Building a dynamic GUI, configurable at runtime by backend tool.

Manu Goel(manu_goel@mentor.com), Mohit Goel(mohit_goel@mentor.com)
Mentor Graphics Corporation

Abstract:

Providing a GUI for a tool which has hundreds of possible and completely unrelated options is a challenging task. The task becomes more complex if more options can be added in future and the aim is that those should be supported in the GUI seamlessly. A dynamic GUI, which can be configured at runtime by the backend server, can serve the purpose nicely. In this paper we talk about such a GUI which can be scaled, tweaked, modified by the backend tool without any change needed on the GUI side.

Dynamic GUI:

What is a Dynamic GUI (Roy, 2008) and how can it solve certain type of problems in GUI.

Dynamic GUIs can handle the cases, where there is a lot of information available to GUI, but only a very small set of that information is applicable to user at any point of time. So if GUI stores and manages all the information, then it would become complex for GUI to manage all the data. GUI also needs to add all the intelligence to cater to users requirements in a logical manner.

Further, this is not a scalable solution. In order to serve any future development, changes on the GUI side are must, without that, the new information will not be available to user. If there is any change in the existing functionality, then this is dependent on GUI as well to make the corresponding changes before user can actually use this.
This becomes more and more complex, if such changes are frequent. There is an additional risk of GUI becoming out of sync with the backend as well. This also needs a team, which will be constantly maintaining the GUI and will keep on making necessary changes to keep in sync with backend.

What is the solution? Solution is to build a Dynamic GUI, which can be controlled and configure by the backend.

To achieve this let’s divide the GUI into two parts. One, the GUI frontend, where all the information is displayed to user, and the other is backend, where actual changes take place and new functionality is added. Whenever there is a change in the existing functionality, or any new functionality is added, it is the backend which is making those changes. So backend is already aware of the change or what is the new functionality.

We want to keep GUI transparent of such changes. Idea is, to keep the GUI very thin. it should not store any of the information statically. Whenever GUI needs to display any information to user, it fetches the information from the backend and whatever user gives as input, passes that to backend. Backend has all the knowledge to interpret the inputs given by the user and act accordingly.

Based on user's input, backend can now control what is the relevant information to user and show only that. So the tool need to have a very well defined interface between frontend GUI and the backend tool to pass the information back and forth.

If the information to be passed from backend to frontend and back, is big in nature, then we need to define a syntax to pass such information, so that both the sides are saved from parsing the information and can use it easily.

Through this mechanism, backend has all the control of what information and in what format any of the information is to be displayed to user and it can change it dynamically based on user's input.

Since GUI is not storing any information statically and is solely relying on backend for all the information, if backend makes any changes in the existing functionality, or adds any new functionality to the existing tool, then that will seamlessly be supported by the GUI.

This also removes the need of a team to manage and maintain GUI because once the GUI and well defined interfaces are developed; no further changes may be required on the GUI side, unless there is a change in any of the API.

We will now talk about a case, where we developed such a model to serve a complex problem we had at Mentor graphics. We will talk in detail about the interface and how the information will flow from GUI to backend and back to GUI to achieve the functionality needed,

Case Study:

We will talk about a GUI, which is developed to be used to configure various verification IPs (Mentor Verification IP, 2014) (Hereon referred as VIPs) developed by DVT division of Mentor Graphics. The GUI also assist user in connecting the VIP and the DUT to enable user to use it directly with their next level tool.

These VIPs (Shah, 2011) are standard protocols, which user can plug in their design to verify certain aspects of their design, for functional verification as well as to find out the coverage. There are a large number of such VIPs and new VIPs keep on getting added to the existing pool.

In order to use these VIPs in the design, user first need to configure the VIP of interest and then connect it to the Design Under Test(DUT).

- Configuring the VIP : These VIPs are generic in nature, i.e. they can be used with any DUT which needs to test the functionality supported by these VIPs. So, to use these VIPs based on the user design, user need to configure the VIP according to their design’s need. The configuration process expects user to set a large number of options (in some cases it may run in to hundreds). Without a GUI, user needs to know all the options which are required to be set, their possible values and the functionality as well. User needs to write all of these options and their values in a file, which will be supplied to the tool while compiling the design with these VIPs.

