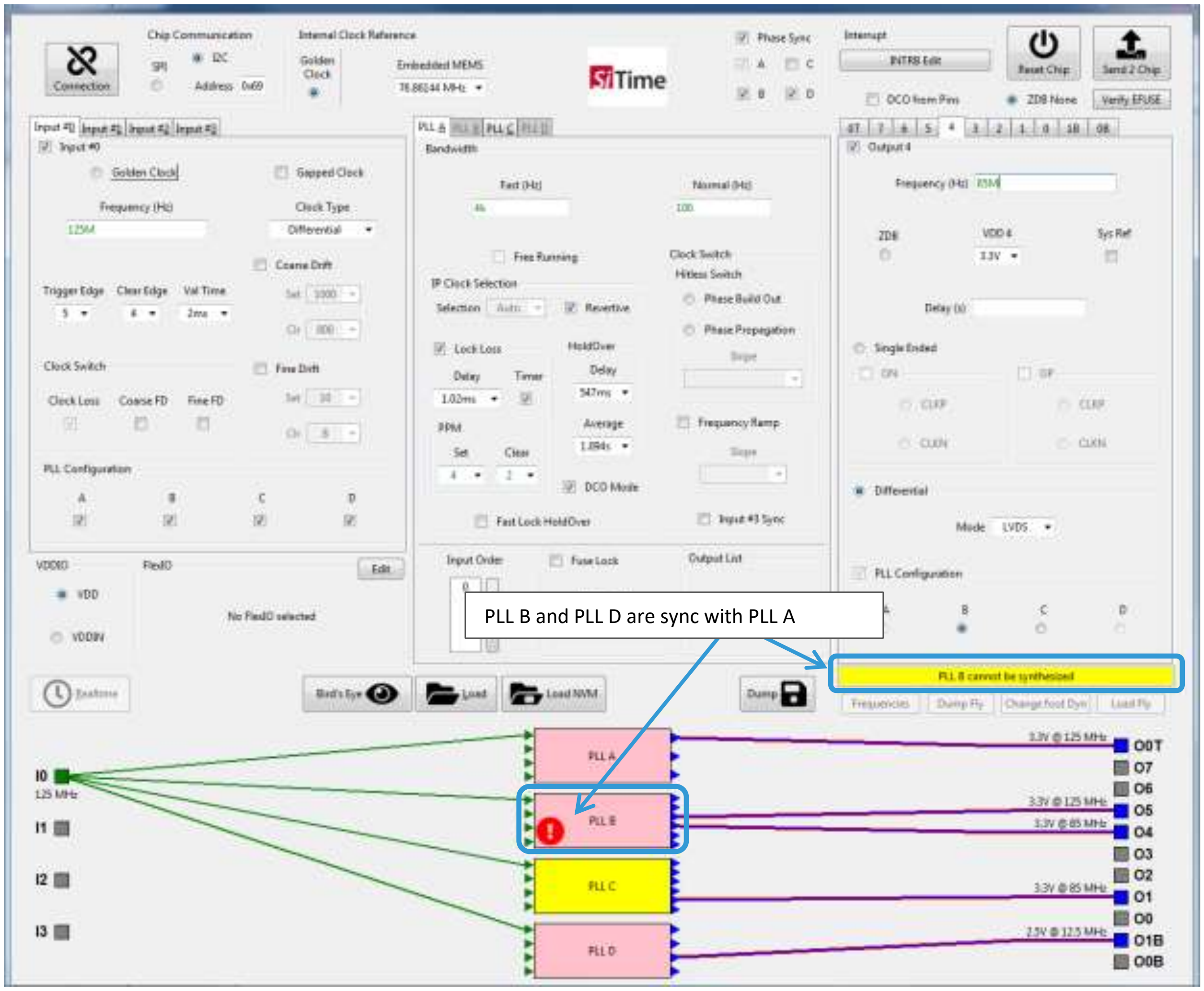
Figure 70: Example of phase sync settings

NOTE: PLLs which are enabled for phase sync will be in free run mode and hence the PLL block in the GUI is greyed out for those which have phase sync enabled.

The algorithm will generate an internal frequency that allows the PLLs to be in sync.

If the outputs are selected such that this synchronization is not possible, the GUI will report the appropriate output and the internal frequency used.

The user may change the output frequency to allow the multiples of the LCM of output frequency and the internal frequency (OCXO) to be within the frequency band [6.7 GHz, 8.4 GHz]. An example error message is shown below.



PLL B and PLL D are sync with PLL A

PLL B cannot be synthesized

Figure 71: Example of 85 MHz, 125 MHz fall (error message)

In the example shown in Figure 71, PLL B output (85 MHz, 125 MHz) is 2125 MHz, the multiples of which (6375 MHz, 8500 MHz) fall beyond the band of interest. Changing 85 MHz to 75 MHz will fix this problem.

7.9 Input to output delay control feature

SiT9514x supports a unique transient performance feature for the output clock delay with respect to the input clock. There exist two different operation modes:

- Default mode of operation:
  The output always starts with a fixed phase relationship to the input rising edge across multiple power ups of the chip, see Figure 72.
  - The start of the output clock with respect to the rising edge of the input is the same across multiple power ups with a total uncertainty of < ±175 ps.
  - The delay parameter shown above, is consistent across multiple power cycles, where: Delay= A Fixed delay ± 175 ps
- Input #3 Sync Mode of operation:
  The output always starts with a fixed phase relationship to the rising edge of an independent clock on Input #3 across multiple power ups of the chip, see Figure 73.
  - The start of the output clock with respect to the rising edge of Input #3 is the same across multiple power ups of the device with a total uncertainty of < ±175 ps.
    - **Input #3** is an independent clock not related to the PLL input.
  - The delay parameter shown above, is consistent across multiple power cycles, where: Delay = Fixed delay ± 175 ps

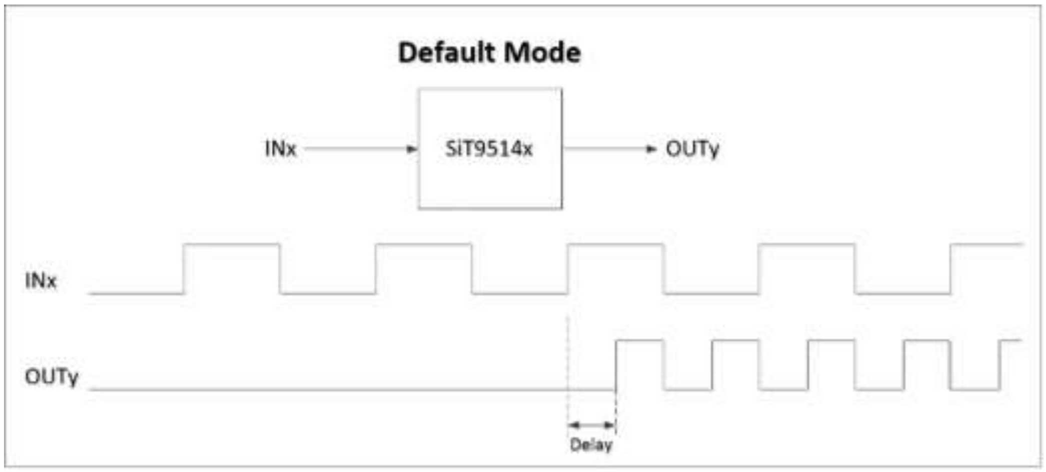


Figure 72: Input to output delay across multiple power ups in default mode

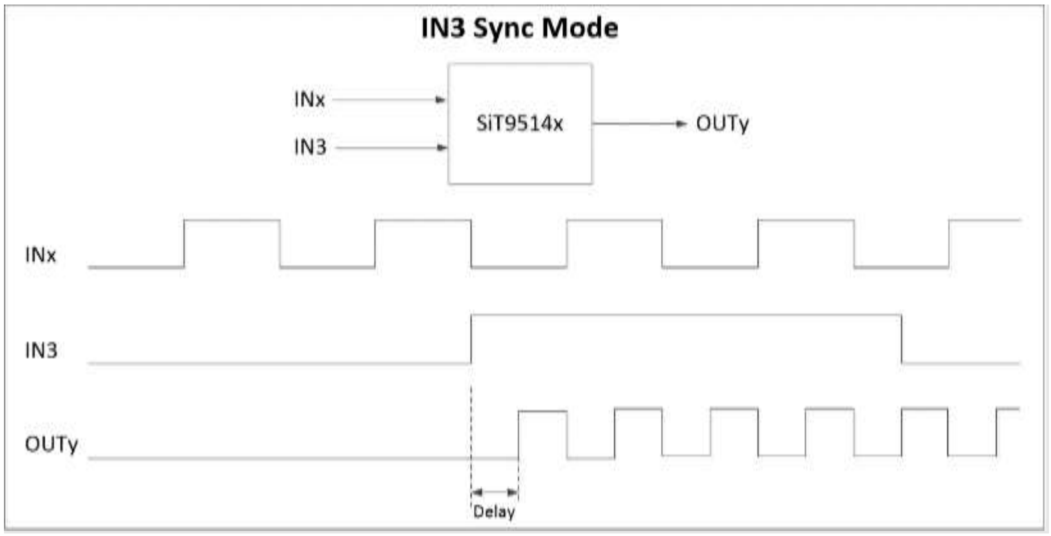


Figure 73: Input #3 to output delay across multiple power ups in sync mode

The Input #3 (IN3) SYNC is a per PLL feature and can be enabled for each PLL selectively in the GUI, see Figure 74. This can be a very useful feature, where the Input #3 clock can be used as an independent SYNC.

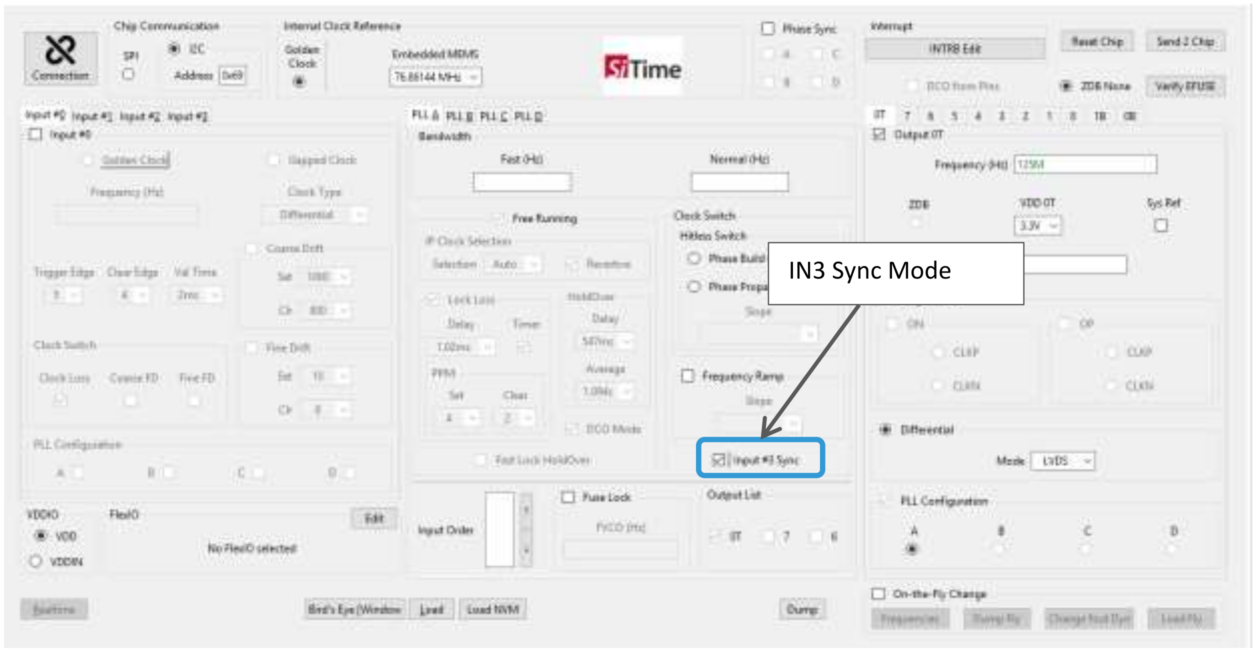


Figure 74: Example showing the selection Input #3 SYNC for PLLA of SiT95148

7.10 SiT9514x jitter attenuator as timing source for JESD204B RF converters in 5G RRU

SiT9514x family of jitter attenuators offer a highly integrated clocking solution for JESD204B compliant interfaces used in radar, servo loop control and multi-channel multi-carrier applications like 5G RRU and phased antenna array MIMO. All devices in the SiT9514x family meet the stringent timing requirements for JESD204B Subclass 0 and Subclass 1.

7.10.1 JESD204B overview

JESD204B is a JEDEC standard which defines a high-speed serial interface link between data converters and logic devices. A block diagram of a JESD204B link showing the data link and timing signals is shown in Figure 75.

