Novel GUI Based UVM Test Bench Template Builder

Vignesh Manoharan
Aeva Inc.
555 Ellis St. Mountain View, CA 94043

2022 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES VIRTUAL FEBRUARY 28 - MARCH 3, 2022

Abstract

Adoption rate of Universal Verification Methodology (UVM) is increasing day by day across industry and the need for building new Verification Intellectual Property (VIP) or testbench is in great demand. Writing effective and structured UVM testbench from scratch is cumbersome most of the time and following a standard structure with provision for better re-usability across projects is also challenging. What if the time taken for initial development cycle is reduced to minutes instead of days with the help of a Graphic User Interface (GUI) to build the verification component templates? This poster presents an overview about the GUI interface used to develop the individual UVM components or the entire VIP templates loaded with features to customize and configure as per the user requirements.

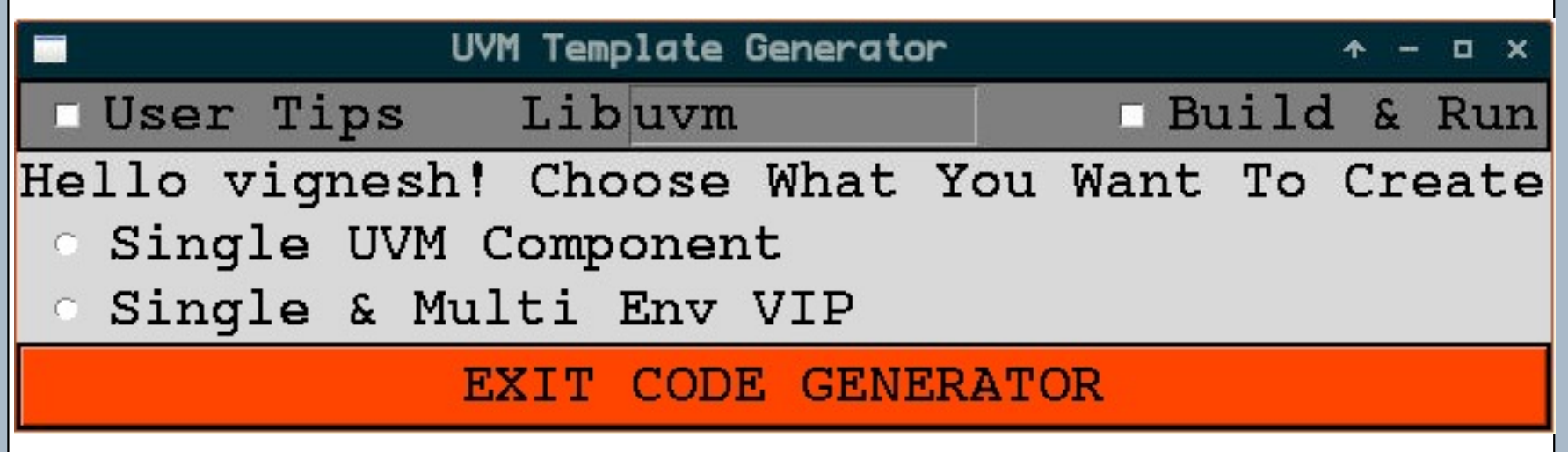
Deep Dive Into UVM Template Generator Operation

The tool is built using Python Tkinter framework to create the GUI layouts in grid fashion mechanism. All text processing and editing are done using python scripting. The tool helps in:

- Building pure UVM template codes
- Building single UVM components or complete UVM testbench and architecture
- Building Multi Agent, Multi Monitor, Multi Scoreboard based Environments
- Building Multi-Environments based flow targeting complex SOC's [System on Chip] scenarios
- Integrating Agents, Monitors, Scoreboards into already existing Environment and helps in integration between environments
- All the codes generated from this tool uses 'Natural Docs' formatting for easier documentation