- Connecting the VIP : These VIPs have a fixed interface, through which the DUT will interact with the VIP. The interface is a set of pins, through which user need to connect his design. The number of such pins is typically very large, in most of the cases there would be more than 50 pins. Again, without a GUI, user needs to know the name of these pins and then create the top level design unit connecting the VIP and DUT manually. GUI automatically connects the VIP with the DUT as expected by the VIP, which user can modify based on the design.

Configuring and connecting these VIPs is a tedious and complex job, and there is a high risk that user may miss out setting some of the options, or may pass incorrect values to these options. Further, user may not be aware of the functionality of each of these options and their possible values, so user needs to go through the user manual of these VIPs in detail and may have to do a lot of back and forth to find out the details of each of these options.

Same complexity is applicable for connecting the VIP and DUT as well. For this as well, user needs to know all the pins and their details, and connect those with the correct pin of his design.

Both of these tasks are error prone, and in case of any mistake, it is very difficult to find out about what went wrong.

So the need was a GUI, which can cater following requirements –

- Show the list of all supported VIPs
- Assist in configuring the VIP
- Show all configuration options in a user understandable format
- Provide the functionality of each such option
- Provide the possible values any option can take
- Show the available pins in the VIP
- Assist in connecting the VIP with the DUT
- Generate the configuration and connection file, which can be directly used.
- Most importantly, GUI should be scalable, i.e. it should support the newly developed VIPs without much or any effort at all on GUI side.
- Support multiple VIPs to be usable in a single DUT

Complexity:

Providing a GUI for such a requirement was a complex task, because all these different VIPs have different set of options. Developing a static GUI for each of these VIPs was not a good idea, both from development as well as maintenance prospect. Further, this solution was not scalable as well to support newer VIPs in future. A static GUI, showing all the options for all the VIPs was also not a possible solution because in that case it would be showing too many options which are not related to the VIP, the user is interested in. So the need was to build a single GUI which can accommodate all type of VIPs, while showing only the relevant information for any user at all the times and can be scaled to support the newer VIPs with a very minimal effort.

Solution:

We came up with an idea of building a dynamic GUI, which starts with no information and is configured by the backend process at run time based on user inputs. It kind of, starts with an empty canvas and then builds upon the GUI, based on user input. The backend tool (VIP container) has all the knowledge and information about all the supported VIPs, their configuration options, their available pin connections etc.. So when the GUI comes up, it starts fetching the information from backend as and when required. The advantage of this approach is, GUI does not need to store any information statically. It completely relies on the information supplied by the backend. Backend can configure the GUI at runtime based on users input.

This also makes the GUI scalable to support any VIP which is supported in future. The backend is also free to modify any of the options without any need for any change on the GUI side. At the same time, backend also does not need to worry how GUI is going to show the information it is providing to GUI. Only thing it needs to worry is, provide right set of information for any option, and rest would be taken care of by GUI.

In order to fetch the information from backend, we wanted to have well defined interface, so that the information can be fetched efficiently with all the required details and ensures that both backend and GUI can work seamlessly without bothering about the functionality of the other side.
We defined a syntax in which the information will be exchanged between GUI and backend.

First set of information needed is the list of VIPs which are supported by the tool. This is the list of VIPs like Ethernet, PCIs, USB etc., and then they can have sub-category like USB2, USB3 etc., they again can have sub-sub categories. So the list of VIPs is like a hierarchy tree, where, it has parent VIP, then children VIPs and so on. Some of these VIPs may just be symbolic to contain their children and may not be selectable. So the information returned to GUI will be a recursive list of list of the supported VIPs. GUI will parse this information and populate the initial view showing the list of available VIPs to user. The interaction between GUI and backend will look like –