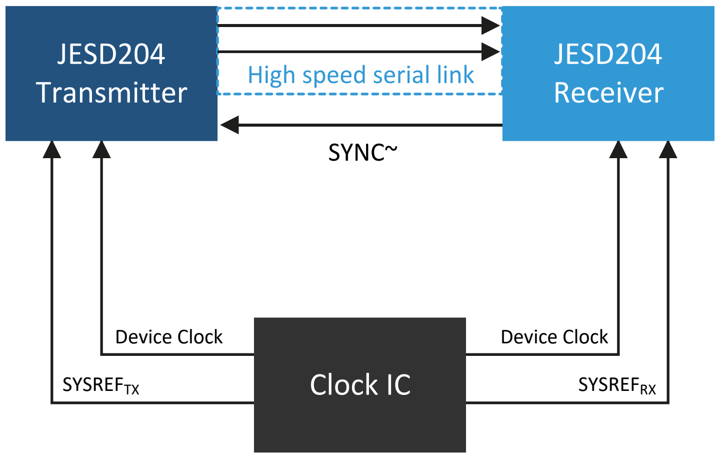


Figure 75: Block diagram of JESD204B interface between ADC (Transmitter) and FPGA (Receiver)

To achieve deterministic latency, each transmitter and receiver in the data link must be clocked by timing references with fixed phase relationships. The SiT9514x serves as the JESD204B clock source for providing these timing references. The SiT9514x distributes both a device clock (**Dev_Clock**) and a source synchronous system reference (**SYSREF**) signal to each device in the link.

7.11 Cascade as clock source for JESD204B timing signals

The SiT9514x can be configured to support the following JESD204B timing signals:

- Device Clock
- SYSREF
- SYSREF Request

A typical clock tree block diagram of the SiT9514x with JESD204 compliant converters and logic devices used in an eCPRI clocked 5G RRU is shown in Figure 76 below.

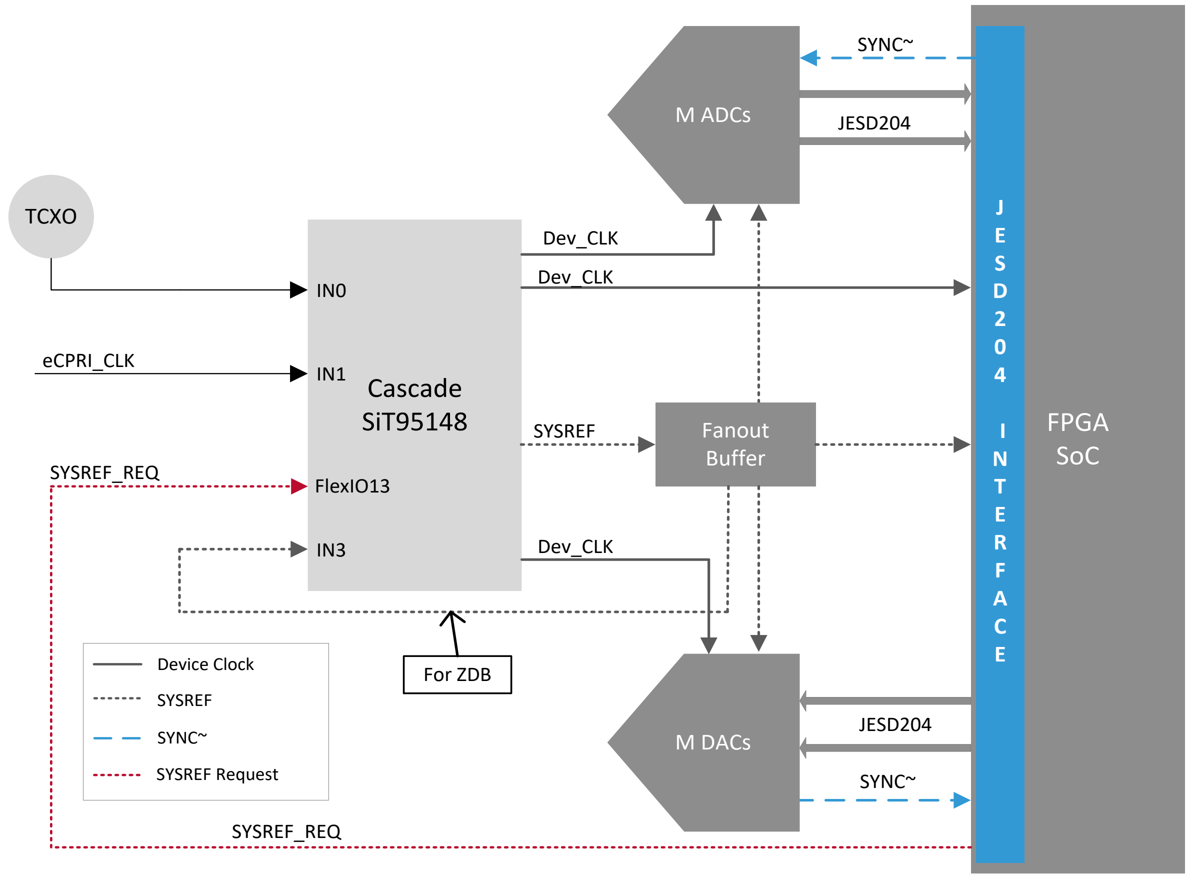


Figure 76: eCPRI clocked 5G RRU clock tree designed around SiT95148 and JESD204 compliant RF FE

In the above 5G RRU application, SiT9514x synthesizes multiple copies of the device clock (**DEV_CLK**) and a divided down phase locked **SYSREF** from one of the two master clock references: **eCPRI** recovered clock or the local **TCXO**. The **SYSREF_REQ** is the JESD204 *Request to Generate* a **SYSREF** trigger signal from the FPGA to a FLEXIO input of SiT95148, which in turn gates the **SYSREF** clock out. To understand the timing relationship of the **SYSREF** trigger (**SYSREF_REQ**) signal to the **SYSREF** clock output, let's review the SiT95148 architecture block diagram shown in Figure 77.

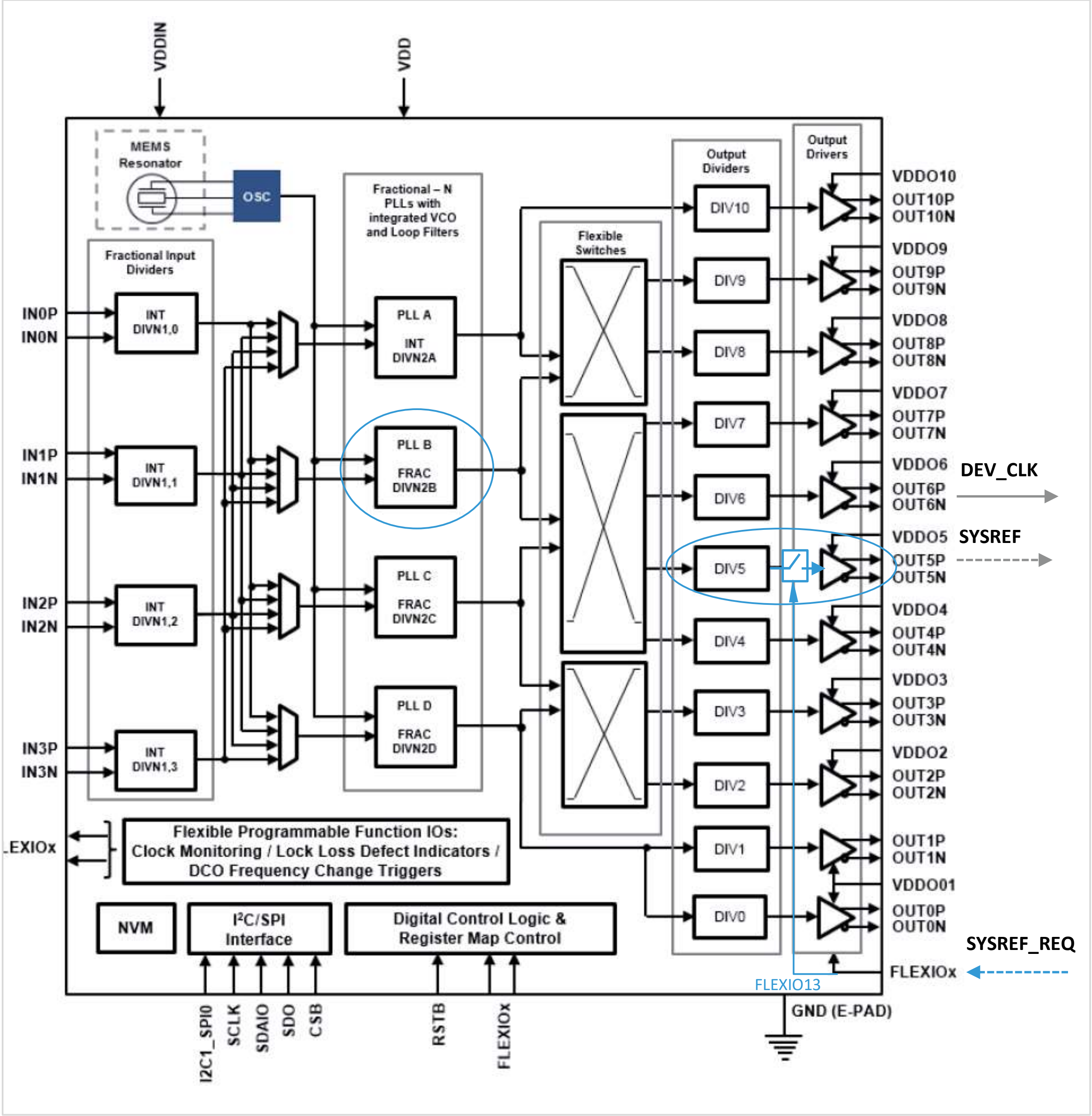


Figure 77: SiT95148 architecture block diagram correspondent for eCPRI clocked 5G RRU clock tree

As shown in the example block diagram Figure 77, **SYSREF_REQ** is driven into SiT95148 **FLEXIO13** configured as an input. In this configuration, **SYSREF_REQ** trigger serves as a gating signal for the PLLB output divider, **DIVO5**. PLLB is configured to generate **SYSREF** as a divided-down phase-locked copy of **Dev_CLK**. The **SYSREF_REQ** is used as a *gating* signal to the internally generated **SYSREF** clock – **SYREF** is *gated* into the output driver when **SYSREF_REQ** is high and *gated off* when **SYSREF_REQ** is low, thereby driving **OUT5** to a logical high. This *gating* of the internal **SYSRREF** to the output is timed on the positive edge of the **SYSREF** clock.

7.12 Configuring the SiT9514x for JESD204B timing signals

Given that most systems are designed around more than two converter devices, select either **PLL B** or **PLL C** to synthesize the required number of device clocks and one SYSREF from the system master clock. In a 5G RRU design, the master clock source is typically an eCPRI or 10 GbE recovered clock. Configure Cascade Platform SiT9514x products so that the selected PLL is in zero-delay buffer (ZDB) mode. Feed the output of the SYSREF fan-out buffer into **Input #3** configured for ZDB mode. This will ensure a repeatable and zero phase delay between the **DEV_CLK** and **SYSREF** pairs.

The following sections describe the procedure to configure SYSREF generation using two types of stimulus: hardware trigger signal: **SYSREF_REQ** on **FLEXIO13** or by writing once each to multiple registers.

7.13 Generating SYSREF via SYSREF_REQ

To generate **SYSREF** from a trigger signal like **SYSRE_REQ** on **FLEXIO13**:

1) Identify the PLL and output on which SYSREF will be driven out.
(Figure 78 shows an example of the initial configuration of SiT95148 for JESD204B.)

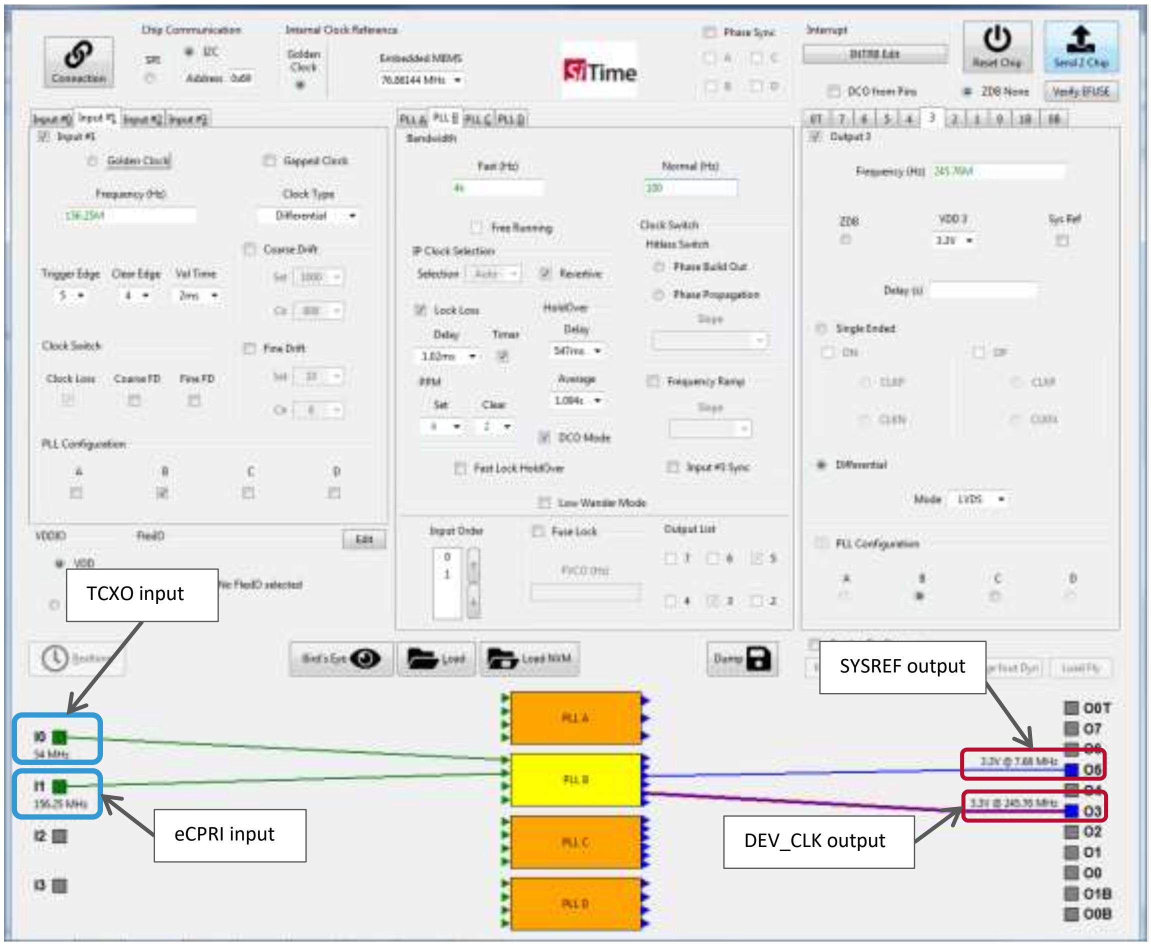


Figure 78: Example of the initial configuration of SiT95148 for JESD204B

2) Go to **Page 0**: **reg 0xFF = 0x00** or go to the **Generic** page in the **Realtime Window**, see Figure 79.