Creating Single UVM Components

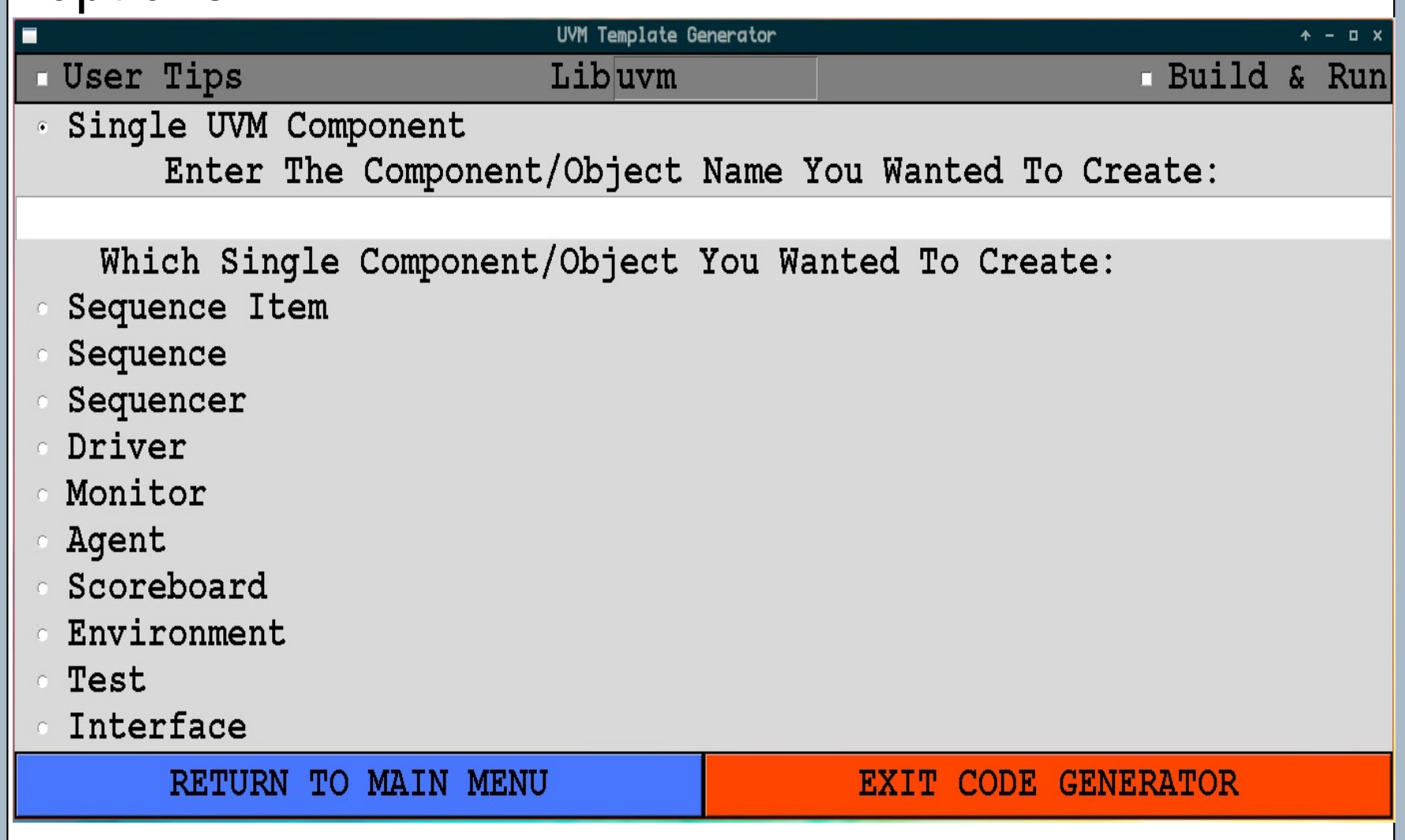
The moment user launches the tool, the GUI pops-up with two options, namely 1. create "Single UVM Component", 2. "Single & Multi Env VIP" as shown in Figure 1.



```
UVM Template Generator
User Tips        Lib uvm                 Build & Run
Hello vignesh! Choose What You Want To Create
○ Single UVM Component
○ Single & Multi Env VIP
              EXIT CODE GENERATOR
```

Figure 1. Initial Tool Layout

Once the user clicks the single component radio button, tool lists out multiple objects and component options to be created namely: sequence_item, agent, environment etc. as shown in Figure 2. The user can choose whichever component or object they want and create the corresponding templates by clicking the 'Generate Code' button. Based on the component the user chooses, the tool displays required customization options.