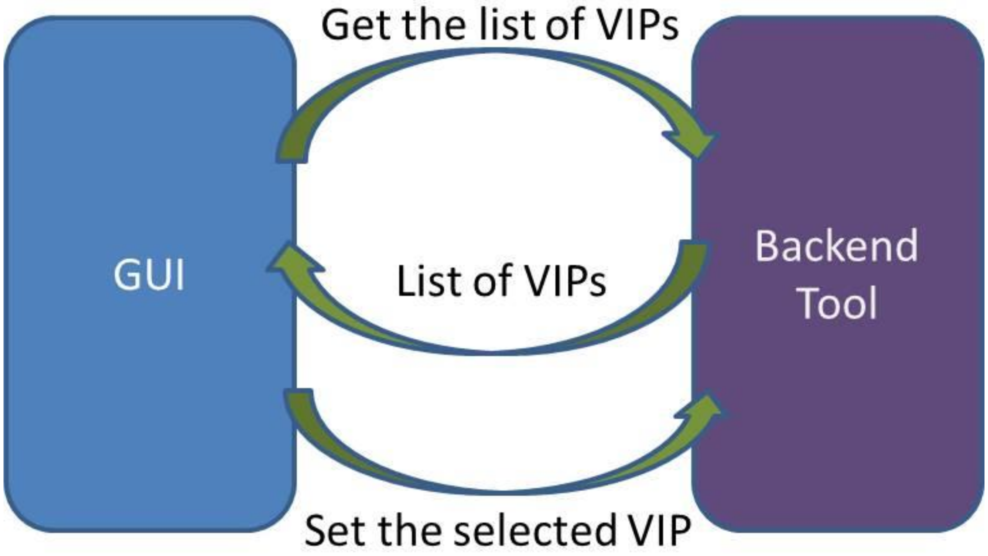


Figure 1: Fetch the list of VIPs

The initial view before user makes any selection will look as shown in the next figure.

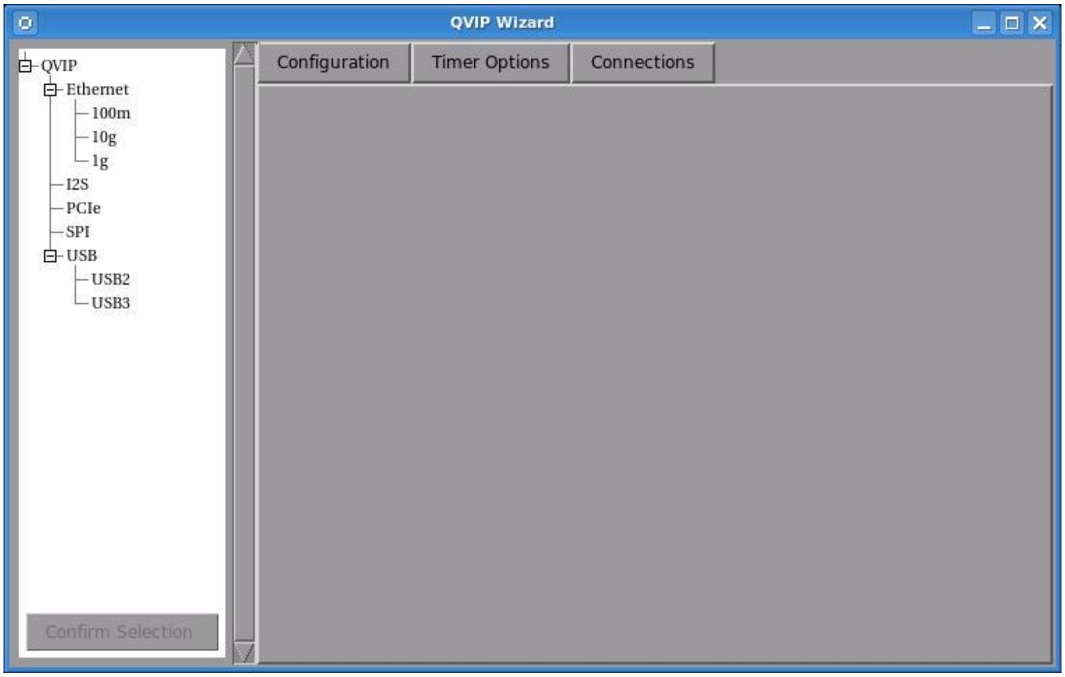


Figure 2: Initial View of GUI

Now, once GUI is up and user can see the complete set of supported VIPs, user can select the VIP user is interested in. For using the VIP, top level details are categorized in three different categories, viz, configuration options, timer options and connectivity. The backend tool can decide to further sub categorize these configuration and timer options. Typically the number of these options is large and can run into hundreds, so these are logically divided in sub categories to be shown in different tabs to make it easier for user to manage these options.