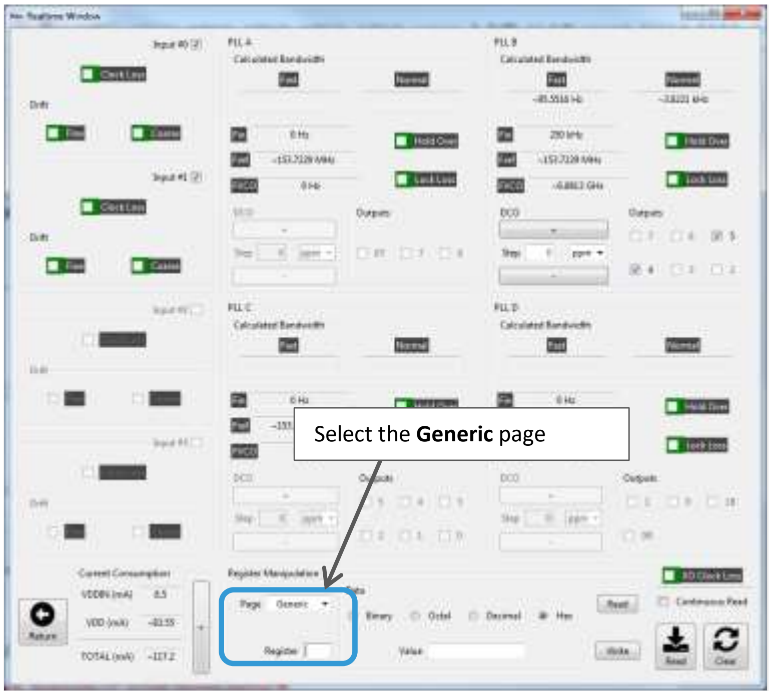


Figure 79: Selecting the Generic page in the Realtime Window's Register Manipulation section of SiT95148 for JESD204B

3) Update the value of register **0x19** shown in Table 2 (as set in the previous example, where **PLLB**: **register 0x19 = 00100010$_b$ =0x22**, see Figure 80).

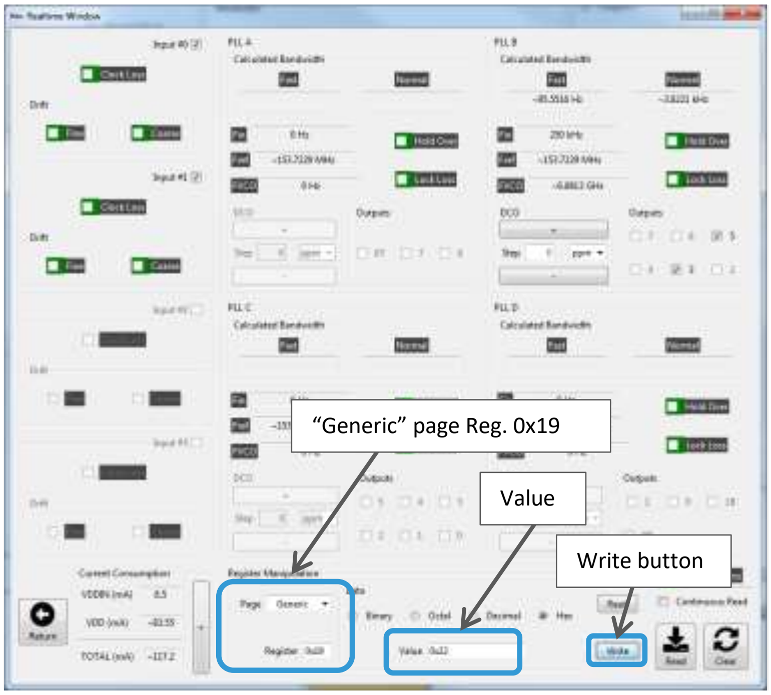


Figure 80: Writing the value to Register 0x19 in the Realtime section of SiT95148 for JESD204B

4) Do a small trigger update. An example I2C script is shown below:

```
            i2c.i2cw(0x69,0x0f,0x00)
            i2c.i2cw(0x69,0x0f,0x04)
            i2c.i2cw(0x69,0x0f,0x00)
```

5) Go to PLL page: **0xFF = 0x0B**.
(For the parameters shown in the example above for **PLLB**, see Figure 81)

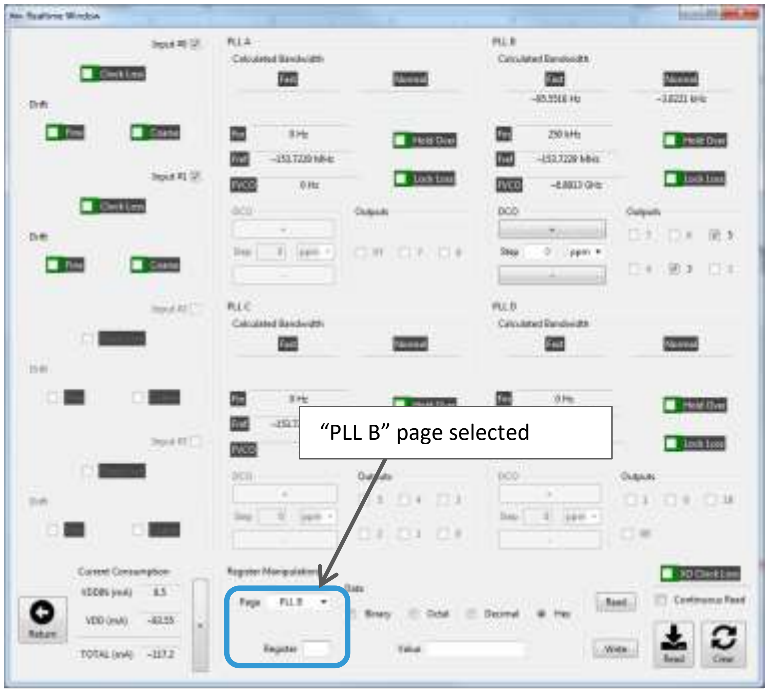


Figure 81: Selection of the PLL B page in the Realtime section of SiT95148 for JESD204B

6) Update bit s[5:0] of register **0x09** as per one hot active encoding (see Figure 82) of the **SYSREF** output. This means that SYSREF can be connected to the one of the PLL outputs and to decode it in register 0x09, we need to write appropriate value for PLL B.

- For example, if **PLLB Output 5** is the **SYSREF** output, write **0x08 (001000$_b$)** to register **0x09**, see Figure 82.

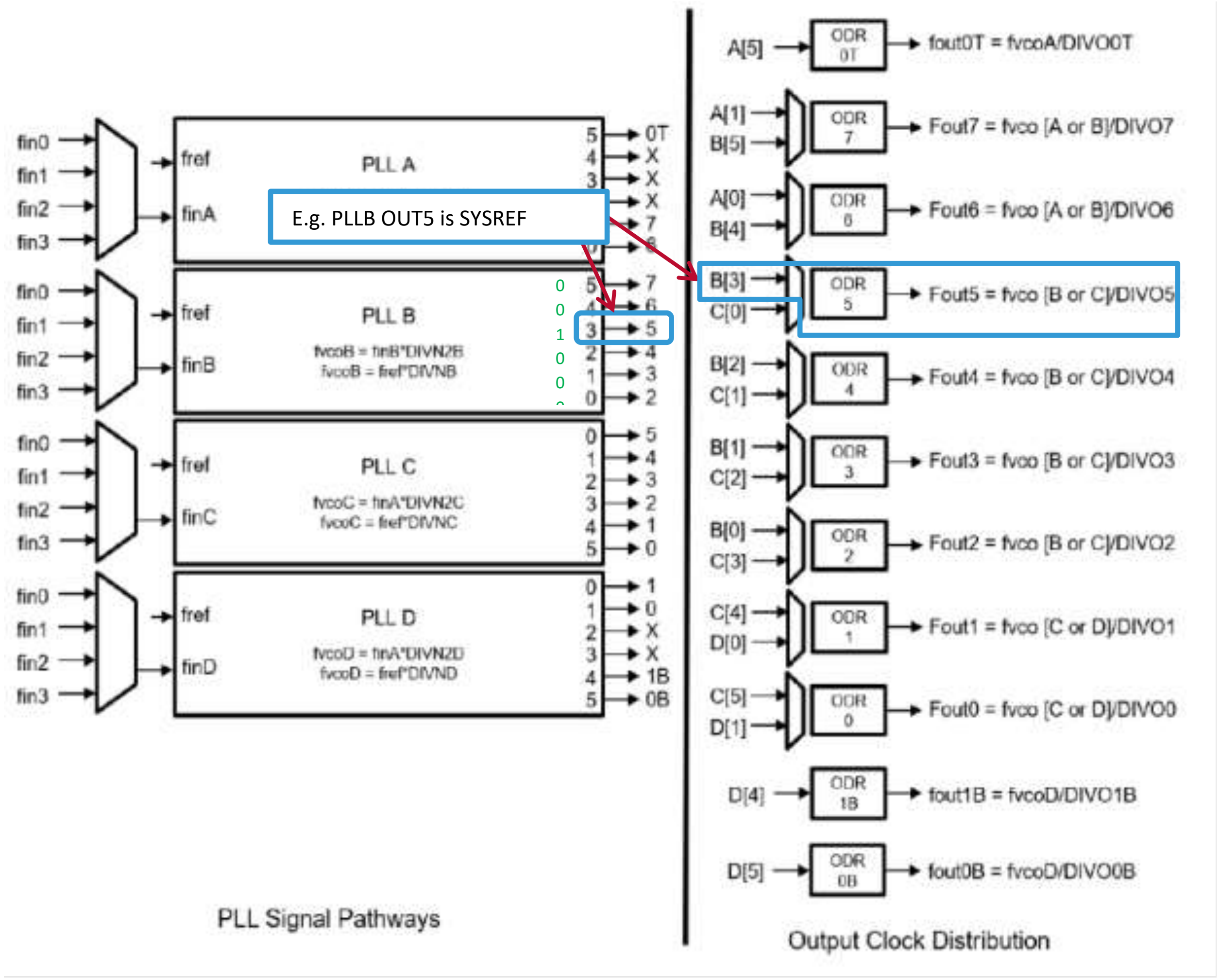


Figure 82: SiT95148 overall clock hierarchy

Table 2: Register settings to enable SYSREF generation from trigger on FLEXIO13

Reg 0x19, Bit#	Function
0	Set high for PLLA generating SYSREF from trigger on FlexIO13
1	Set high for PLLB generating SYSREF from trigger on FlexIO13
2	Set high for PLLC generating SYSREF from trigger on FlexIO13
3	Set high for PLLD generating SYSREF from trigger on FlexIO13
4	Keep this bit low = 0
5	Keep this bit high = 1
6, 7	Do not change the values, default is 0,0.

Generating SYSREF via registers

SYSREF generation can also be controlled by toggling bit 1 of register **0x05** in the respective PLL. The following steps outline the procedure to gate **SYSREF** on a specific output using register writes:

1) Follow steps 1 to 6 as described in the previous section.
2) Register **0x05**, bit **1 [1]** is the **SYSREF** trigger bit (SiTime recommends reading the current value of the **0x05** register and then using the **current value** to change bit 1), see Figure 83.

