



# A Synthesized and Consistent Clock Management Solution Covering Full Pre-silicon Steps

Bin Liu, Shunjun Lou, Li Li

Modem Hardware Group

Intel Inc.

Xi'an, Shaanxi, China

[www.intel.com](http://www.intel.com)

## ABSTRACT

*Along with the size and complexity are booming, the clock integration arose as a challenging step. Benefited by modern clock optimization technics, it also introduced new complexity to the clock system. In the legacy approach, clock integration would be manually taken by multiple departments. Meanwhile, different focus points of clock possibly leads to error modification of clock data, and the irregular clock management flow would easily result in error-prone clock data update and further affected other related steps. Regarding the characteristic of clock management, the paper initiated a solution named iClock which synthesized the common participatory from multiple departments, and managed to maintain a consistent clock database. Besides the concept, a practical example would be applied to illustrate the advantage of iClock coordinating and covering full pre-silicon steps. By iClock, each objective department would easily extract the required data from iClock, and the data consistency also got ensured for error-free use. The procedure of Spyglass and Design Compiler has been offered benefits by this solution. The initiative of iClock would be referred to inspire more adaptive and user specific clock management solutions.*

## Table of Contents

1. Introduction .....	4
2. Background .....	5
3. Clock Table and Consistent Database.....	8
3.1 Clock Table Merge.....	9
3.2 Clock Table Analysis .....	9
3.3 Smart Clock Edition.....	12
3.4 Edition History Backtrace.....	12
4. Clock Extraction for Different Procedure .....	13
4.1 Spyglass CDC Script for DE.....	14
4.2 Frequency Requirements for VE.....	14
4.3 STA Script for BE.....	15
5. GUI Mode for Feature Enhancement .....	16
5.1 Clock Tree Window .....	16
5.2 Clock Graph Window.....	17
5.3 Clock Search Window.....	17
6. Summary .....	18
7. Prospect.....	19
8. Acknowledgement.....	19
9. References .....	19

## Table of Figures

Figure 1 Hierarchical clock structure.....	4
Figure 2 Clock system integration full steps in pre-silicon.....	5
Figure 3 Clock info maintenance flow by iClock.....	6
Figure 4 Clock tables' merge to full chip clock table by iClock.....	6
Figure 5 iClock extracted Excel form.....	7
Figure 6 Main window of iClock GUI .....	7
Figure 7 Clock tree graph of iClock GUI.....	8
Figure 8 Clock tree window of iClock GUI.....	8
Figure 9 Clock consistence check.....	10
Figure 10 Clock data edition menu .....	11
Figure 11 Clock Edition Table .....	11

Figure 12 Different clock groups with easily recognized color .....	12
Figure 13 Clock Table Comparison .....	13
Figure 14 iClock Overall Layout .....	16
Figure 15 Clock tree window usage.....	17
Figure 16 Smart clock search .....	18

## 1. Introduction

Modern clock system design concept has introduced series of power saving technics which also made the clock system more complex. As the Fig.1 shows, a typical structured clock system would involve PLLs (Phase-Locked Loop) to generate basic source clocks, MUXes (Multiplexer) to select which clock as the output, dividers to generate low frequency clocks, and clock gating cells to switch on or off the clocks. The introduction of those clock design elements made it difficult to define, implement and layout the clock system.

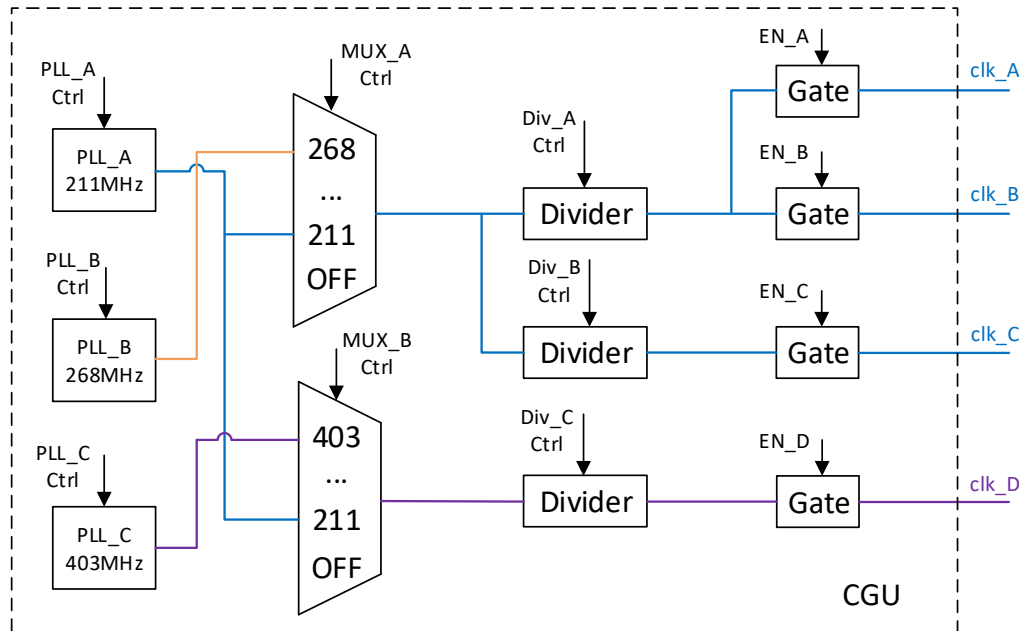


Figure 1 Hierarchical clock structure

In general, the clock information would be given concept definition from system engineer (SE), and the information involves the kernel clock data such as the clock tree and clock frequency. Such a form is usually edited manually and maintained in document, and Excel form is a popular way to consist the highly structured data. Then the core clock data form would be forward to the design engineer (DE) who would complement more clock information to specify constraints of those clocks. Here the clock DE plays as the close interface between SE, verification engineer (VE) and backend engineer (BE) because DE would pick up the core clock information from SE and make design clock form. It is risky for the clock information extraction from SE's to DE's. Regarding dozens of subsystems and hundreds of blocks integrated into the SoC, the clock design data would be distributed and maintained by each subsystem clock DE, which commonly follows the top level clock design requirement. Finally, those scattered clock data would be merged together to draw the overall system clock tree. However, it still faces the hazard when different experienced engineers would possibly failed to make qualified clock data. This will then further affect the quality of top level clock integration. Besides, the information mismatch between top level and subsystem level would result in the integration risk too. As described, the long and manually operation steps are frequently done by different engineers, and clock data consistency check is rarely introduced to guarantee the clock data integration success.