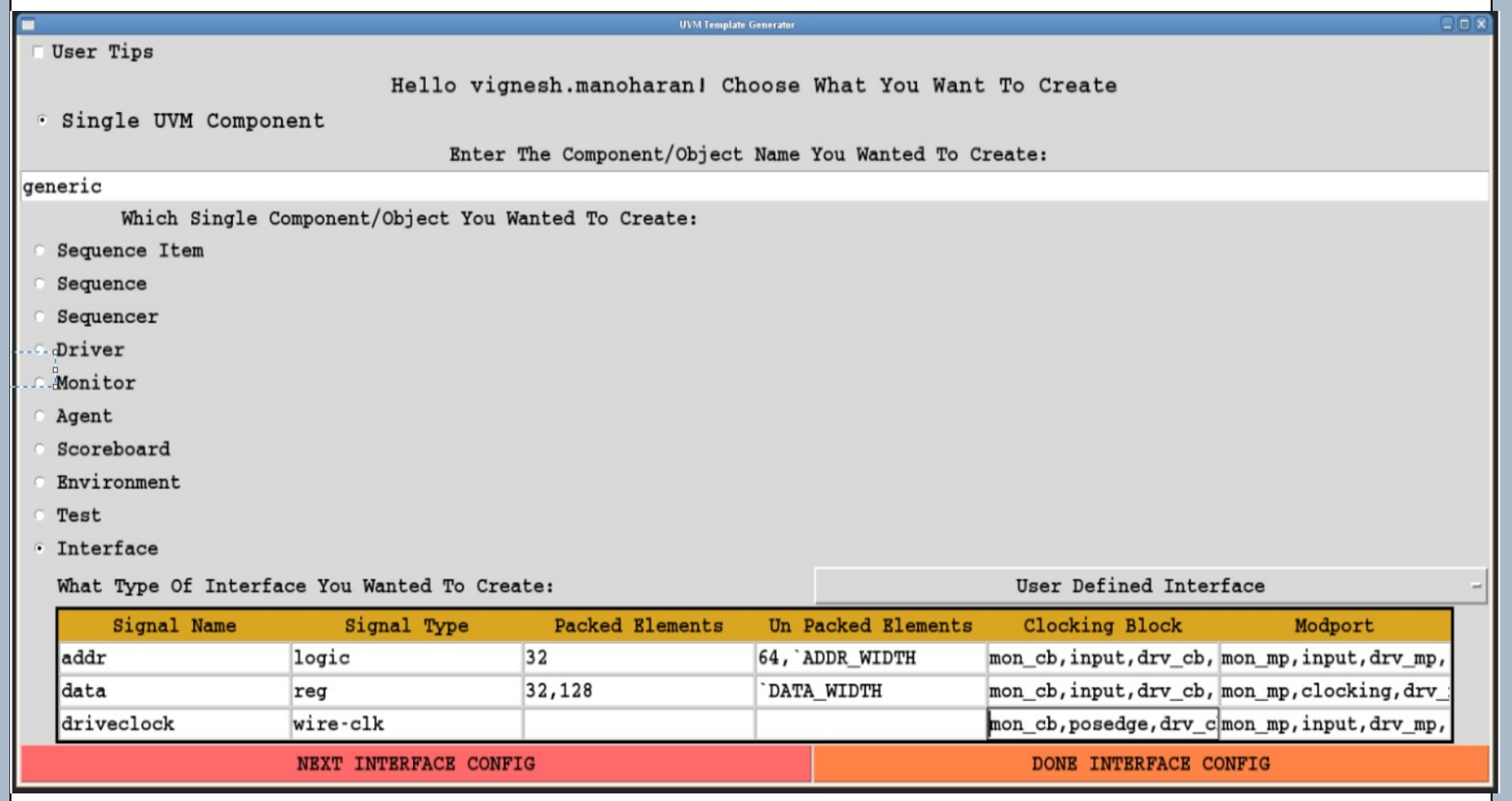


```
UVM Template Generator
User Tips        Lib uvm                 Build & Run
○ Single UVM Component
   Enter The Component/Object Name You Wanted To Create:

   Which Single Component/Object You Wanted To Create:
○ Sequence Item
○ Sequence
○ Sequencer
○ Driver
○ Monitor
○ Agent
○ Scoreboard
○ Environment
○ Test
○ Interface
RETURN TO MAIN MENU              EXIT CODE GENERATOR
```