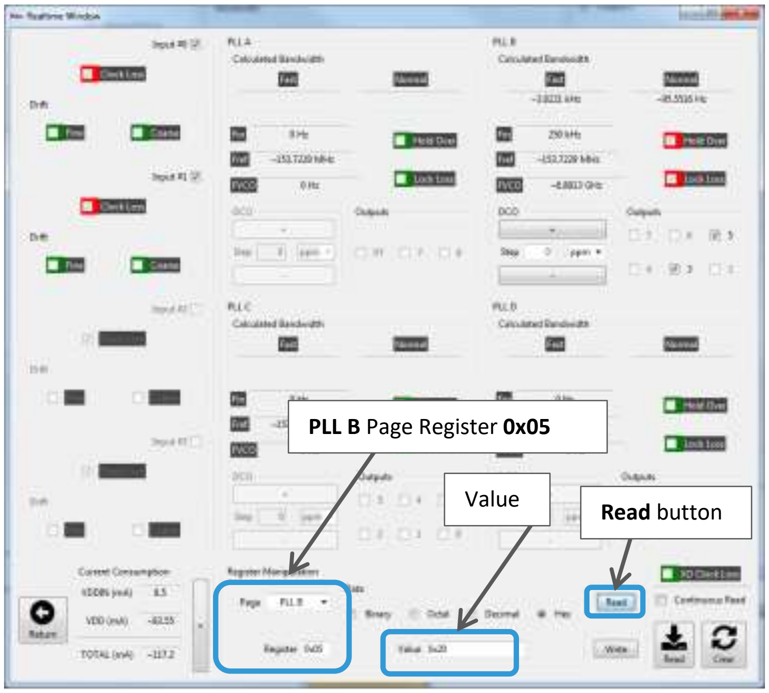


Figure 83: Reading Page PLL B Register 0x05 *before* changing bit[1] of the SiT95148 for JESD204B

a. Enable SYSREF on the selected output, set bit 1 high, see Figure 84.

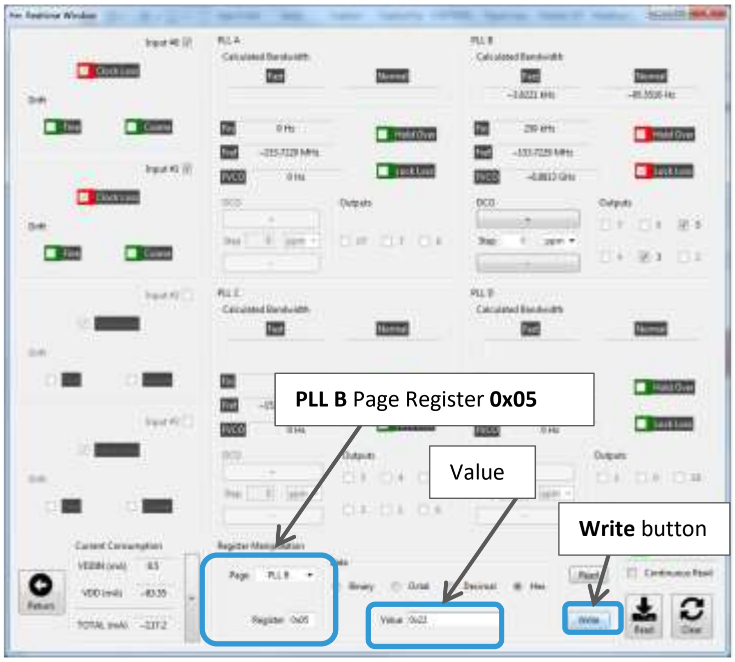


Figure 84: Write bit[1] high to the Page PLL B Register 0x05 of the SiT95148 to enable SYSREF

b. Disable SYREF on the selected output, set bit 1 low (see Figure 85)

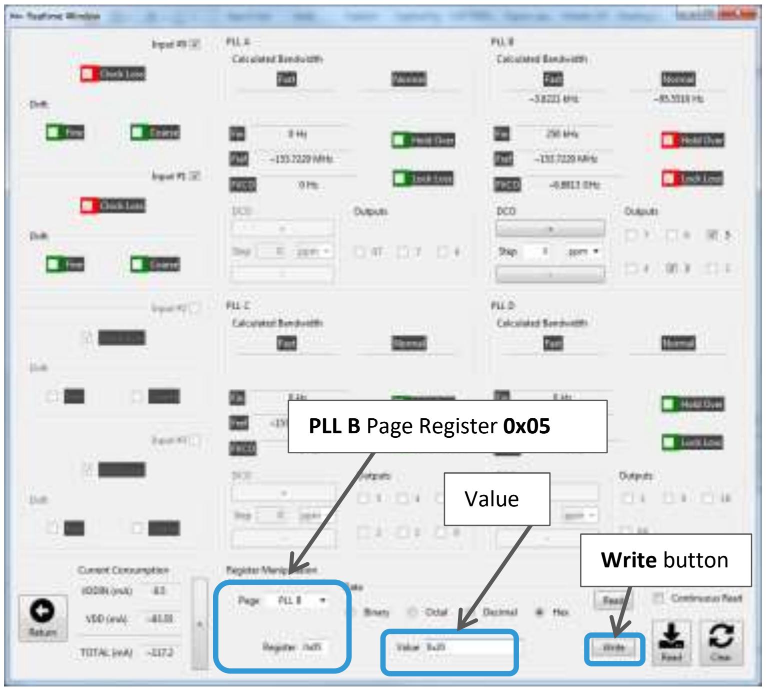
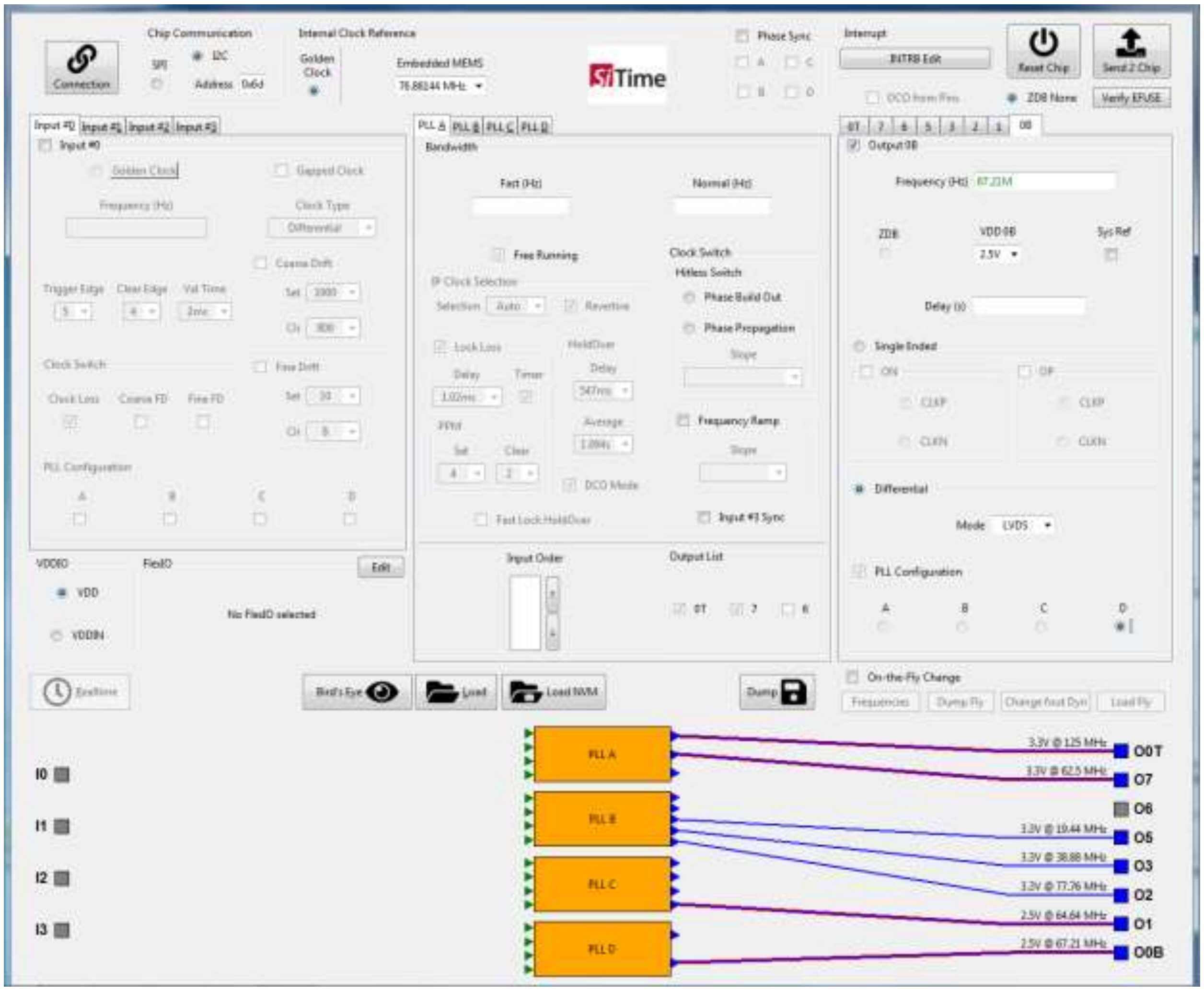


Figure 85: Write bit[1] low to the Page PLL B Register 0x05 of the SiT95148 to disable SYREF

8 Snapshots of specific use case scenarios



Frequency (Hz) 67.21M
VDD 0B 2.5V
Mode LVDS

Figure 86: Free run scenario 1

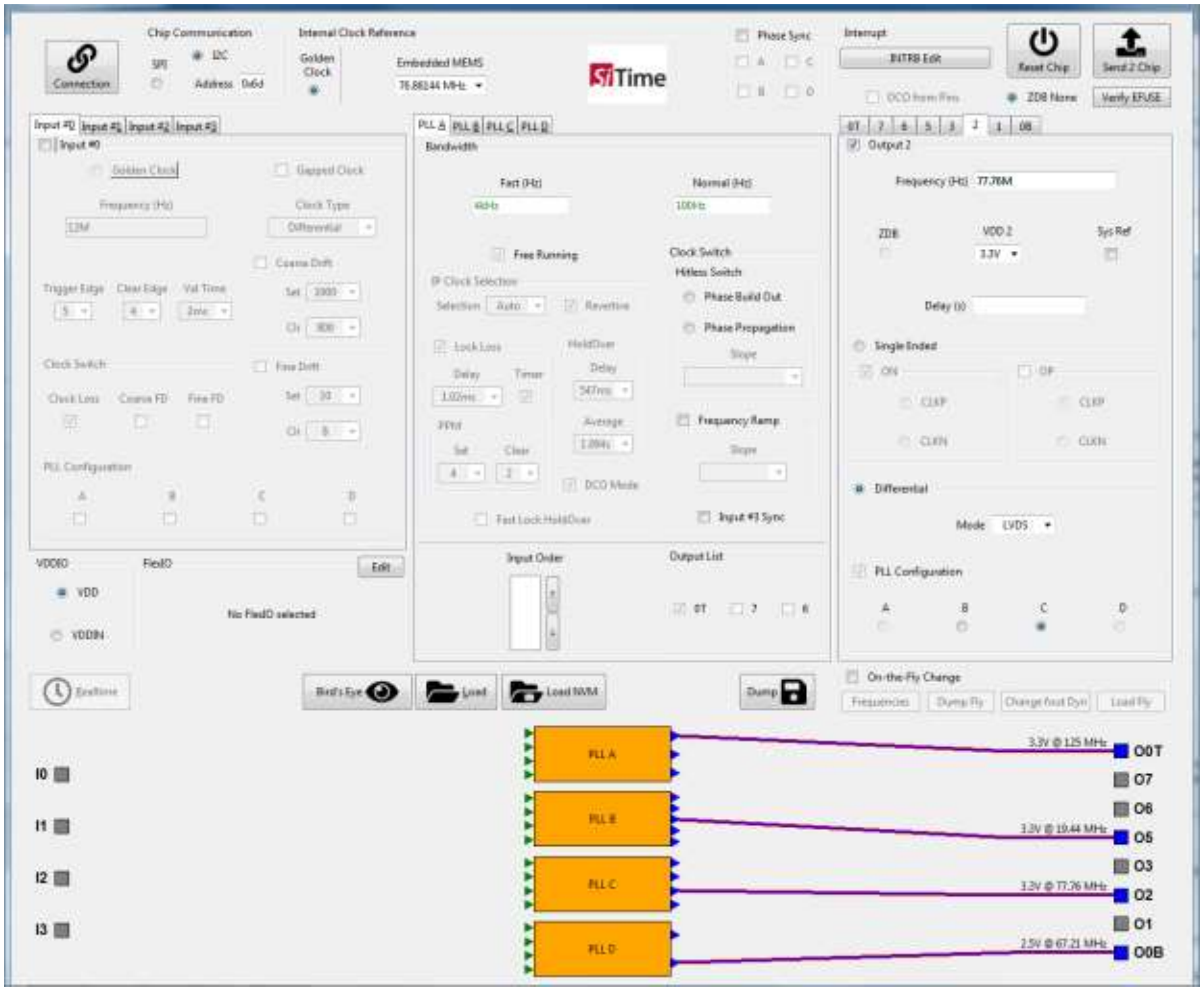
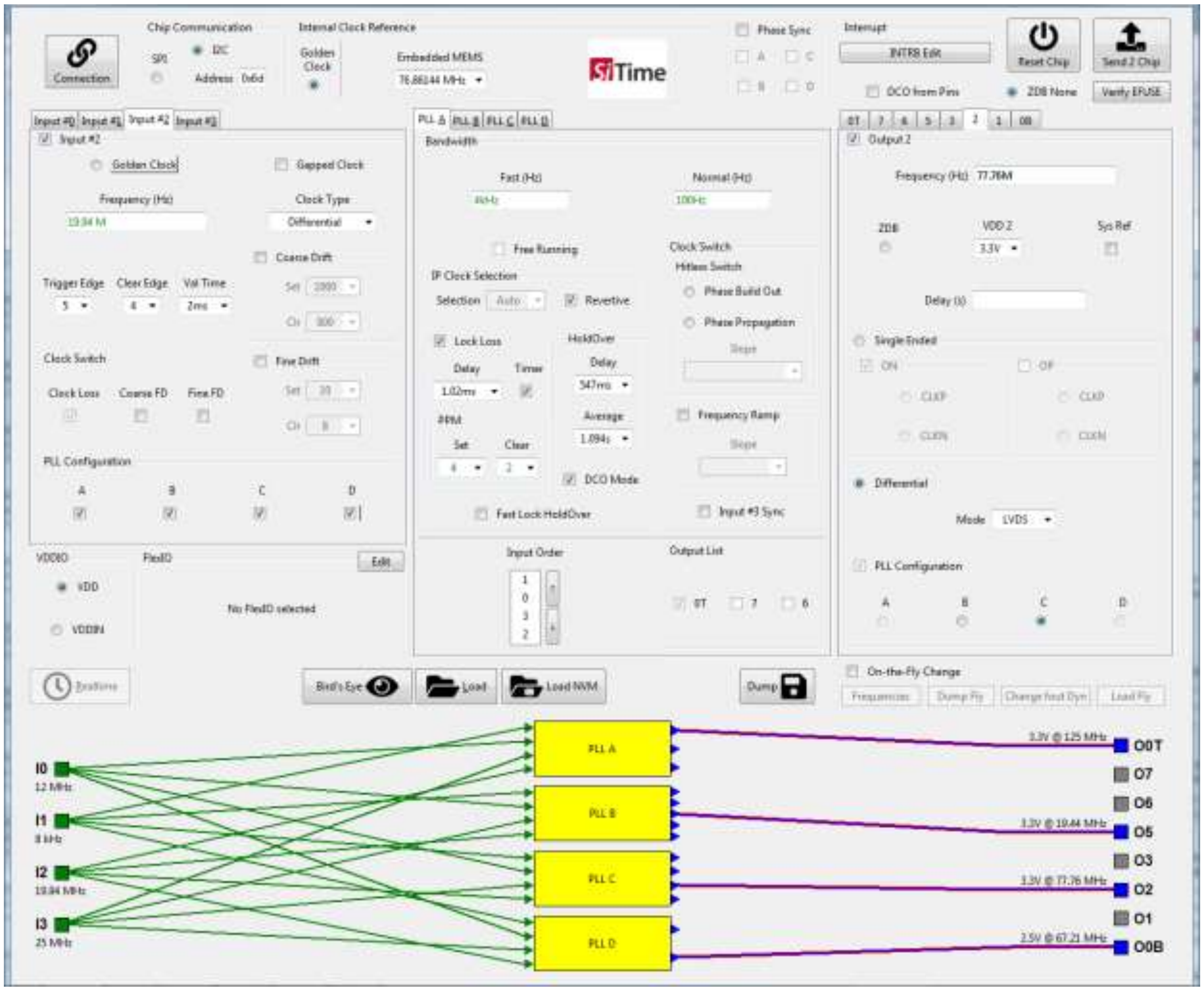