The full pre-silicon steps of clock integration would also involve clock information extraction from the clock design database. Due to the lack of available clock information container, both SE and DE often edit clocks via the Excel but had to tolerate the edition inconvenience and take the clock

integration risk. It is given by Fig.2 that VE, BE and DFT would extract necessary clock information from the database, and implemented the specific script to go through their own steps:

- VE would extract clock relationship and names to prepare the clock domain definition script for Spyglass CDC process, and also to guide the testbench clock generator to produce the expected clock frequency.
- BE would extract more clock information from the clock database manually and scripted for the backend tooling Design Compiler to do synthesis.
- DFT engineers would also refer to the clock form, select specific function clock for test clock purpose, and also tell BE the extracted information for corresponding synthesis of DFT areas such as SCAN and BIST.

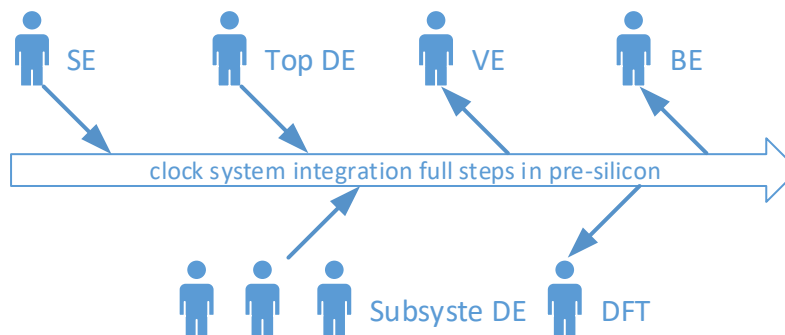


Figure 2 Clock system integration full steps in pre-silicon

Unfortunately, the data extraction steps are also manually operated by different engineers. The risk of data loss and data mismatch would critically affect the quality of verification and layout. Therefore from the clock system integration flow in Fig.2, it is obviously necessary to avoid manual operation, and to improve the clock data consistence check when the database is modified. The centralized clock info management has improved the data accuracy fed with Spyglass and Design Compiler, which would be taken as a customized internal flow binding closely with Synopsys toolings.

In this paper, it would propose a synthesized and consistent clock management solution named iClock to realize the centralized clock storage, error-free clock integration, and quick clock information extraction. iClock is developed based on Python<sup>[1]</sup>, Tkinter<sup>[2]</sup>, and Graphviz<sup>[3]</sup>.

## 2. Background

The crucial point to solve the clock data consistence is to figure out a customized solution, and such solution is not only the centralized clock data storage, but also to satisfy all kinds of clock data extraction request. This is the basic requirement for iClock and it is compatible with the old work mode by Excel, and also supports more flexible data edition and extraction.

As Fig.3 shows, the original clock data base with limited info is delivered from SE, and he usually edits and stores the info by Excel. Afterwards, the excel would be extracted into a consistent database by iClock. This demands iClock to analyze the data format of SE's delivered clock file named BlockClock. DE would directly edit the clock data via iClock or the extracted data Excel format. Both of work modes are supported. It often happens SE would modify the BlockClock database, and it is compulsory to compare the extracted iClock design database and the modified BlockClock database,

which is supported by iClock. From iClock solution concept, its primary target is to ensure the clock data consistence between the SE's and the DE's.

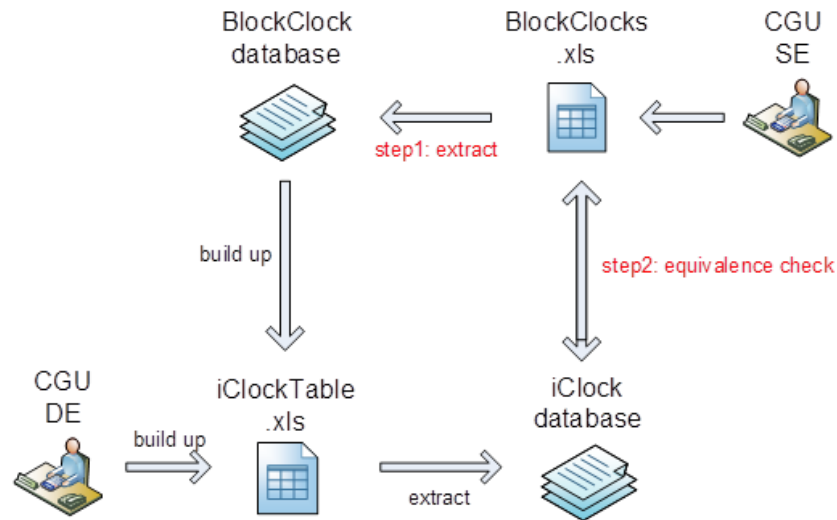


Figure 3 Clock info maintenance flow by iClock

Once the clock data consistence from the origin got assured, top level clock DE would organize the clock data by iClock, and subsystem clock DEs would also build the low level clock tree. Once all of scattered clock info is constituted, iClock would merge the top clock table and other subsystem clock tables together, and produce a full chip clock table. The merge step is given by Fig.4, and it would be generally taken several times once any clock table is updated.

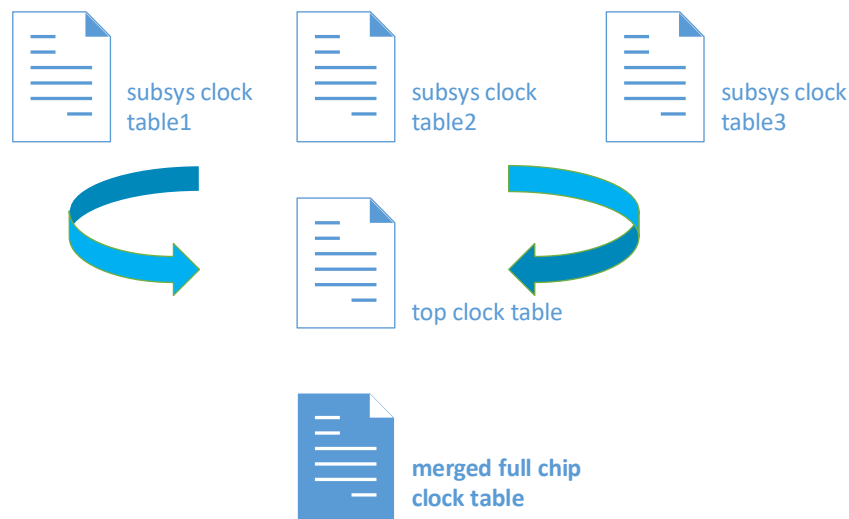


Figure 4 Clock tables' merge to full chip clock table by iClock

After that, any engineer no matter SE, DE, VE, BE or DFT would extract the consistent clock data by iClock. Another important contribution of iClock is that it serves as a centralized clock data pool and involves everything about clock.