Figure 2. Single UVM Component Tool Layout

Interface Creation: The tool provides the user with multiple options to develop an interface file such as:

- Creating a default interface with an empty shell
- A user defined interface via the GUI as shown in Figure 3.
- Loading a spreadsheet



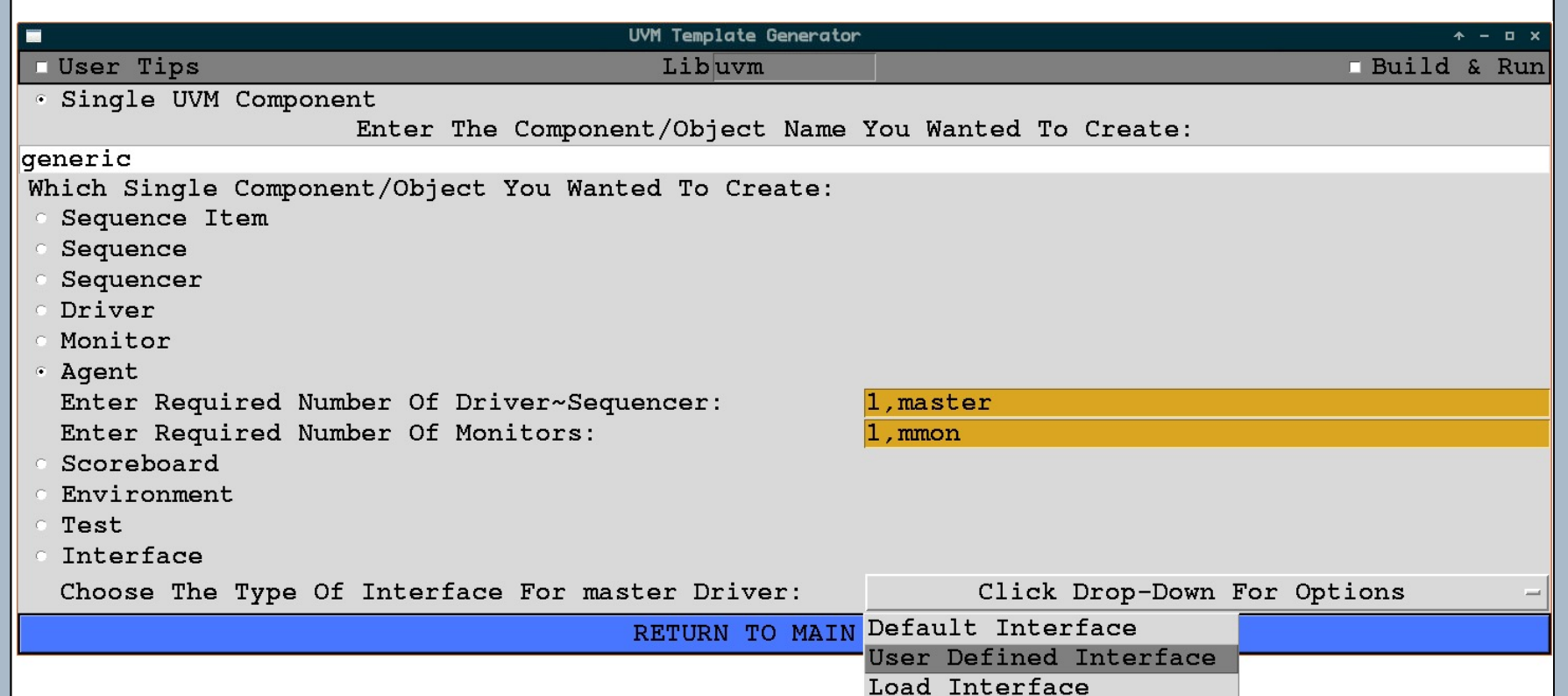
```
UVM Template Generator
User Tips
              Hello vignesh.manoharan! Choose What You Want To Create
⦿ Single UVM Component
                 Enter The Component/Object Name You Wanted To Create:
generic
      Which Single Component/Object You Wanted To Create:
○ Sequence Item
○ Sequence
○ Sequencer
○ Driver
○ Monitor
○ Agent
○ Scoreboard
○ Environment
○ Test
⦿ Interface
  What Type Of Interface You Wanted To Create:      User Defined Interface
```