Chip Communication: I2C, Address 0x6d
Internal Clock Reference: Golden Clock, Embedded MEMS 76.80144 MHz
Input #0: Frequency (Hz) 12M, Clock Type Differential
Bandwidth: Fast (Hz) 4kHz, Normal (Hz) 100Hz
Output 2: Frequency (Hz) 77.76M, VDD 2 3.3V, Differential, Mode LVDS, PLL Configuration C
3.3V @ 125 MHz O0T
3.3V @ 19.44 MHz O5
3.3V @ 77.76 MHz O2
2.5V @ 67.21 MHz O0B

Figure 87: Free run scenario 2


Chip Communication: SPI [✓ I2C] Address 0x6d
Internal Clock Reference: Golden Clock [✓]; Embedded MEMS 76.86144 MHz

Input #2 [✓]
Golden Clock
Gapped Clock [ ]
Frequency (Hz): 19.44 M
Clock Type: Differential
Trigger Edge: 5 | Clear Edge: 4 | Val Time: 2ms
Coarse Drift [ ] Set 2000 Clr 300
Fine Drift [ ] Set 20 Clr 8
Clock Switch: Clock Loss, Coarse FD, Fine FD
PLL Configuration: A [✓] B [✓] C [✓] D [✓]
VDDIO: [✓ VDD] VDDIN
FlexIO: No FlexID selected

PLL A
Bandwidth: Fast (Hz) 4kHz | Normal (Hz) 100Hz
Free Running [ ]
IP Clock Selection: Selection Auto; Revertive [✓]
Lock Loss [✓]: Delay 1.02ms, Timer [✓]; #PM Set 4, Clear 2
HoldOver: Delay 547ms; Average 1.094s
DCO Mode [✓]
Fast Lock HoldOver [ ]
Clock Switch: Hitless Switch — Phase Build Out / Phase Propagation
Frequency Ramp [ ]
Input #3 Sync [ ]
Input Order: 1, 0, 3, 2
Output List: 0T [✓] 7 [ ] 6 [ ]

Phase Sync [ ]: A [ ] B [ ] C [ ] D [ ]
Interrupt: INTRB Edit
DCO from Pins [ ] [✓ ZDB None]

Output 2 [✓]
Frequency (Hz): 77.76M
ZDB | VDD 2: 3.3V | Sys Ref [ ]
Delay (s):
Single Ended: ON / OF — CLKP, CLKN
[✓ Differential] Mode: LVDS
PLL Configuration: A B [✓ C] D
On-the-Fly Change [ ]

Figure 88: Lock scenario 1

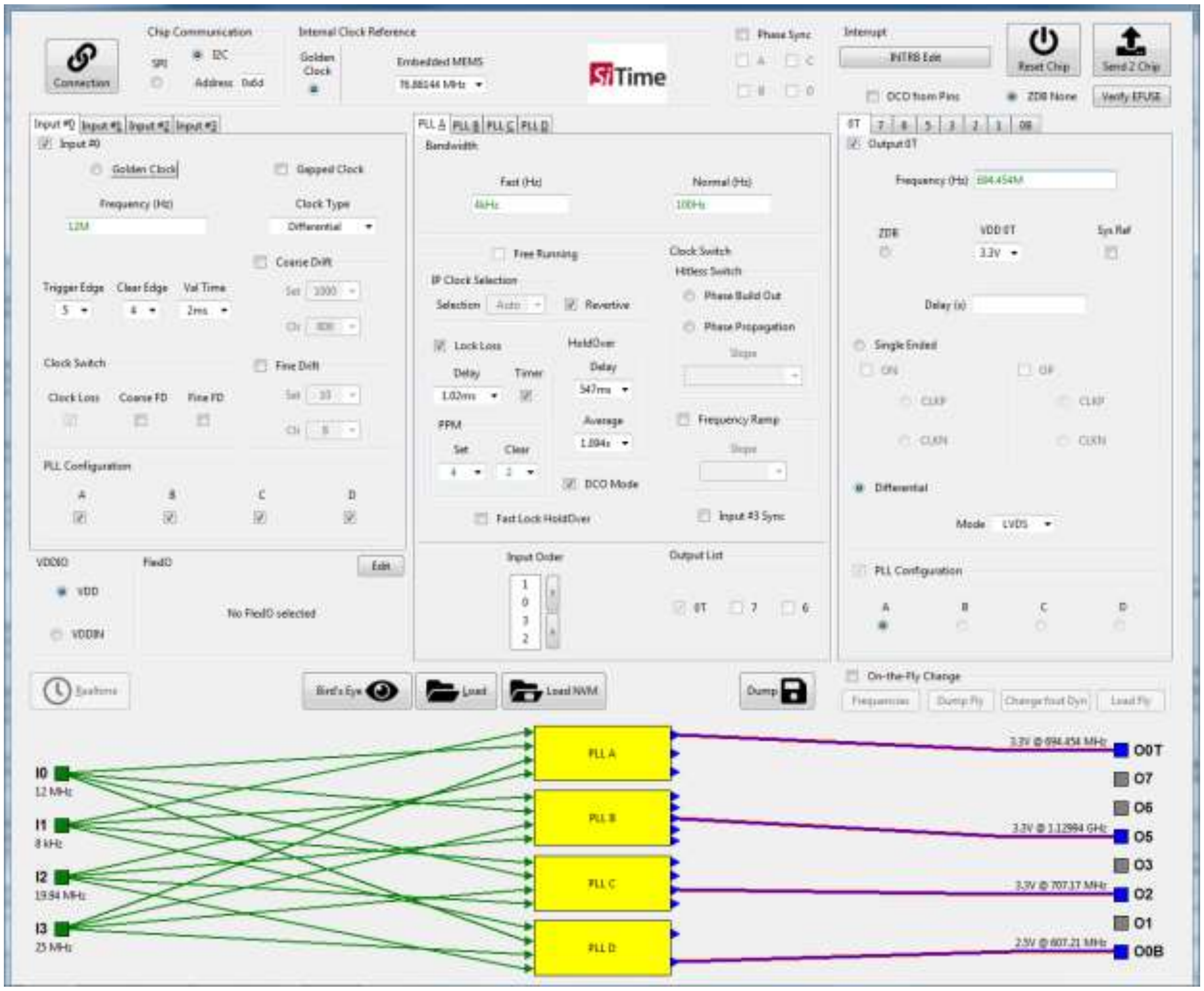


Figure 89: Lock scenario 2

8.1 Free running DCO

After programming the chip, click the **Realtime** button to use the DCO mode, see Figure 90.