Project	Clock Name	Clock Type	Clock Frequency	Clock Source	Divider Ratio	Waveform	Layout Level	Constraint Level
	p1la_clk	P_CLK	208		1	(1, 2, 3)	Chip	PLL
	gclk_dtcn_mux_uspc	G_CLK	208	p1la_clk	1	(1, 2, 3)	Chip	DTCM
	gclk_mcg_u_ahbp4top_o	G_CLK	52	gclk_dtcn_mux_uspc	4	(1, 2, 9)	Chip	CGU
	gclk_xbar_ahb5_slave_o	G_CLK	104	gclk_dtcn_mux_uspc	2	(1, 2, 5)	Chip	L_CGU
	pin_dbg_mclk_bus_i	G_CLK	208	gclk_dtcn_mux_uspc	1	(1, 2, 3)	DBG	CGU
	gclk_dbg_dap_p4_hclk_o	G_CLK	104	pin_dbg_mclk_bus_i	2	(1, 2, 5)	DBG	CGU
	gclk_dbg_debug_clk_ahb_slave_o	G_CLK	52	pin_dbg_mclk_bus_i	4	(1, 2, 9)	DBG	CGU

Figure 5 iClock extracted Excel form

iClock users would view or edit the clock data in any of clock table (top clock table or subsystem clock table) by Excel editor shown in Fig.5 or by iClock GUI (Graphic User Interface) shown in Fig.6. Comparing with the Excel edition style, iClock GUI edition is more convenient with advantages below:

- Reorganize disordered clocks by clock tree order with different color.
- Automatic clock consistence check when importing or exporting clock Excel tables.
- Summarizing the overall clock tree e.g. clock number, PLL number.
- Graphical presentation of clock tree to master specific clock source and its children clocks shown in Fig.7.
- Easy clock source and sequence trace. As Fig.8 shows, it would help tracing the whole clock source chain of a specific clock, and get its parent and children clocks.
- Smarter search function to give required search pattern for any clock info column.
- Detailed edition backtrack to help clock edition history for the non-text clock database.
- Clock database comparison function to assist differentiate historical versioned clock tables.
- Flexible clock data extraction to required script formats for VE, BE or DFT.

Name	Clktype	Frequency	Sourcename	Division	Waveform	Layout Scope
emptyClockItem	G_CLK	26		1	1 2 3	CGU L_CGU
p1la_clk	P_CLK	208.0		1	1 2 3	Chip PLL
gclk_dtcn_mux_uspc	G_CLK	208.0	p1la_clk	1	1 2 3	Chip DTCM
gclk_mcg_u_ahbp4top_o	G_CLK	52.0	gclk_dtcn_mux_uspc	4	1 2 9	Chip CGU
gclk_xbar_ahb5_slave_o	G_CLK	104.0	gclk_dtcn_mux_uspc	2	1 2 5	Chip L_CGU
pin_dbg_mclk_bus_i	G_CLK	208.0	gclk_dtcn_mux_uspc	1	1 2 3	DBG DBG
gclk_dbg_dap_p4_hclk_o	G_CLK	104.0	pin_dbg_mclk_bus_i	2	1 2 5	DBG DBG
gclk_dbg_debug_clk_ahb_slave_o	G_CLK	52.0	pin_dbg_mclk_bus_i	4	1 2 9	DBG DBG
gclk_mcg_u_ahbp5top_o	G_CLK	104.0	gclk_dtcn_mux_uspc	2	1 2 5	Chip CGU
gclk_xbar_ahb4_slave_o	G_CLK	52.0	gclk_dtcn_mux_uspc	4	1 2 9	Chip L_CGU
gclk_pll_A_0_2	G_CLK	104.0	p1la_clk	2	1 3 5	Chip CGU
gclk_pll_A_0_4	G_CLK	52.0	gclk_pll_A_0		1 5 9	Chip CGU
gclk_dtcn_mclk_fix1	G_CLK	104.0	gclk_pll_A_0		1 2 3	Chip DTCM
gclk_mcg_u_i2c2_o	G_CLK	52.0	gclk_dtcn_m		1 2 5	Chip CGU
gclk_per1_ahb2_usif1_o	G_CLK	104.0	gclk_dtcn_m		1 2 3	Chip L_CGU
gclk_per1_ahb1_usif1_o	G_CLK	104.0	gclk_dtcn_m		1 2 3	Chip L_CGU
gclk_mcg_u_oct_dvc_axi_o	G_CLK	104.0	gclk_dtcn_m		1 2 3	Chip CGU
gclk_pll_A_0_3	G_CLK	69.0	p1la_clk		1 3 7	Chip CGU
gclk_dtcn_mux_out_1	G_CLK	208.0	p1la_clk		1 2 3	Chip DTCM
gclk_out_1_o	G_CLK	52.0	gclk_dtcn_m		1 2 9	Chip CGU
pll_clk	P_CLK	338.0			1 2 3	Chip PLL
gclk_dtcn_mclk_lte_tdec	G_CLK	338.0	pll_clk		1 2 3	Chip DTCM
pin_mclk_lte_tdec_i	G_CLK	338.0	gclk_dtcn_m		1 2 3	LTE CGU
gclk_orx_tdec	G_CLK	338.0	pin_mclk_lte_tdec_i		1 2 3	LTE CGU
gclk_orx_tdec_ahb_orx_dfecpu0	G_CLK	113.0	gclk_orx_tdec	3	1 2 7	LTE CGU
gclk_orx_tdec_ahb_orx_dfecpu2	G_CLK	113.0	gclk_orx_tdec	3	1 2 7	LTE CGU
gclk_orx_tdec_ahb_orx_dfecpu1	G_CLK	113.0	gclk_orx_tdec	3	1 2 7	LTE CGU
gclk_orx_tdec_ahb	G_CLK	113.0	gclk_orx_tdec	3	1 2 7	LTE CGU