Signal Name	Signal Type	Packed Elements	Un Packed Elements	Clocking Block	Modport
addr	logic	32	64,`ADDR_WIDTH	mon_cb,input,drv_cb,	mon_mp,input,drv_mp,
data	reg	32,128	`DATA_WIDTH	mon_cb,input,drv_cb,	mon_mp,clocking,drv_
driveclock	wire-clk			mon_cb,posedge,drv_c	mon_mp,input,drv_mp,

```
NEXT INTERFACE CONFIG            DONE INTERFACE CONFIG
```

Figure 3. User Defined Interface Details Filled via GUI

Agent Creation: When the user wants to create an agent, the tool further provides options for the user to enter the number of driver-sequencers or monitors they want with required names as shown in Figure 4. The tool generates the necessary code templates which are compile clean and ready to use. As soon as the code is generated, the tool goes to the default/initial layout.



```
UVM Template Generator
User Tips               Lib uvm                               Build & Run
⦿ Single UVM Component
              Enter The Component/Object Name You Wanted To Create:
generic
Which Single Component/Object You Wanted To Create:
○ Sequence Item
○ Sequence
○ Sequencer
○ Driver
○ Monitor
⦿ Agent
  Enter Required Number Of Driver~Sequencer:        1,master
  Enter Required Number Of Monitors:                1,mmon
○ Scoreboard
○ Environment
○ Test
○ Interface
  Choose The Type Of Interface For master Driver:   Click Drop-Down For Options
                     RETURN TO MAIN  Default Interface
                                     User Defined Interface
                                     Load Interface
```

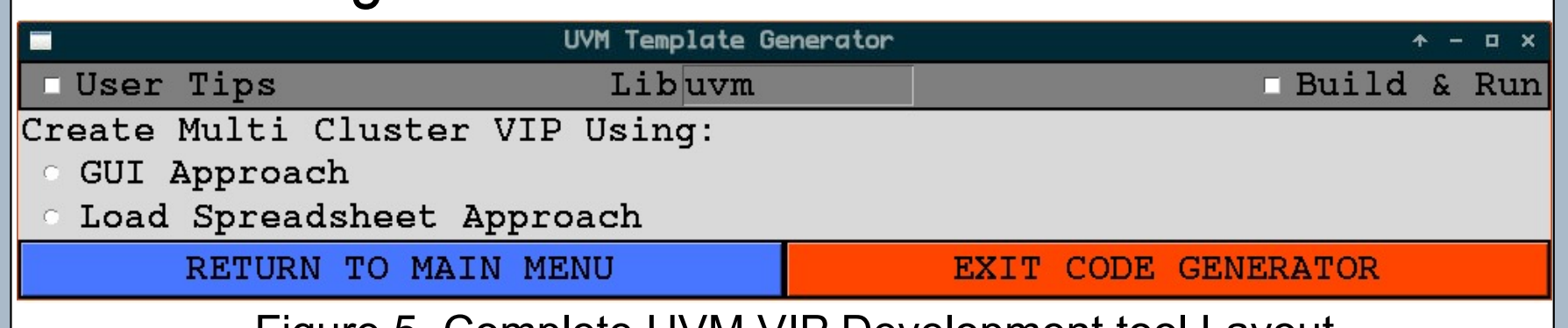
Figure 4. Component Specific Customization Layout