Once user confirms the VIP user is interested in, GUI needs to know what all different categories of configuration and timer options are there for that particular VIP. This ensures that backend can have different set of tabs for different VIPs.

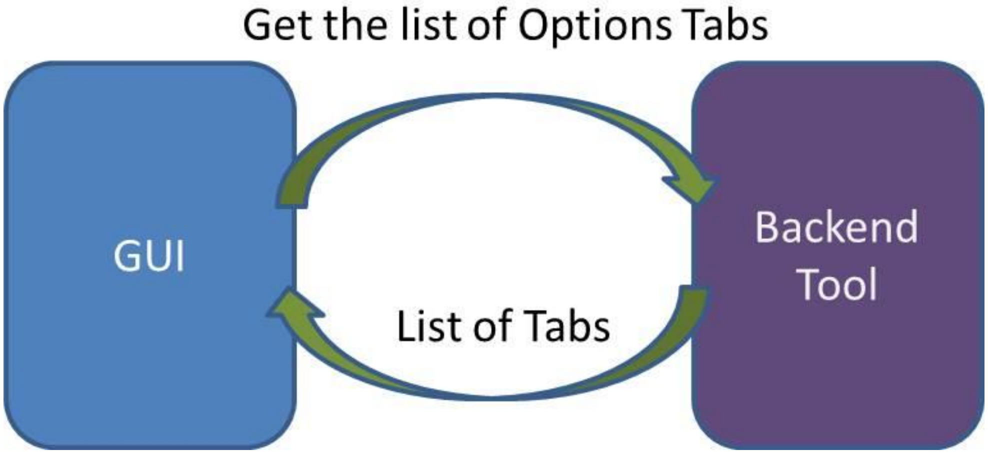


Figure 3: Fetch the list of options tabs

The list of tabs is actually a recursive list, so these configuration and timer options can be sub-divided in different categories, which can further be sub-divided.

Now, GUI has all the categories of these options, GUI need to find out the details of options in each of these categories.

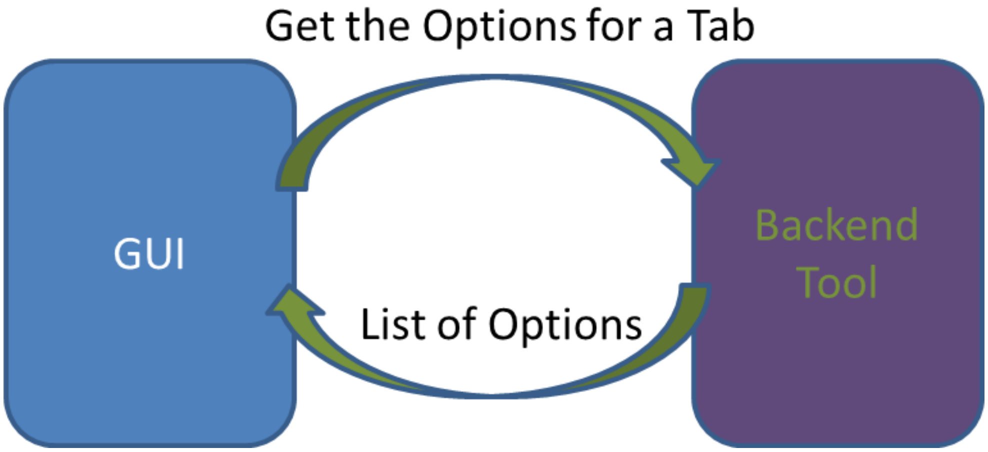
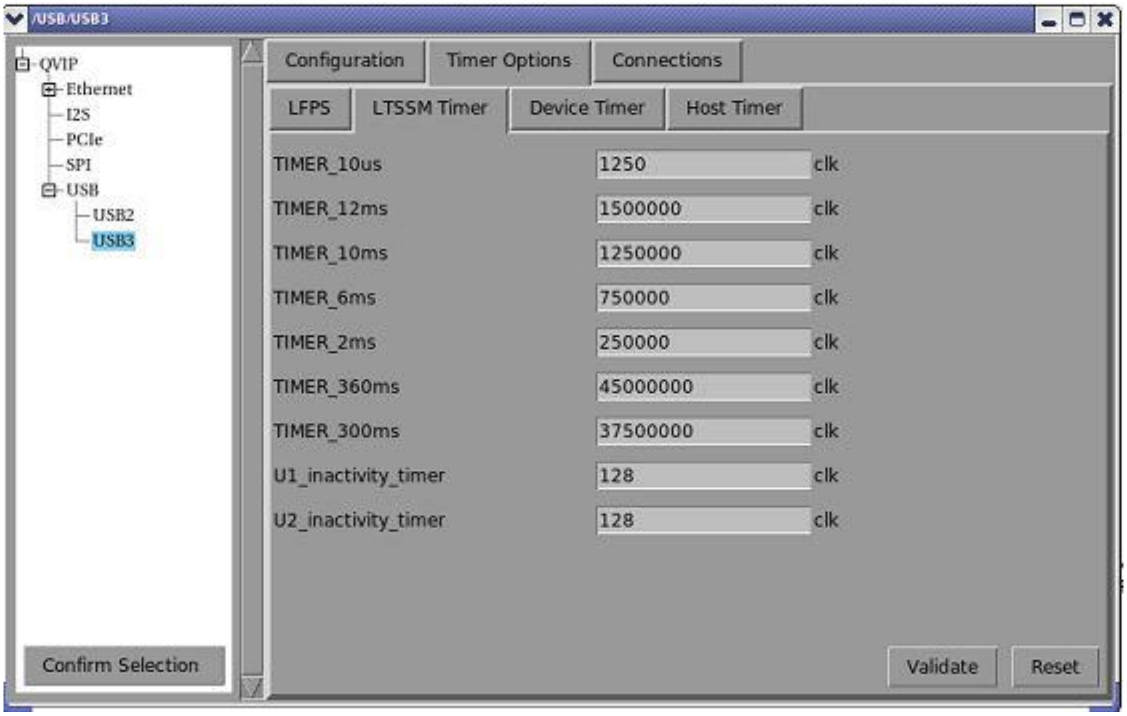