Figure 6 Main window of iClock GUI

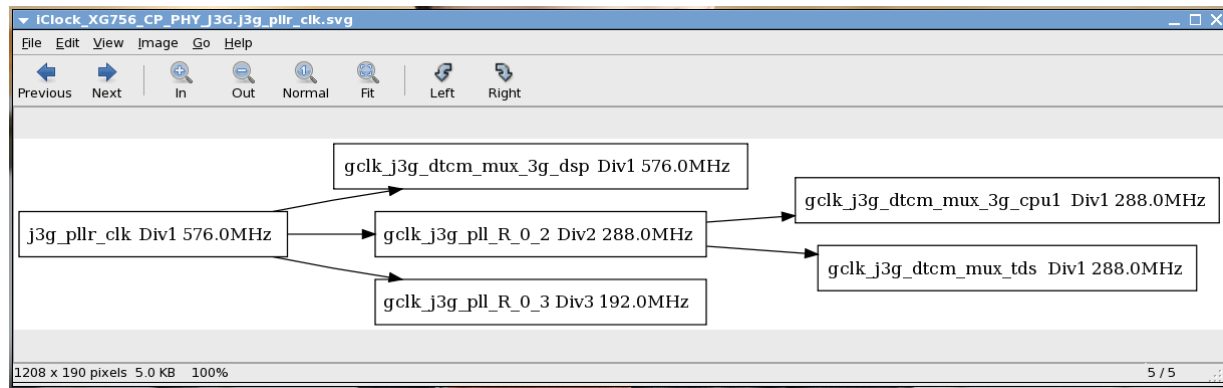


Figure 7 Clock tree graph of iClock GUI

**iClock v1.2**

File Edit Search View Compare Help

Name	Frequency	Division	Layout	Scope
▼ j3g_pll_clk	576.0	1	J3G	PLL
gclk_j3g_dtcn_mux_3g_dsp	576.0	1	J3G	DTCM
▼ gclk_j3g_pll_R_0_2	288.0	2	J3G	CGU
gclk_j3g_dtcn_mux_3g_cpu1	288.0	1	J3G	DTCM
gclk_j3g_dtcn_mux_tds	288.0	1	J3G	DTCM
gclk_j3g_pll_R_0_3	192.0	3	J3G	CGU

show Clock Infos

---

get Tree View

get Sequence View

---

focus Clock Item

generate Clock Graph

Figure 8 Clock tree window of iClock GUI

With an overview about the iClock's rich features, it would give a panorama from basic concepts to the function details. The purpose is to help understanding why the in-house tooling is designed as such an architect, and how those functions are combined to contribute the full clock steps through pre-silicon phase. Paragraph 3. would explain the main body of iClock about the clock table, and also the elements which compose a consistent database. Paragraph 4. would give some demos to illustrate how to apply the clock table to extract differently required context in pre-silicon phase. Paragraph 5. would show the GUI mode of iClock, and give basic instructions for how to operate this tooling. Paragraph 6. and 7. would be final summary and prospect about iClock's strength and features to be added in the short future.

### 3. Clock Table and Consistent Database

Before introducing the clock table, it is necessary to repeat that the iClock table is primarily extracted from SE's BlockClocks table which only involve quite limited information and just about the clock source, frequency request and relationship. However, there will be more clock related information to be appended to iClock table, which would involve all needed clock data such as shown in iClock



extracted Excel form Figure 5 but more. Regarding the paper length, it is given most commonly used clock information below with short description:

- Name: clock name and should be unique across the overall clock table.
- Type: clock type and indicate if it is a PLL root clock, a DTCM clock, a gated clock or a pin input clock.
- Frequency: clock frequency. In some scenario, the frequency would be different according to different voltages.
- Source: the clock source which generates the current clock.
- Division: the division relationship with source clock e.g. if current clock A is 200Mhz, and its source clock B is 400MHz, then the division number of A is 2.
- Waveform: it is a commonly required parameter for BE, which is used to specify the duty ratio for a clock e.g. if clock A waveform is (1 2 3), then it means the duration of high value '1' and low value '0' equals.
- Layout: used to specify which layout block the current clock belongs to. In general, there would be some independent layout blocks inside a SoC.
- DtcM: indicate if it is given from a DTCM, and there are some other DTCM related parameters used to be composed together and explain the relationship between the clock and target DTCM.
- Synchronicity: this will define the clock group, and iClock will generate the timing exception for different clock groups. All of clocks inside the same clock group will be deemed as synchrouous.
- Scan clock mapping: normally, there will be several scan clocks in chip. Try to avoid unnecessary clock crossing check between different clocks from different DTCMs. The scan clock mapping should done in a smart way to save effort to analyze those unreal timing violations during scan mode.
- Instance name: it is usually composed of a common defined block hierarchy macro with a specific clock path, which would point to the clock location.

### 3.1 Clock Table Merge

Besides those typical clock elements, there are still some elements about the specific usage. For an clear explanation, iClock introduction would be mainly around those elements given above. It would be also found that the listed clock elements are not given by SE, but defined by clock DE. Both of top level clock DE and subsystem clock DE would later merge their corresponding clock tables together. The merge effort would be implemented by iClock with the command below:

- `iclock -i a.csv -merge b.csv c.csv -o full.csv`

The example command would merge top level clock table a.csv with subsystem level clock tables b.csv and c.csv, and finally generate a synthesized overall clock table full.csv. This merge procedure is also demonstrated in Figure 4.

### 3.2 Clock Table Analysis

In general, for a single clock table (excel format) which fulfills the iClock format requirement, would be compatible with iClock edition mode, and also available to be edited in iClock. Once such a table is fed to the tooling, it would imply a data analysis procedure of source extraction and database generation. During the data source extraction, it will not only check the original data consistence, but also regulate the data into the iClock database. For the original data consistence, it will firstly give the overall clock data report which inolves:

- Detailed clock summary
- Any violated clock information

For instance, iClock is supplied with a processor subsystem clock table, and then it would report the processor subsystem's overall clock information as Figure 9. The reported messages include:

- Synchronicity clock groups
- Clock sources
- Number of each clock types
- Violation

```

ClockCheckWarning: [gclk_arc_cgu_clk_apb] division : "12" is larger than 8! Is this really what you want?
Synchronicity groups:
SYNC_MUX_ARC: 49
SYNC_DFT_TEST_CLK_38M: 1
SYNC_MUX_38M: 1
SYNC_DFT_LV_WRCCK: 1
SYNC_PLLK: 1
SYNC_DFT_LV_BISR_CLK: 1
SYNC_IN_0_1: 2
SYNC_PLLO: 4
SYNC_MUX_NOC_SGL1_DBG_SGL2_APTCK: 1
SYNC_IN_0_1_VPLL: 1
SYNC_RTC_0_1: 1
SYNC_DFT_SCAN_CLK_0: 1
SYNC_DFT_EDT_CLK: 1
7 Clocks(duty cycle==1/2):
gclk_arc_clk (1, 2, 3)
gclk_arc_dtcn_mux_arc (1, 2, 3)
gclk_arc_cgu_clk_arc (1, 2, 3)
gclk_arc_rtt_clk (1, 2, 3)
gclk_arc_pllq_0_2 (1, 3, 5)
gclk_arc_mcip_clk (1, 2, 3)
gclk_arc_pllq_0_4 (1, 5, 9)
#####
#      Clocks Summary      #
#####
-----
Primary Clock | Generated Clock | SUM
PAD | PLL | LIB | PIN | PCLK_ALL | PAD | PLL | LIB | DTCM | CGU | BLK | GCLK_ALL | TOTAL
-----
0 | 1 | 0 | 12 | 13 | 0 | 3 | 0 | 1 | 11 | 37 | 52 | 65
-----

PLL && Generated Clocks Number: 3
Primary && PIN Clocks Number: 12
Generated && DTCM Clocks Number: 1
Synchronicity Group Number: 13
Synchronicity Group Number(only appear once): 10
Synchronicity Group(SGL1) Number: 1
Synchronicity Group(SGL2) Number: 0

```

**Figure 9 Clock consistence check**

The red color marked message is used to point out which clock data might be unreasonable or inconsistent with other clocks. Therefore, it is necessary to read the warnings carefully and edit the target clock until all of warnings are clean. The report is also helpful for a visitor and master the overall clock information of a subsystem. Then the user would save the clock table again to a CSV or a excel format. It means user would both edit the clock data by Excel editor or iClock, and the suggested way is to use the iClock since it supplies lots of convenient operations which are not given by Excel editor. While editing the clock data via iClock, it would apply several common operations by right-clicking the target clock, and there would be a menu popping up as Figure 10.

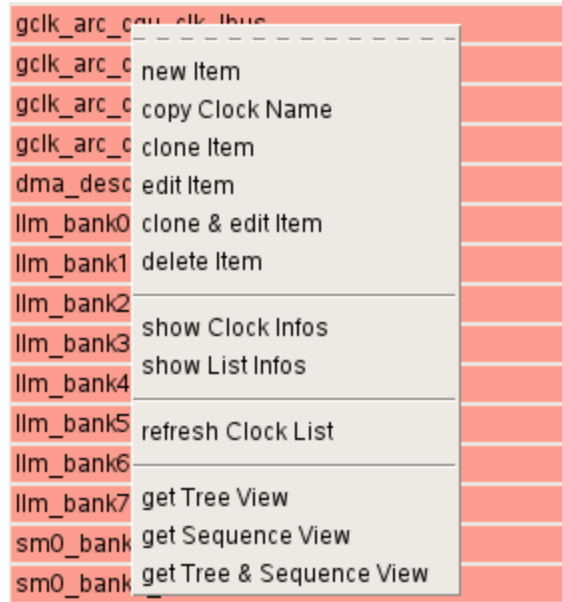


Figure 10 Clock data edition menu

When editing a clock item, a clock entry table would pop up as Figure 11. In the table, all of clock parameters would be modified, and user would submit it. The point to be noted that, there is also a clock consistence rule check while submitting the clock table, and there might be warning or violation messages given once the clock edition is not expected. The smart clock edition content would be found in 3.3

Figure 11 Clock Edition Table

As it shows, the user would easily add new, copy, clone, edit or delete a clock item, and also get clock information by the log window. Meanwhile, the user would get clock relationship from the clock tree window illustrated by Figure 8.

### 3.3 Smart Clock Edition

After the initial analysis and arrangement, all of clock information would be pushed into the formatted clock database. The database architect is not the core point in this paper, but it is valuable to know each clock is reordered with its parent clock and child clocks, which finally generate a tree data structure. Such a tree structure would make clock edition easier and more efficient to draw the clock tree diagram in the clock tree window.

Another advantage of iClock clock edition is each edition would follow several rule checks and auto adjustment such as:

- Check if the edition is valid and follows the clock consistence rules.
- If the clock edition is valid, iClock would help automating all of other possible affected clocks e.g. if source clock A is decreased from 400Mhz to 200Mhz, and then its child clocks clock B and C would be automated to be decreased from 200Mhz to 100Mhz.
- If the edition, addition or deletion of clock affected the existing clock tree structure e.g. change clock A's source clock from clock B to clock C, then user would apply the clock list refresh to reorder the clock tree.
- Once the clock tree is changed and user reorders it, iClock would refresh its internal clock structured database and rearrange the clock items. It is easy to identify the different clock groups and their contained clocks from the various color shown in Figure 12.

gclk_arc_cgu_clk_gic	G_CLK	281.6	gclk_arc_dtc_m_mux_arc
gclk_arc_pllq_0_2	G_CLK	422.4	arc_pllq_clk
gclk_arc_pllq_0_3	G_CLK	281.6	arc_pllq_clk
gclk_arc_pllq_0_4	G_CLK	211.2	arc_pllq_clk
pin_arc_32khz_clk	P_CLK	38.4	pin_upc_32khz_clk
pin_arc_38m_ana_clk	P_CLK	38.4	pin_upc_xo_38m_ana_clk
pin_arc_38m_xo_clk	P_CLK	38.4	pin_upc_xo_38m_clk
pin_arc_38m_xo_cscck_clk	P_CLK	38.4	pin_upc_xo_38m_clk
pin_arc_LV_BISR_CLK_PDG_TOP	P_CLK	38.4	pin_upc_LV_BISR_CLK
pin_arc_LV_WRCK	P_CLK	38.4	pin_upc_LV_WRCK
pin_arc_bbss_307m_clk	P_CLK	307.2	pin_upc_bbss_307m_clk
pin_arc_clk_jtag_tck	P_CLK	25.2	pin_upc_tck_clk
pin_arc_edt_clock	P_CLK	40.0	pin_upc_edt_clk
pin_arc_kernel_clk	P_CLK	38.4	pin_upc_sys_clk
pin_arc_scan_clk	P_CLK	40.0	pin_upc_scan_clk
pin_arc_test_38mhz_clk	P_CLK	38.4	pin_upc_test_38mhz_clk

Figure 12 Different clock groups with easily recognized color

### 3.4 Edition History Backtrace

It is commonly required to compare the clock table between adjacent versions. For excel format, it is too hard to do the comparison, and it is either not friendly to read the text difference with CSV format. The clock table edition comparison got satisfied in iClock, and it is easy to do the clock edition history backtrace. It is illustrated in Figure 13, once the clock table example\_subsystem.csv is compared with example\_subsystem\_ver2.csv by iClock, it would give categorized comparison messages:

- The clock property differences. It shows there is 1 clock item property difference.
- The clock objects removed. It shows there is 1 clock item removed.
- The clock objects added. It shows no new clock item is added.