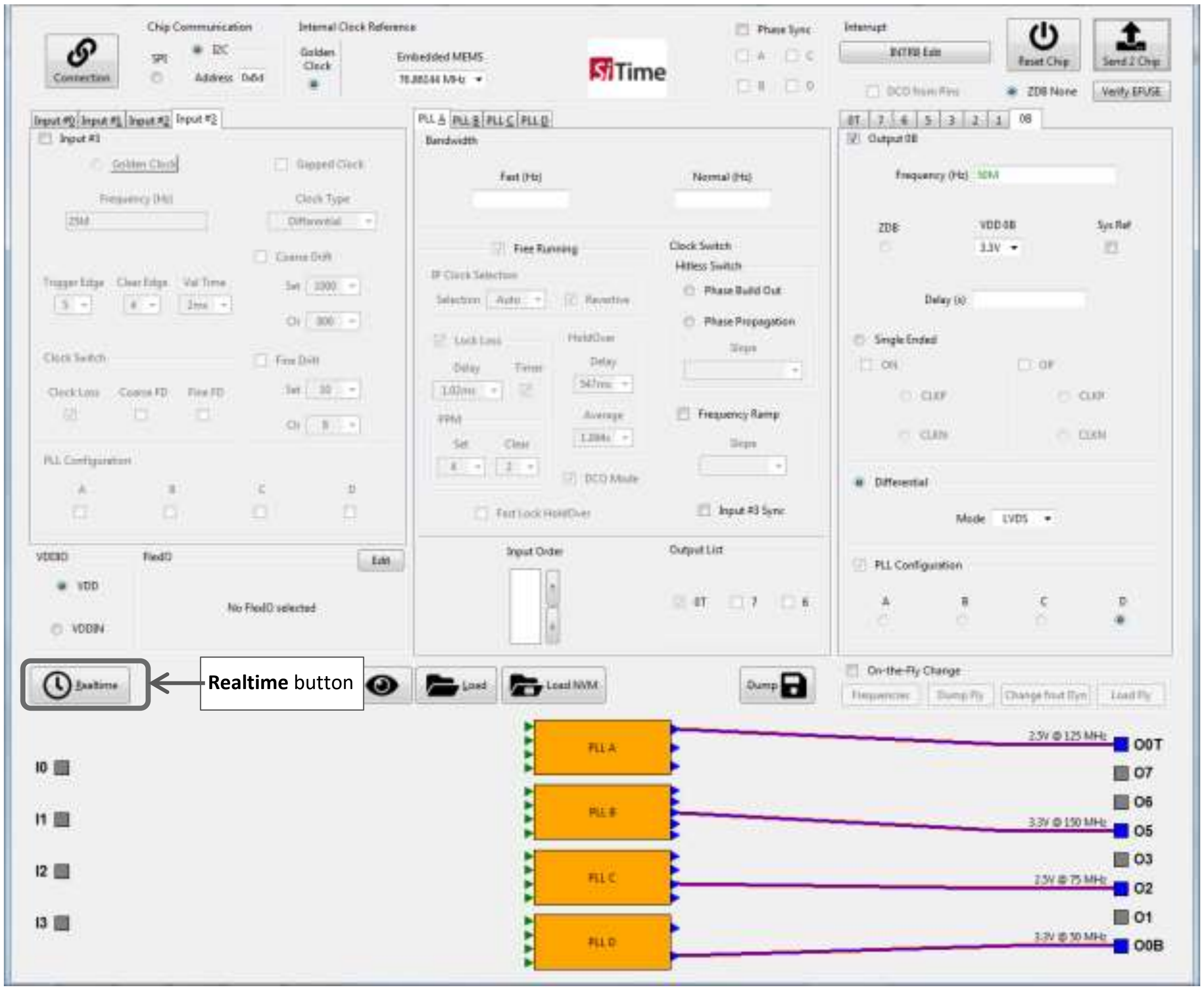


Figure 90: Free Running DCO Mode configuration

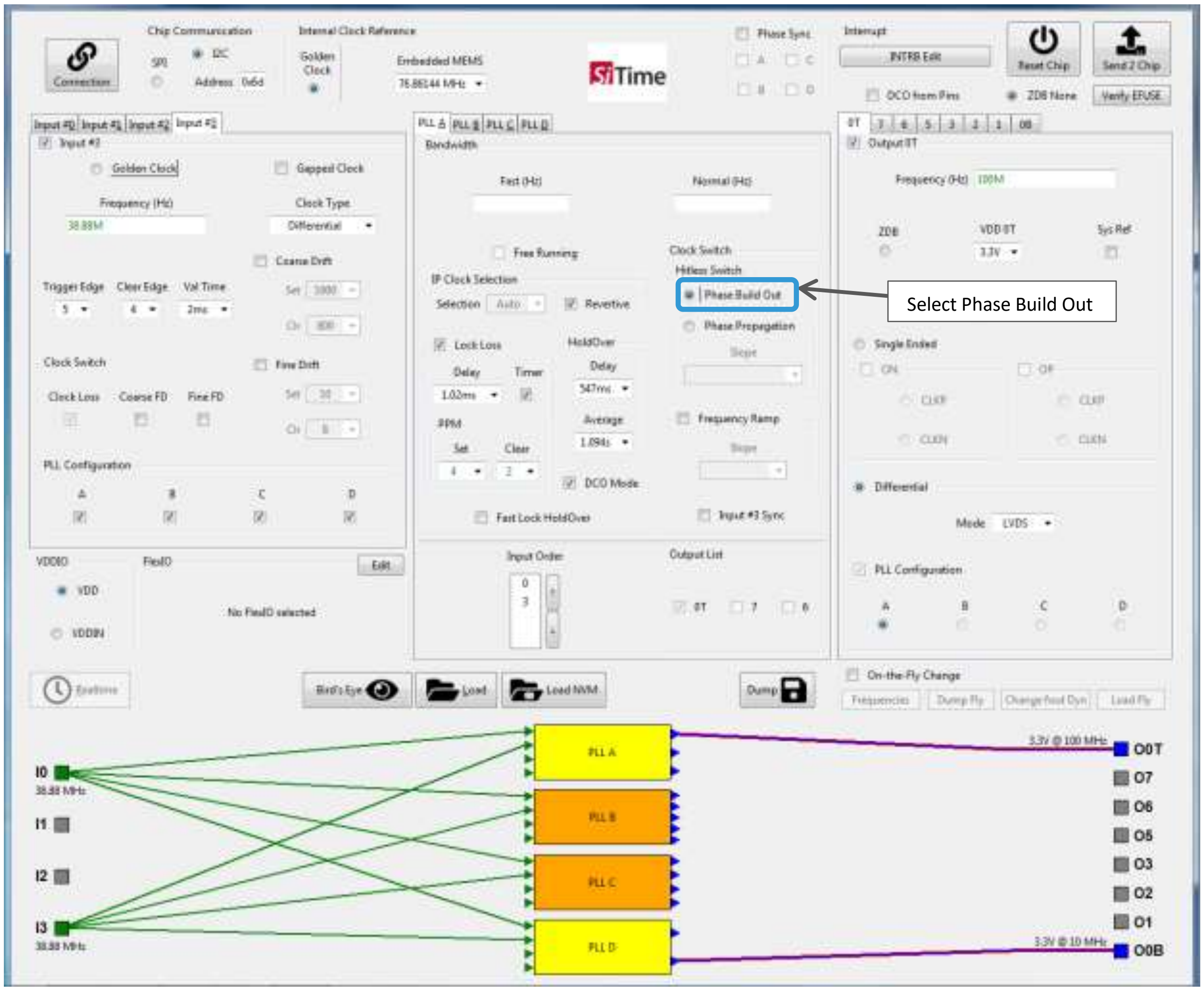


Figure 91: Hitless Switch (Phase Build Out setting)

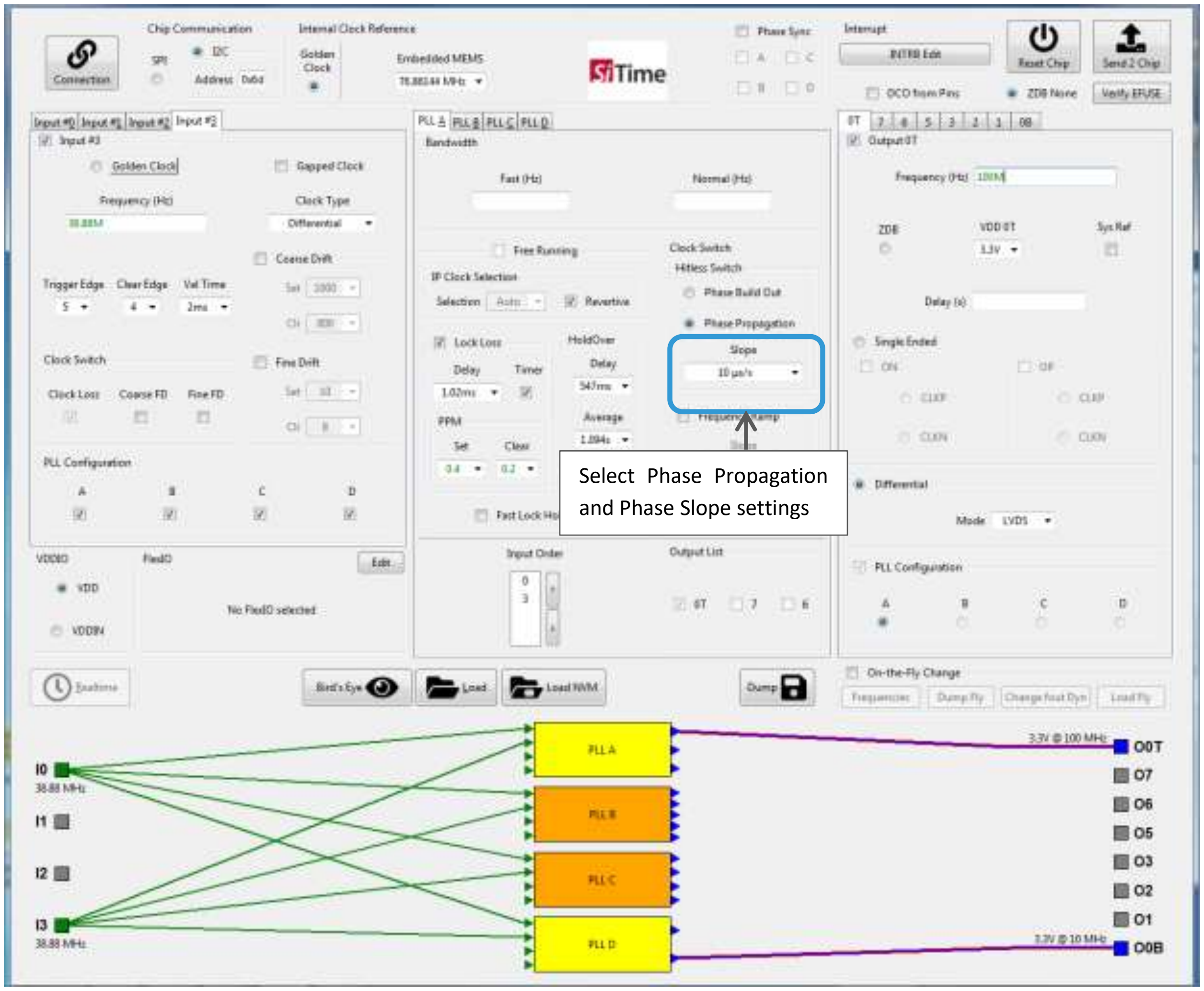


Figure 92: Clock Switch (Phase Propagation and Slope settings)

8.2 Zero-delay buffer mode

A zero-delay buffer (ZDB) is available in the SiT95141 clock generator and SiT95145/7/8 clock jitter attenuators for use in applications that require minimum delay between the selected input and output.

The ZDB mode is available and can be configured for any of the PLLs. This provides the option to close the feedback loop of the PLL on the PCB and thereby, bypassing the internal feedback dividers, and cancelling the delays introduced by the internal dividers and clock distribution pathways. The **Input #3** pins are used as the external feedback and any of the outputs from the PLL which is being set up in ZDB mode should be routed to the **Input #3** differential inputs. SiTime recommends using **Input #0** as the input clock when using **Input #3** as the external feedback clock in ZDB mode. The terminations used for **Input #3** would depend on the driver type chosen. The preferred option is to use an LVDS or LVDS boost output AC-coupled into a differential 100 Ω termination at the **Input #3** input side, see Figure 93.

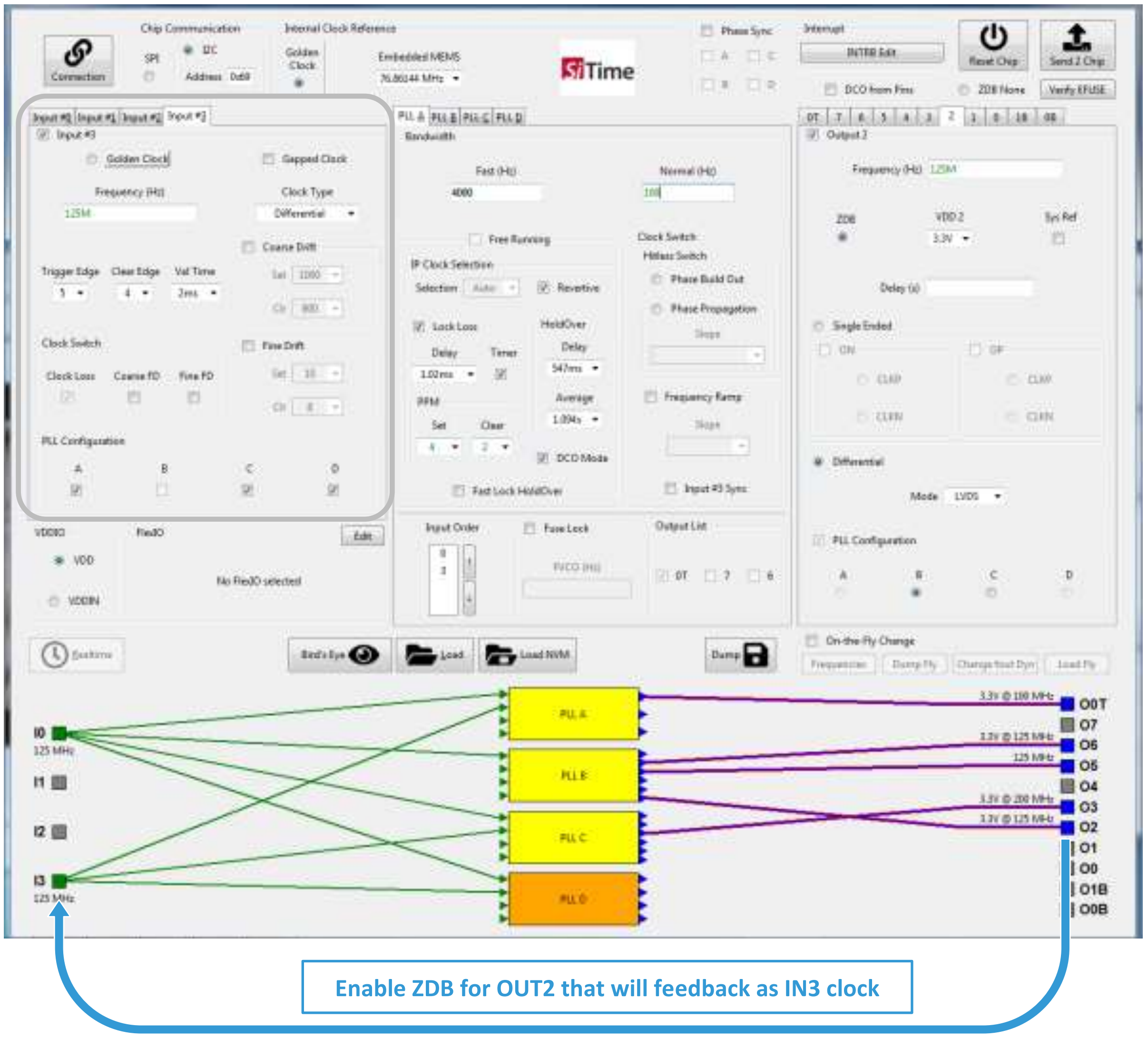


Figure 93: Zero-delay buffer (ZDB) mode

9 Low wander mode

Low wander mode provides the best jitter cleaning by employing a dual loop PLL, see Figure 94. Refer to the datasheet and application note for a complete description.