Figure 4 : Fetch the options details for each tab

This operation fetches a lot of information required to populate these options tabs. It has a list of all the options to be shown in each of these tabs along with all other related information for each option. The details associated for each option are –

- Name of the option
- Type of the option, like Entry Box, Combo box etc
- Possible values or type of values the option can take
- Default Value(If any)
- Tool tip
- Suffix
- If value change impacts next option's possible value set.

The above two steps are the core of this GUI. Here, backend is controlling and configuring the GUI. Based on users input, it can change the options dynamically. It can control how an option should be presented to user, what all possible values the option can take.



TIMER_10us	1250	clk
TIMER_12ms	1500000	clk
TIMER_10ms	1250000	clk
TIMER_6ms	750000	clk
TIMER_2ms	250000	clk
TIMER_360ms	45000000	clk
TIMER_300ms	37500000	clk
U1_inactivity_timer	128	clk
U2_inactivity_timer	128	clk

Figure 5 : Options Tab

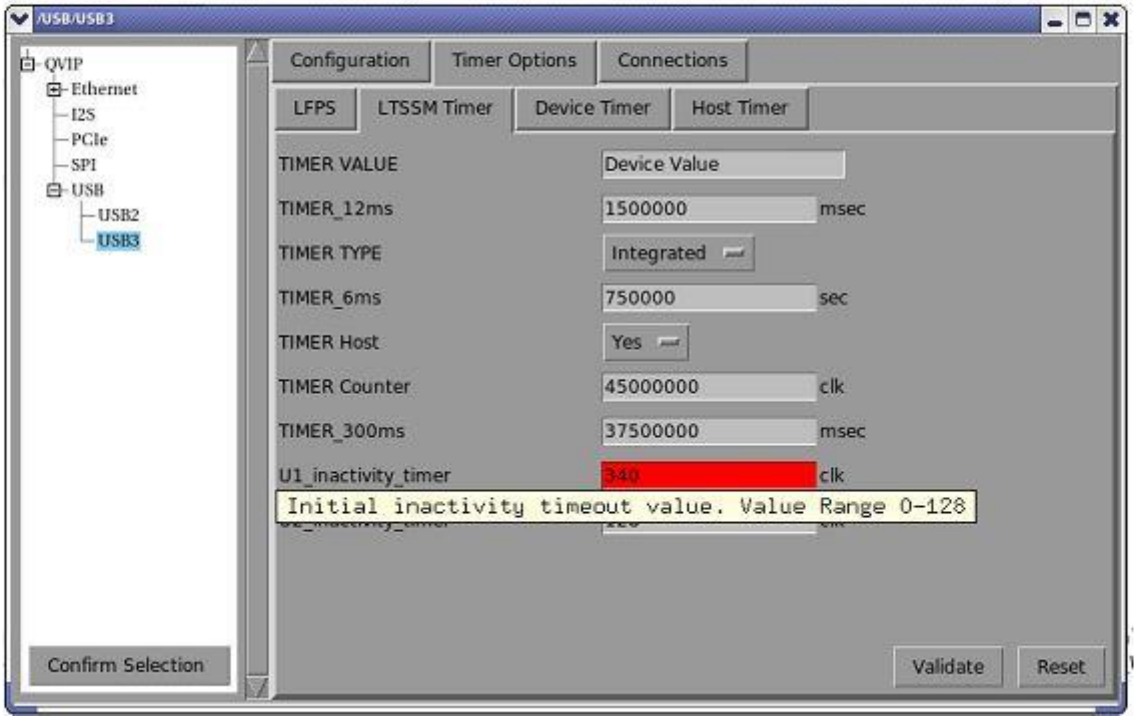
The details of these options are stored in an associative array by the backend in the format as expected by the GUI and is passed to the GUI in the form of a recursive list. Upon receiving the information, GUI also stores it in an associative array with option name as the key. Backend also stores the information with the same key making it easier to store and parse the information supplied back by GUI.

Based on these details, GUI constructs all the required panes with all the options and their other details.