With this feature, the clock edition history is easy to track, and even the clock table is managed more than one DE, each edition history would be available to understand instead of only relying on the versioned check-in comment.

```
clockContainer : [example_subsystem.csv] compare with [example_subsystem_ver2.csv]
```

```
=====
# clock property differences BEGIN
=====
#Diff clock object NO.1
[gclk_arc_mcip_clk clock] diff with [gclk_arc_mcip_clk clock]
<division>                                1 :                                2
=====

#=====
# clock property differences END
#=====
# clock objects removed in example_subsystem_ver2.csv BEGIN
#=====
#Removed clock object NO.1
clock name = pin_arc_32khz_clk
clock source name = pin_upc_32khz_clk
clock instance name = ${PHY_UPC_ARC}clk_cgu_32khz_i
frequency = 38.4
division =
waveform = ()
scope = PIN
layout = PHY_UPC_ARC
clock type = P_CLK
synchronicity = SYNC_RTC_0_1
clock dtcm =
clock dtcm clock number =
clock dtcm bit position =
delay scan clock =
delay scan clock frequency =
delay scan divider name =
delay scan divider ratio =
delay scan clock mapping =
test clock control =
testClockControlCellName =
=====

#=====
# clock objects removed in example_subsystem_ver2.csv END
#=====
# clock objects added in example_subsystem_ver2.csv BEGIN
#=====
# clock objects added in example_subsystem_ver2.csv END
#=====
```

Figure 13 Clock Table Comparison

## 4. Clock Extraction for Different Procedure

The magic of iClock consistent database is that it would apply the centralized data to generate

different output for kinds of procedure. There are several typical applications with the database in pre-silicon phase, and they involve:

- Script for Spyglass CDC (Clock-crossing Domain Check) for DE
- Clock frequency requirement for VE
- Script for STA (Static Timing Analysis) for BE

It would give examples how to use iClock to generate the listed outputs in the following sections. iClock supplies both batch mode and GUI mode for user to generate the scripts or header files, and this paper only illustrates how to use the batch mode to make those outputs.

#### 4.1 Spyglass CDC Script for DE

Another advantage of iClock is that the tooling development considered the distributed requirements from different function groups, and it is proposed to only maintain the centralized database. In other words, once iClock offers easy-use APIs (Application Programming Interface) of consistent database, every user would write Python script to be embedded by iClock, and produce the expected script format. The Spyglass CDC script is developed by the DE group independently, and maintained by them too. Such cooperation working mode makes iClock to be extensively utilized by all of function groups in pre-silicon phase, and also releases the in-house tooling developer's effort for requirement response.

In iClock batch mode, user would type in such a command format, and a script for Spyglass CDC would be generated:

```
➤ iclock -i a.csv -cdc -o a.sgcd
```

It is known by experienced CDC reviewer that the most important configuration file is the clock information. For instance, the synchronicity should be set correctly, otherwise, some inappropriate clock crossing domain design would escape from the Spyglass CDC check. It is given a short piece of statements from the generated CDC script below:

```
➤ clock -domain SYNC_MUX_ARC -tag gclk_arc_clk -period 1.184 -edge {0 1} -name "${PHY_UPC_ARC}inst_arc.clk"
➤ clock -domain SYNC_MUX_ARC -tag llm_bank1_gclk -period 4.735 -edge {0 1} -name "${PHY_UPC_ARC}inst_arc.ihs_cluster_top.clk_o"
```

The CDC script automation further decreases the possibility of manual operation, and ensures the data extraction from the iClock centralized database. Comparing with manual interpretation from the design file or clock excel table previously, this process significantly reduced the manual operation risk, and makes Spyglass figuring out the asynchronous places as much as possible with a correct clock configuration.

#### 4.2 Frequency Requirements for VE

Through the pre-silicon verification phase, clock frequency check is a basic requirement which would guarantee the testing system is under a correct clock configuration. Otherwise, some asynchronous places or timing violations would escape in RTL simulation or gate-level simulation. It is used to manually extract each module's or subsystem's clock info and also deployed in stimulus sequence and checker of environment. It is also the similar as 4.2 shows manual work would possibly introduce new unexpected fault such as inappropriate clock set or check. Therefore, the VE group also starts to

work with iClock, and customizes the Python script which was friendly plugged into iClock database, and then generate clock set sequence and check statements with C or UVM.

For instance, here it is given the clock check statements with C and UVM format. The two short pieces demonstrated the small effort dedicated by VE with iClock is much valuable, and the automation flow would generate all expected clock frequency check statements. User would apply any of them in their test case from the centralized clock check repository instead of preparing it themselves. Besides those convenient element methods and macros, VE would also customize other clock related methods with the iClock database.

```
➤ #define CLK_ARC_A_FREQ_200_4_CHECK(IS_OK)
  bcompare_clock_frequency("phy_upc_arc.inst_arc.clk_a", 200400,
  IS_OK) // clock frequency check macro with C
➤ #define CLK_ARC_A_FREQ_GET(FREQ)
  bget_clock_frequency("phy_upc_arc.inst_arc.clk_a", FREQ)
  // clock frequency get macro with C
➤ `define CLK_ARC_A_FREQ_200_4_CHECK(IS_OK)
  bcompare_clock_frequency("phy_upc_arc.inst_arc.clk_a", 200400,
  IS_OK) // clock frequency check macro with UVM
➤ `define CLK_ARC_A_FREQ_GET(FREQ)
  bget_clock_frequency("phy_upc_arc.inst_arc.clk_a", FREQ)
  // clock frequency get macro with UVM
```