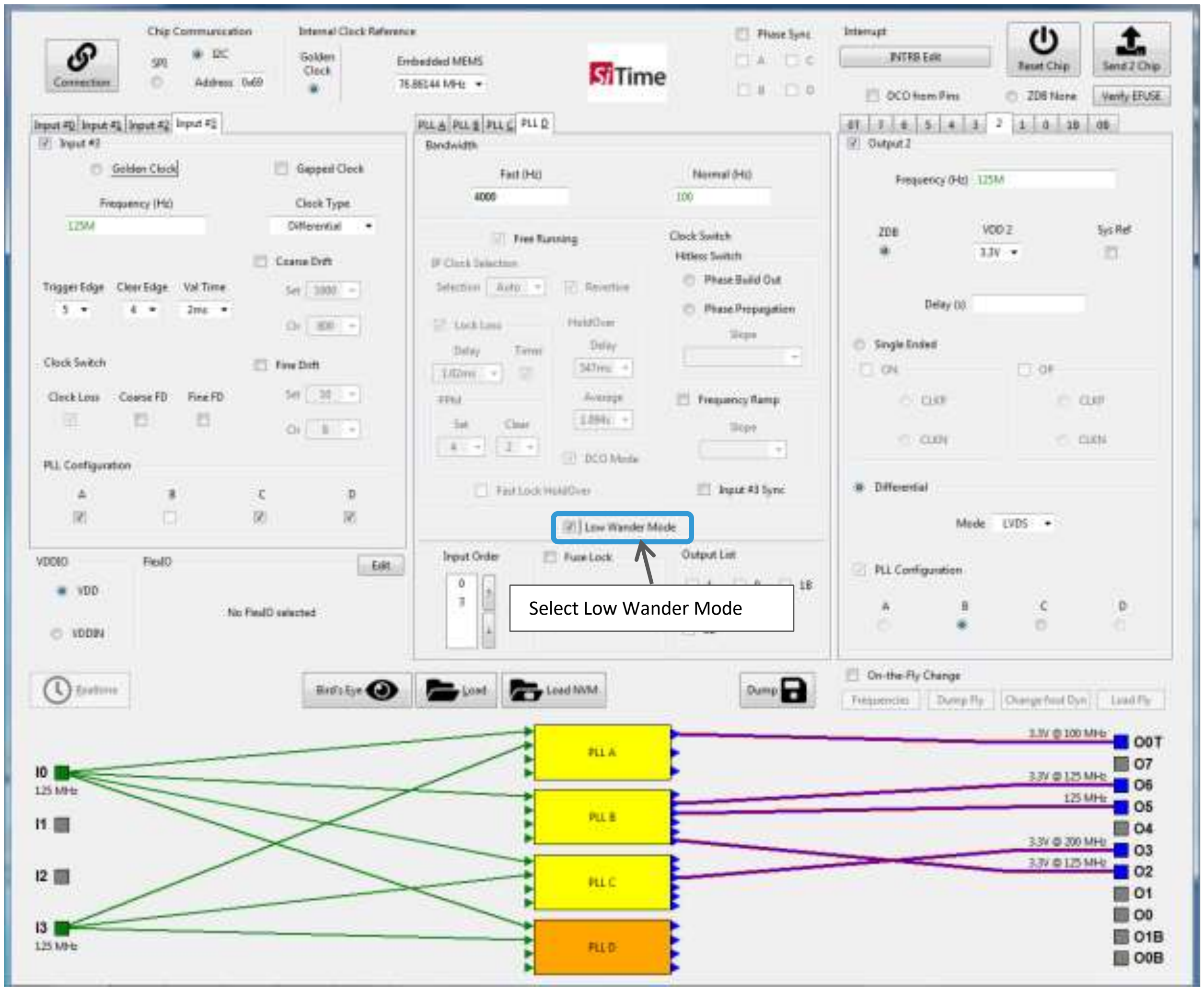


Figure 94: Selecting low wander mode

10 Usage guidelines for jitter performance optimization

10.1 Output placement and frequency planning

The output placement and frequency planning should be such that outputs with a frequency difference in the range 12 kHz to 20 MHz should not be placed next to each other and should be spaced as far apart as possible to minimize output-to-output coupling leading to in-band spurs.

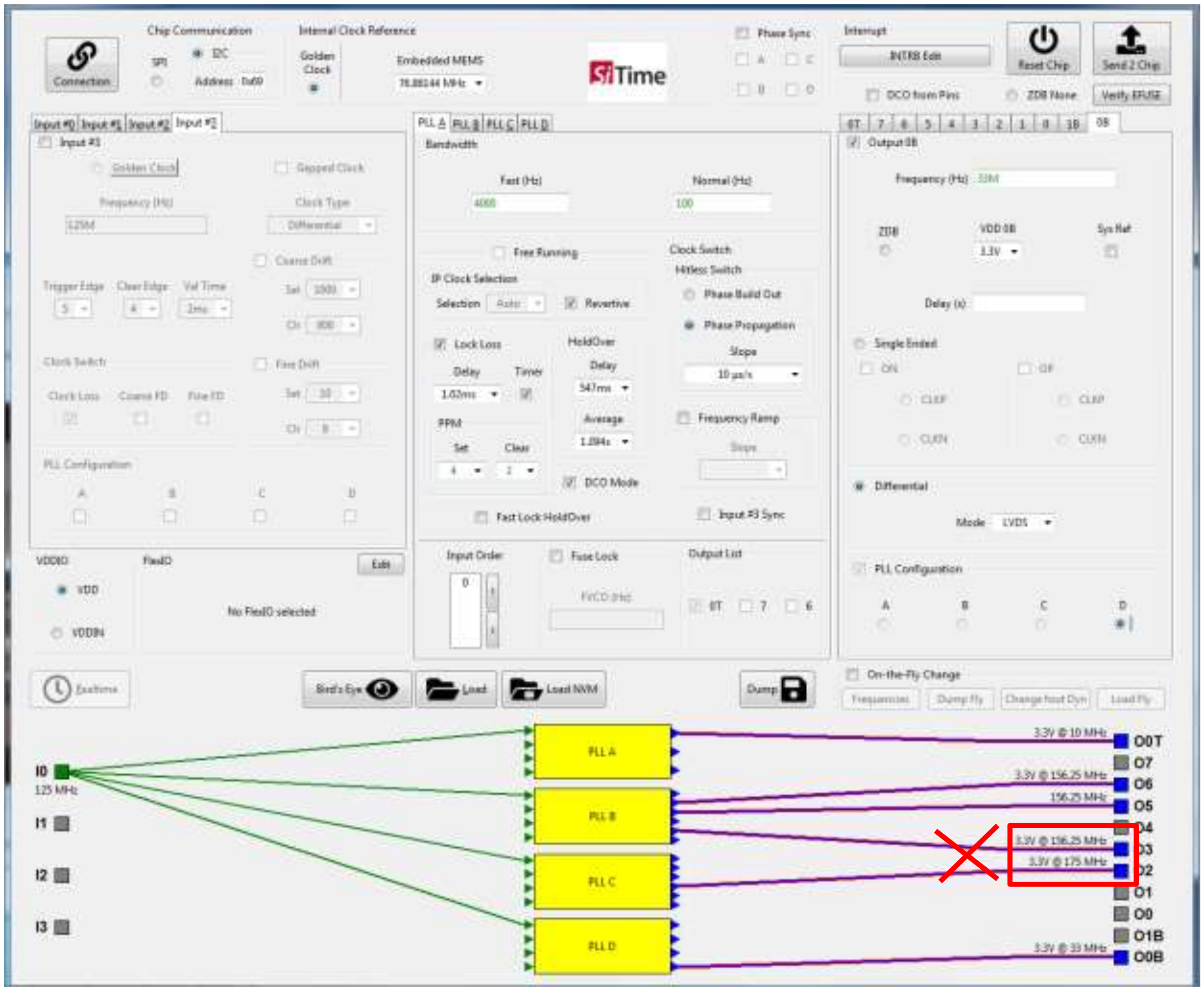
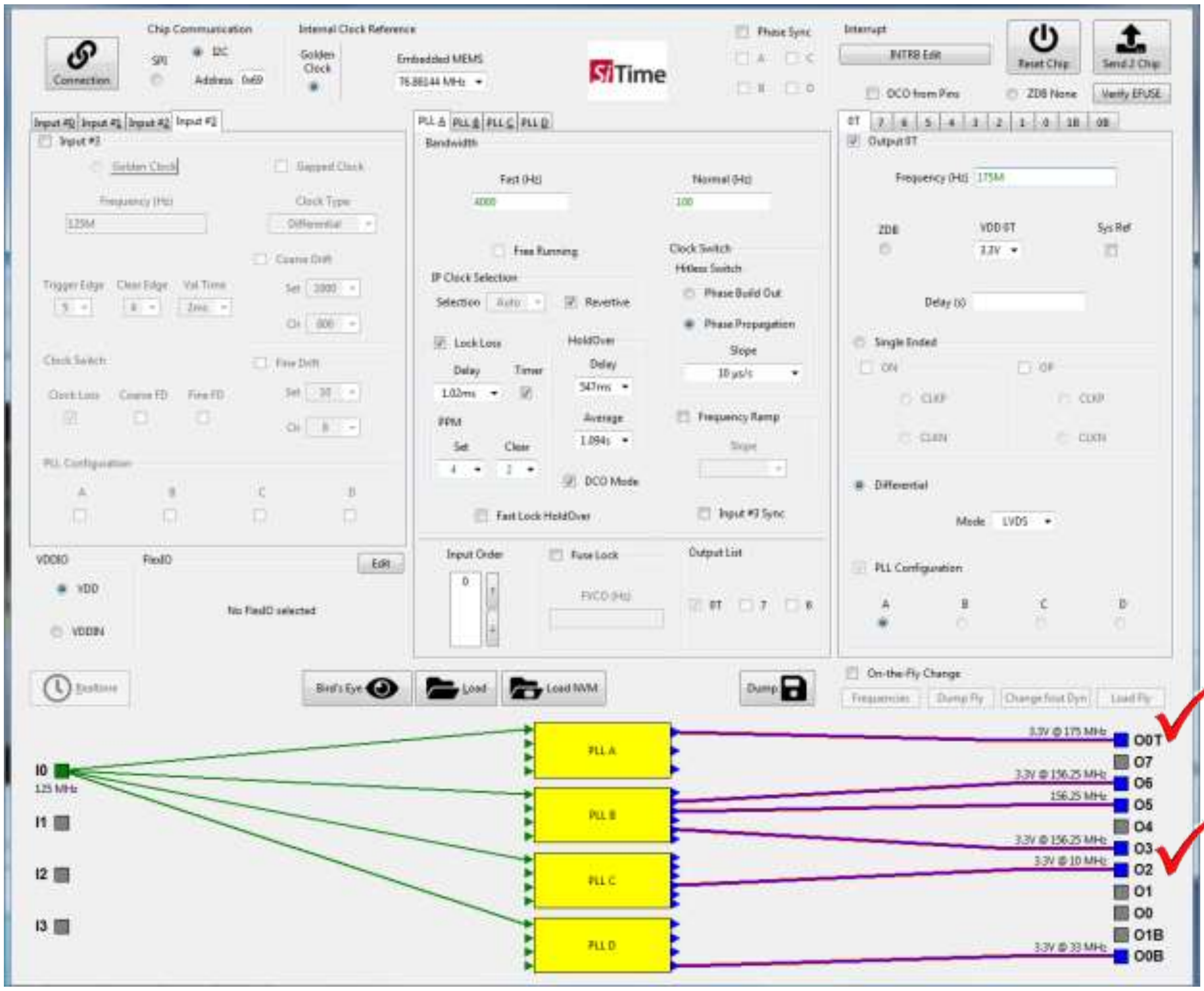


Figure 95: Example where outputs O3 and O2 should be spaced apart


Chip Communication: SPI, [✓ I2C], Address 0x69
Internal Clock Reference: Golden Clock, Embedded MEMS 76.8 MHz

Input #3: Frequency (Hz) 125M; Clock Type Differential; Trigger Edge 5; Clear Edge 4; Val Time 2ms; Coarse Drift Set 2000, Clr 800; Fine Drift Set 10, Clr 8

PLL A: Bandwidth Fast (Hz) 4000, Normal (Hz) 100; Lock Loss Delay 1.02ms, PPM Set 4, Clear 2; HoldOver Delay 547ms, Average 1.094s; [✓ DCO Mode]; Clock Switch: Hitless Switch [✓ Phase Propagation], Slope 10 µs/s

VDDIO: [✓ VDD] VDDIN; FlexIO: No FlexIO selected

Output 0T: Frequency (Hz) 175M; VDD 0T 3.3V; Differential, Mode LVDS; PLL Configuration A

O0T	3.3V @ 175 MHz
O6	3.3V @ 156.25 MHz
O5	156.25 MHz
O3	3.3V @ 156.25 MHz
O2	3.3V @ 10 MHz
O0B	3.3V @ 33 MHz

Figure 96: Optimal profile (outputs re-arranged to minimize spurs)

Additionally, for configurations having adjacent LVCMOS outputs, avoid up to the third-harmonic of the adjacent LVCMOS output, to land within the integration frequency band of the jitter sensitive output frequency, see Figure 97.