Now user has all the information of interest available to him. User knows what all options are required to be set to use the VIP of interest, can see possible set of values, their types, default value, functionality(through tool tip) of each option etc. This saves user from making a lot of mistakes, like missing out certain options, setting wrong values to any option etc. It also saves user a lot of look ups to user manual to understand the functionality of various options.

If any option has set the flag indicating GUI that its value change will impact the next options possible value set, then as soon as user makes any changes in that option, the possible value set for the next option is fetched from the backend.

After filling up the details of various options, user can run a check to validate the options for correctness. This helps in ensuring that at every stage user has provided right set of information and makes it easier to fix the errors. Without the GUI, user had no way to validate the data user has supplied in, and was very difficult to identify the incorrect entries.
GUI will highlight all user errors whenever user runs the validation step.



TIMER VALUE	Device Value	
TIMER_12ms	1500000	msec
TIMER TYPE	Integrated	
TIMER_6ms	750000	sec
TIMER Host	Yes	
TIMER Counter	45000000	clk
TIMER_300ms	37500000	msec
U1_inactivity_timer	340	clk

Initial inactivity timeout value. Value Range 0-128

Figure 6 : Highlight an invalid entry

Now with all the options are set and validated, user is ready to connect the DUT with the VIP. For this GUI needs to know all information needed to create these connections.

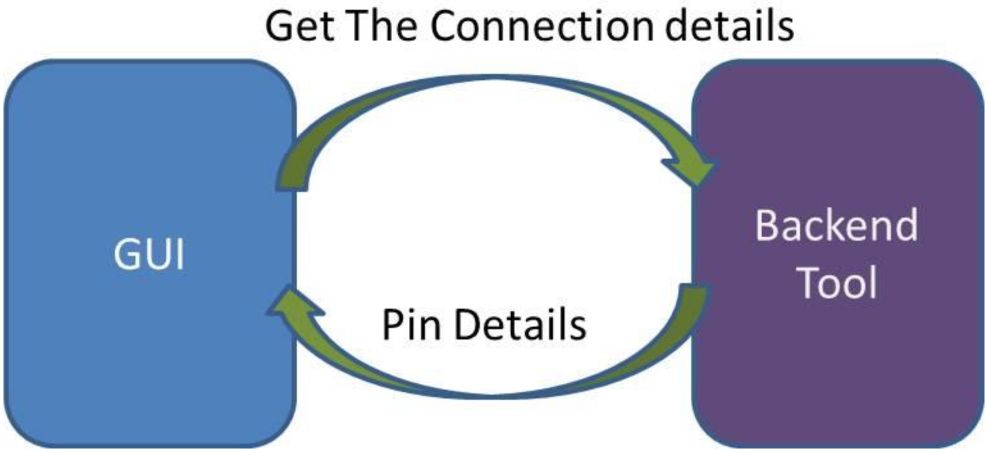


Figure 7 : Fetch connection details

The connection details are a set of pints, through which the VIP will interact with the DUT. The information supplied for each of the pin includes –