Creating Complete UVM VIP

The moment user clicks the "Single & Multi Env VIP" from the initial layout, the tool provides a couple of options as shown in the Figure 5, namely

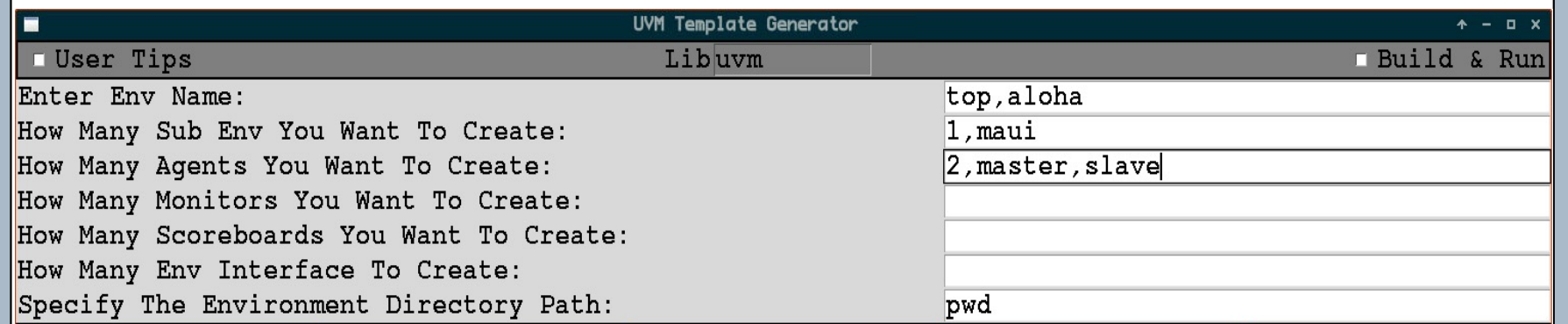
1. GUI Approach
2. Load Spreadsheet Approach

The complete GUI based approach tool layout is shown in Figure 6.



```
UVM Template Generator
User Tips            Lib uvm                    Build & Run
Create Multi Cluster VIP Using:
○ GUI Approach
○ Load Spreadsheet Approach
RETURN TO MAIN MENU               EXIT CODE GENERATOR
```

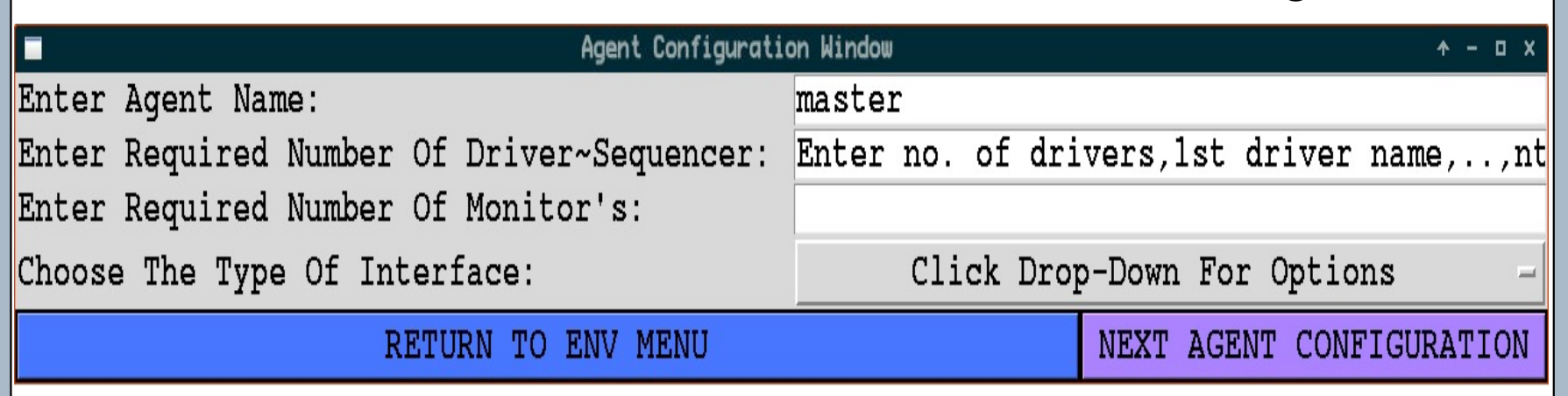
Figure 5. Complete UVM VIP Development tool Layout



```
UVM Template Generator
User Tips                      Lib uvm                       Build & Run
Enter Env Name:                                 top,aloha
How Many Sub Env You Want To Create:            1,maui
How Many Agents You Want To Create:             2,master,slave
How Many Monitors You Want To Create:
How Many Scoreboards You Want To Create:
How Many Env Interface To Create:
Specify The Environment Directory Path:         pwd
RETURN TO MAIN MENU | EXIT CODE GENERATOR | ENV SETUP | LOAD ENV CFG | NEXT ENV CFG
```