For Outputs O5/O3 (156.25 MHz), a crosstalk spur location may occur due to adjacent placement of the 50 MHz LVCMOS (50 MHz) outputs.

156.25 MHz – 3*50 MHz = 6.25 MHz

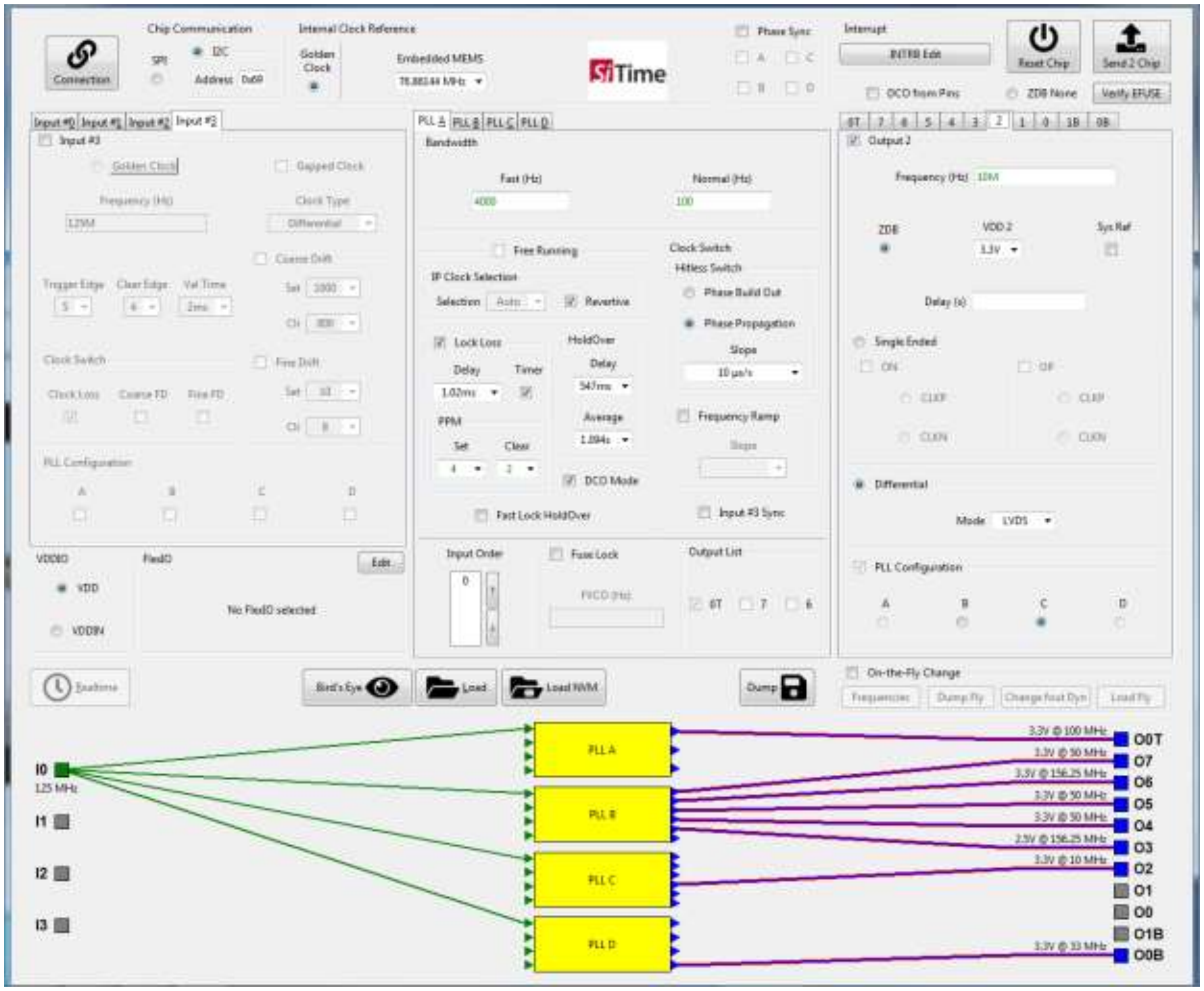
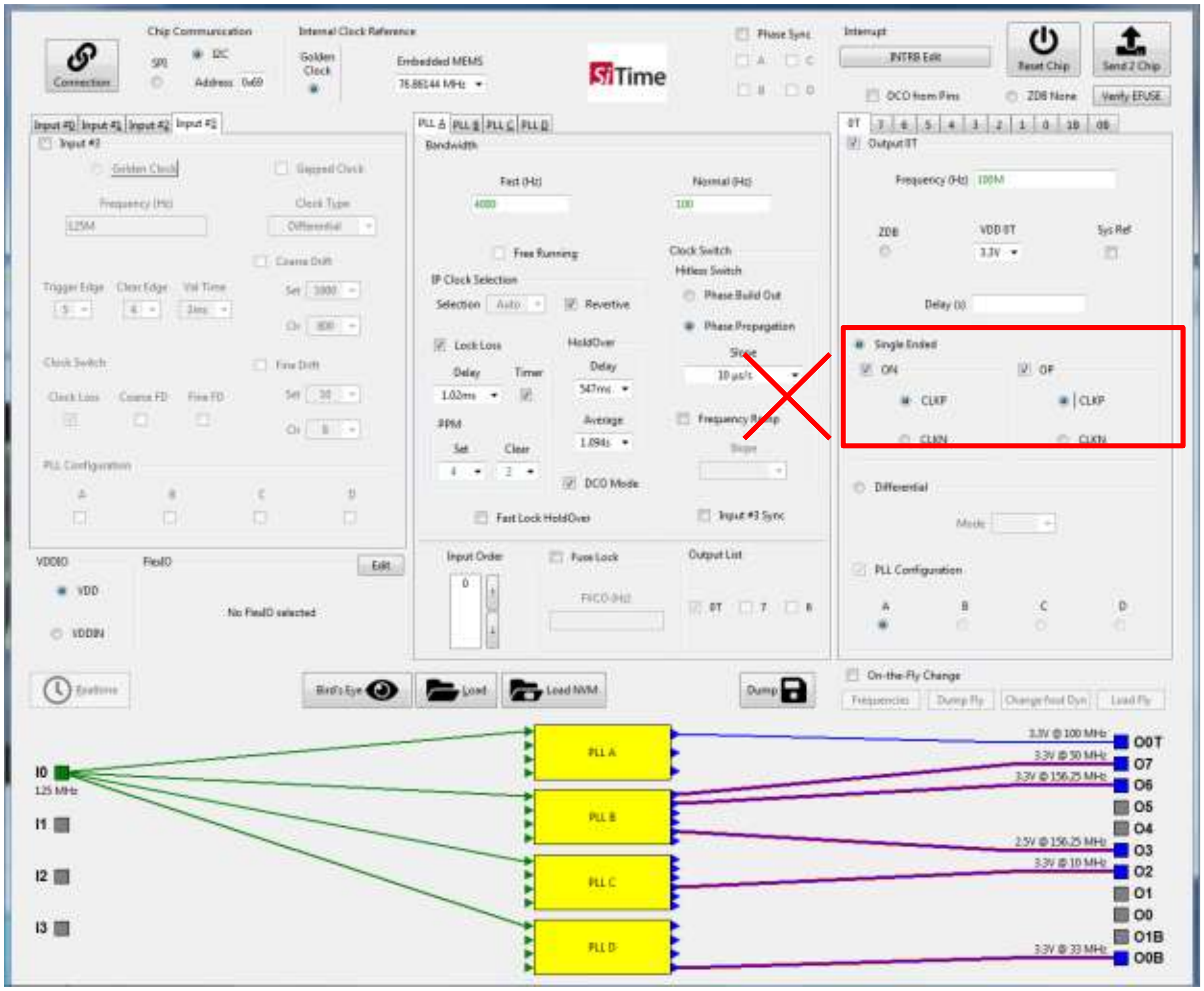


Figure 97: Third harmonic of the adjacent LVCMOS output

10.2 CMOS output type selection

When setting the **Single Ended** CMOS output format, it is recommended to always select complimentary outputs to minimize single-ended CMOS output-to-differential-output coupling.

For complimentary CMOS format, the output type (**Output OT**) selection should be with **ON** and **OP** selected, **ON=CLKN**, and **OP=CLKP**.



Output OT
Frequency (Hz): 100M
VDD OT: 3.3V
[✓ Single Ended] Differential
[✓ ON]: [✓ CLKP] CLKN
[✓ OP]: [✓ CLKP] CLKN

Figure 98: Example of an *incorrect* profile for complimentary CMOS format

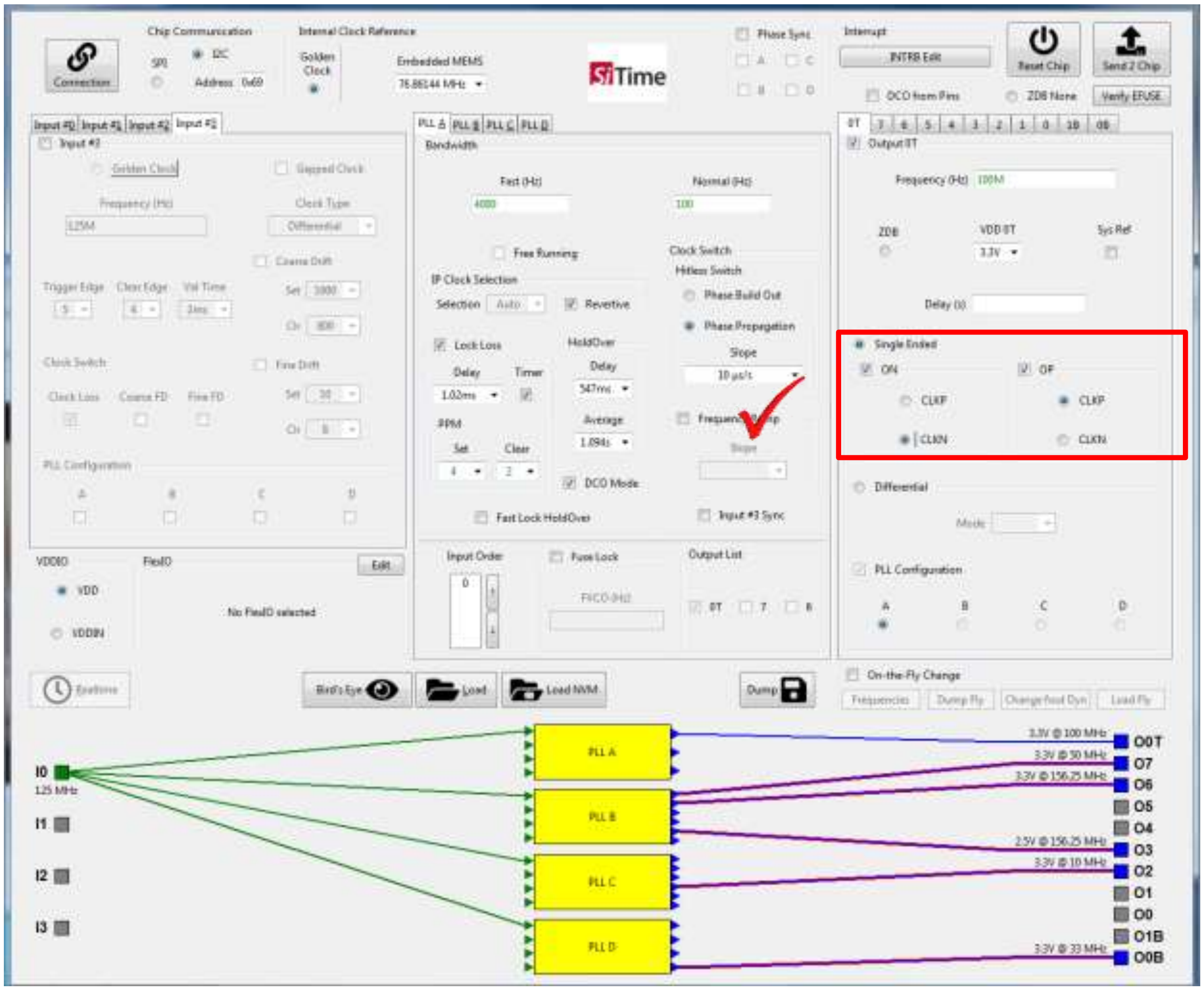


Figure 99: Optimal profile for complimentary CMOS format

11 Document Information

Table 3: Revision history

Version	Release Date	Change Summary
1.0	31-Jan-2019	Original doc
1.01	11-Jun-2019	Corrected block diagram in section 3
1.02	30-Mar-2020	Changed according to GUI rev.1.28.4rc4
1.03	10-Nov-2020	Added Low Wander Mode description Updated rev table date format
1.04	16-Feb-2021	Extensive editorial changes throughout.

SiTime Corporation, 5451 Patrick Henry Drive, Santa Clara, CA 95054, USA | **Phone:** +1-408-328-4400 | **Fax:** +1-408-328-4439