- Pin Name
- Width
- Direction
- Can the pin be left unconnected
- Default expected name on the DUT side

Now GUI creates the connections pane, where user is shown all the connections between VIP and DUT. The connections are done based on the default expected names as provided by the VIP. However, user can edit the connections for changing the pin name, keeping any pin unconnected, changing the DUT name etc.

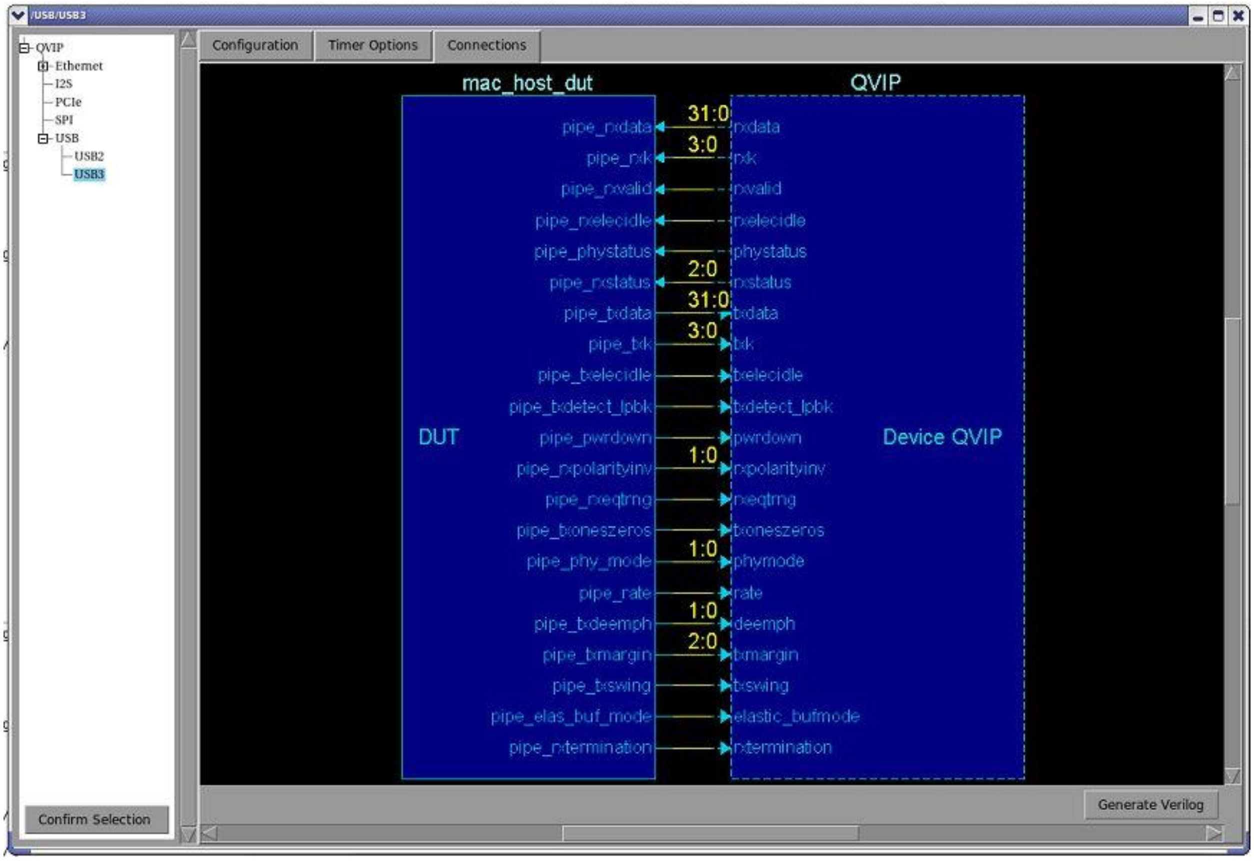


Figure 8 : Connections pane

To edit the pin name or leaving it unconnected, user simply needs to double click on the pin, and then make the necessary changes to that pin. Once user is done with editing the connections pane, now user is set to generate the Verilog files with setting for all the required configuration options as well as to connect the DUT and the VIP of interest.