### 4.3 STA Script for BE

Another important usage of iClock database is to serve the precise clock group set for BE's Design Compiler STA and other procedure preparation. As it is explained previously, the consistent iClock database involves all of needed information which is also a treasure for BE. It is noted the script for backend procedure would be more complex and careful because every clock should be not only precise for itself but also compatible with its parent clock and child clocks. Especially, if there is some missing asynchronous information which is not extracted between several clocks, it would be too much risky for those undetected area. The project practices also illustrated that, in the initial one or two projects, both of iClock database and BE extraction script needed to be reviewed repeatedly and the generated script has been also examined until the tape-out silicon proved the stability of the tooling and the flow. Here it is also given some short pieces of generated BE script for STA and other procedure:

```
=====
# Variable Definition
=====
set clk_src(dma_descr_gclk) "gclk_arc_clk"
...
=====
# Pin Definition
=====
set pin_src(dma_descr_gclk) "${PHY_UPC_ARC}inst_arc/rccu1/clkout"
...
=====
# Clock Definition
=====
create_gclk -name dma_descr_gclk -mclk $clk_src(dma_descr_gclk)
```

```
-edge { 1 2 5} -pin $pin_src(dma_descr_gclk) -clkgrp SYNC_MUX_ARC
...
```

Besides the code listed above, a legacy SoC concatenated clock database would generate over hundred pieced of script for backend procedure, but they are quite well organized and easy to maintain once the any clock item of the thousand clocks is updated. No more manual definition existed in BE clock exportation flow, and this no doubt increases the tape-off signoff confidence.

## 5. GUI Mode for Feature Enhancement

Besides the main window shown in Figure 6, there are also several utility window for specific usage. It was architected not only for clock DE but also for VE and BE. Thus, it is reasonable to supply several practical features to search or visualize one or several clocks. This paragraph would introduce other windows and their corresponding functions. The overall window layout is shown in Figure 14.

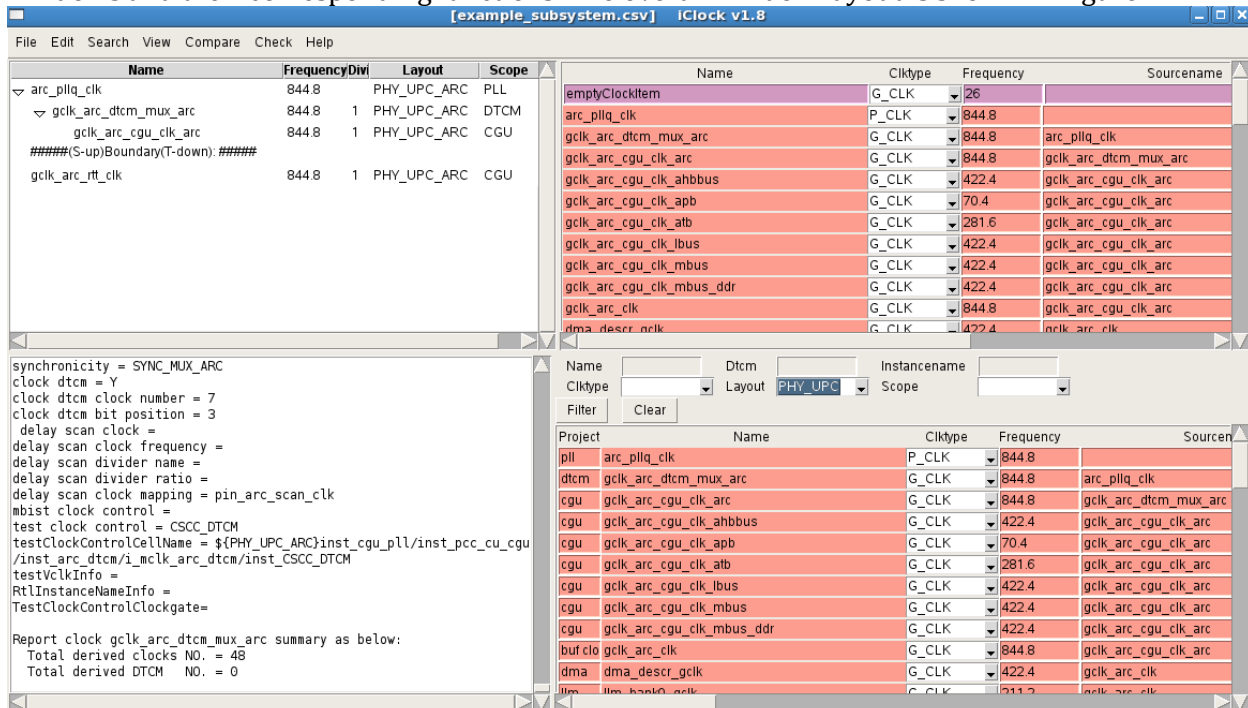


Figure 14 iClock Overall Layout

It is same with other commercial EDA toolings there is a menu bar, and also several embedded windows in iClock. Major functions inside the menu bar have been already introduced, and the following is to give an additional brief demonstration for other windows:

- Clock tree window
- Clock graph window
- Clock search window

### 5.1 Clock Tree Window

It is commonly asked to trace the parent clock or several child clocks, which would help understanding the clock relationship. The user would right click any clock item and select “get tree view” or “get sequence view” as shown in Figure 10. It is also available to right click the clock item in the clock tree window to get more clock information in the log window, get parent clock or child clocks, or generate a visualized clock graph to be introduced in 5.2 .