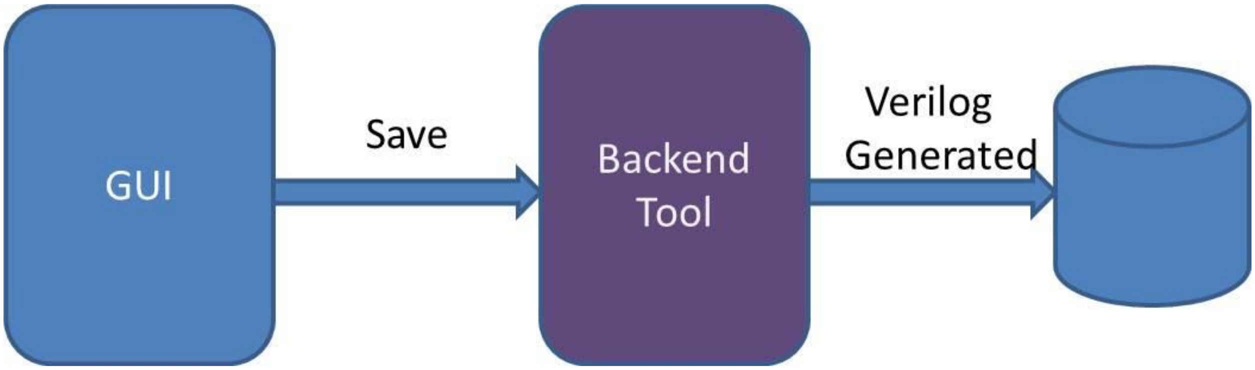


Figure 9 : Generate Verilog

Before generating the Verilog, all the options will be validated for the correctness again by the GUI and if there is any error, then that would be highlighted so that user can fix the error. This ensures that user always have good set of options saved and does not waste any time in debugging any issue because of incorrect setting of any option. This operation will generate the Verilog files. These files can directly be plugged in to the tool which will be compiling the VIP and DUT, which user can further use to run the simulation.

If user wants to add another VIP to the same DUT, then user can add another VIP before creating the connections. All options for the additional VIP will be shown to user and user needs to set them accordingly. When user moves to connection pane, it will show the DUT and all the VIPs which user has added in to use. The default connections will be done accordingly. Once user generates the Verilog files, one file for each VIP will be generated, which will contain all the options and a single file will have the details of DUT connecting to all the VIPs.

Conclusion:

With such a GUI, which is mostly controlled by backend and changes dynamically based on users input, the task which would have taken a few days can be done in less than an hour. Furthermore, user is saved from all the pain of going back and forth to user manual to understand the functionality and other details of all the options. It also saves him from making mistakes saving his precious time in debugging the issues due to incorrect option settings.

Further, on the development side, this GUI is completely scalable. It will automatically support any VIP which is added in future. Whenever the tool supports a new VIP, all the backend team needs, add the details of this VIP to the backend database, and GUI will seamlessly support it. This solution ensures that at all times, user is only seeing the information which is of use to him.

Backend is also free to make any change in any of the option in future, without any need of any change on the GUI side.

In the current implementation, the backend stores the database in associative array, and the interface is also Tcl based, however, the backend is free to choose the language in such a implementation and they can have their own data structure, as long as they supply the data in the required format to GUI.

Bibliography

(Shah, 2011) Shah M (2011, 01 11) http://www.semiwiki.com/forum/showwiki.php?title=PerfectVIPs:Verification+IP+Wiki Retrieved 10 21, 2014 http://www.semiwiki.com/forum/showwiki.php?title=PerfectVIPs:Verification+IP+Wiki

(Mentor Verification IP, 2014)
http://www.mentor.com/products/fv/verification-ip **Retrieved 10 21, 2014**
http://www.mentor.com/products/fv/verification-ip

(Roy, **2008**)
Roy Van P (2008, 17 07)
http://www.uclouvain.be/en-43648.html **Retrieved 10 21, 2014**
http://www.uclouvain.be/en-43648.html