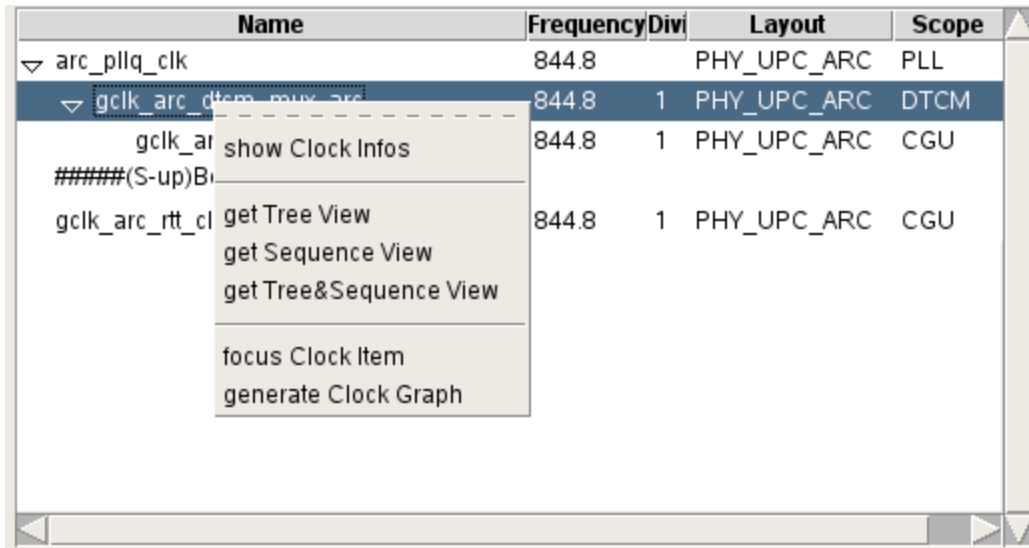


Figure 15 Clock tree window usage

## 5.2 Clock Graph Window

While it is to trace a big clock group, the clock tree window is not the only way to understand the full picture. With the same clock database, iClock also implements another kind of window, clock graph window. In general, the graph window is not present in the GUI layout, but user would call it via main window or clock tree window while selecting an option “generate clock graph” with right clicking on a clock item. Figure 7 gives a brief clock structure which involves a root clock and several child clocks, and the clock graph mode is more appropriate to master the overall clock tree.

## 5.3 Clock Search Window

With extensive iClock application to all of function groups, the user experience demanded an independent window which would show the interesting clocks. Sometimes, the user would search a clock, a DTCM clock tree or overall clocks of a specific subsystem. The smart search feature is different with general text search because it supplies an enhanced regular pattern search mode and a different category match mode. For instance, as Figure 16 shows, user would type in interesting key word in the “name” entry or other entries, or directly select a clock type, a layout or a scope to filter the target clocks.

Name		Dtcm		Instancename	
Clktype		Layout	PHY_UPC	Scope	
Filter	Clear				
Project	Name	Clktype	Frequency	Source	
pll	arc_pllq_clk	P_CLK	844.8		
dtcm	gclk_arc_dtcn_mux_arc	G_CLK	844.8	arc_pllq_clk	
cgu	gclk_arc_cgu_clk_arc	G_CLK	844.8	gclk_arc_dtcn_mux_arc	
cgu	gclk_arc_cgu_clk_ahbbus	G_CLK	422.4	gclk_arc_cgu_clk_arc	
cgu	gclk_arc_cgu_clk_apb	G_CLK	70.4	gclk_arc_cgu_clk_arc	
cgu	gclk_arc_cgu_clk_atb	G_CLK	281.6	gclk_arc_cgu_clk_arc	
cgu	gclk_arc_cgu_clk_lbus	G_CLK	422.4	gclk_arc_cgu_clk_arc	
cgu	gclk_arc_cgu_clk_mbus	G_CLK	422.4	gclk_arc_cgu_clk_arc	
cgu	gclk_arc_cgu_clk_mbus_ddr	G_CLK	422.4	gclk_arc_cgu_clk_arc	
buf clo	gclk_arc_clk	G_CLK	844.8	gclk_arc_cgu_clk_arc	
dma	dma_descr_gclk	G_CLK	422.4	gclk_arc_clk	
llm	llm_bank0_gclk	G_CLK	211.2	gclk_arc_clk	
llm	llm_bank1_gclk	G_CLK	211.2	gclk_arc_clk	

Figure 16 Smart clock search

Once the search result is shown in the clock search window, user would continue the same operation as it does in the main window. With such an additional clock info window, it serves a good clock data reference.

## 6. Summary

iClock is an in-house tooling initiated to solve the clock consistence through pre-silicon, and now it also started to serve post-silicon test and driver composition. The author and his involved big SoC team used to taste the pain for the fussy clock storage and extraction procedure, and figured out the consistent clock database to finally offer the integral solution. The centralized clock data maintenance and smart clock rule check decreased the risk of several manual clock operation loops, and also change the function groups' cooperation mode.

Previously, DE, VE and BE would be all responsible to their extracted data info, but now with iClock only clock DE would do the clock tree design. VE and BE would just master the iClock application, and customize their script for the specific procedure. iClock was kicked off in 2015 December, and released its version 1.0 in 2016 March. Now the latest version is 1.8 with quite several new features introduced. The in-house tooling has gone through totally three billion gates SoC projects, and now it has reached the stability period. Only a little project specific configuration is required for a project start, and then iClock would keep precise serving from the project start to the project end.

From the Synopsys tooling application view, iClock is a well customized pre-stage solution which keeps the clock data accuracy as a solid base to feed the Spyglass and Design Compiler. With the iClock contribution, both of process efficiency and quality got improved for DE, VE and BE. This tooling removes the redundant and error-prone operations, which makes DE, VE and BE work more closely around iClock.

## 7. Prospect

iClock currently has stepped to a stable phase, and only proceeded some improvement. While the paper is under preparation, the internal methodology team initiated another large program which will collect power and reset information. It is estimated once the three key types of system information have been finally synthesized as a solid database, it would play a more important role. For example, the power, reset and clock combination would enhance the system scenario coverage, and it would be more precise to check if the power sequence is as expected.

iClock not only serves function group but also DFT (Design for Testability) group. There have been already some clock information for DFT clock set and post-process script generation. This tooling development is further enlarging the test clock info scope in the running project. iClock has entered post-silicon test and driver team vision, they also started to make consistent clock set and check header files.

Therefore, the general orientation for iClock development is to extend both of application depth and width, and moreover, to push forwards the core consistent database concept for the overall SoC development procedure and deep cooperating with Synopsys toolings.

## 8. Acknowledgement

This wonderful tooling has went through hard development days, and got kinds of help from concept, development, application, maintenance and feedback. Finally, it succeeded and won lots of internal users with silicon proven projects. It is appreciated those colleagues involved in iClock growth from a baby to an adult. The thanks list involves: QU Fuyang, YU Fuzhen, ZHANG Hua, WANG Bo, WANG Dongrui, ZHAO Wenfeng, ZHENG, Xiaoliang and SHEN, Xiaofeng.

## 9. References

- [1] Python standard library website <https://docs.python.org/3/library/index.html>
- [2] Tkinter library website <http://infohost.nmt.edu/tcc/help/pubs/tkinter/web/index.html>
- [3] Graphviz library website <http://www.graphviz.org>