Figure 6. GUI Approach tool layout with partial filled in data

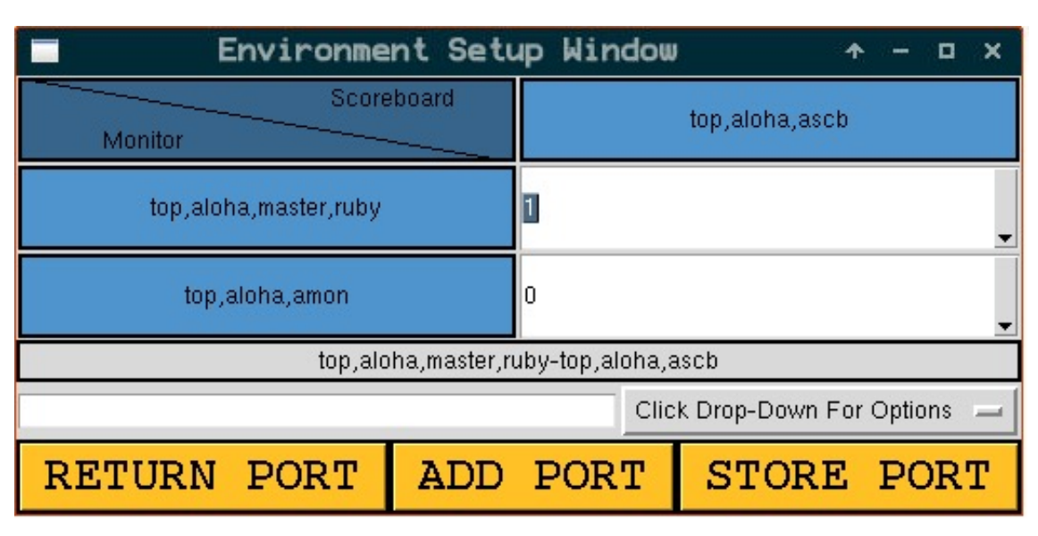
As the user starts filling in the details about the environment to be created, tool intuitively brings up the required widgets to provide necessary details. For example, when the user starts filling the details about the agent, the tool provides input widgets to enter the details about driver, agent level monitor and the interface information as shown in Figure 7.



```
Agent Configuration Window
Enter Agent Name:                           master
Enter Required Number Of Driver~Sequencer:  Enter no. of drivers,1st driver name,..,nt
Enter Required Number Of Monitor's:
Choose The Type Of Interface:               Click Drop-Down For Options
RETURN TO ENV MENU                          NEXT AGENT CONFIGURATION
```

Figure 7. Popped up tool window for acquiring agent details

Once the user enters all the required information about the environment to be built, the user will click the "Env Setup" button found at the bottom of the tool window. By clicking that button, the tool generates a matrix table with all the monitors, scoreboards and provide the option for user to make the necessary connection as shown in Figure 8.



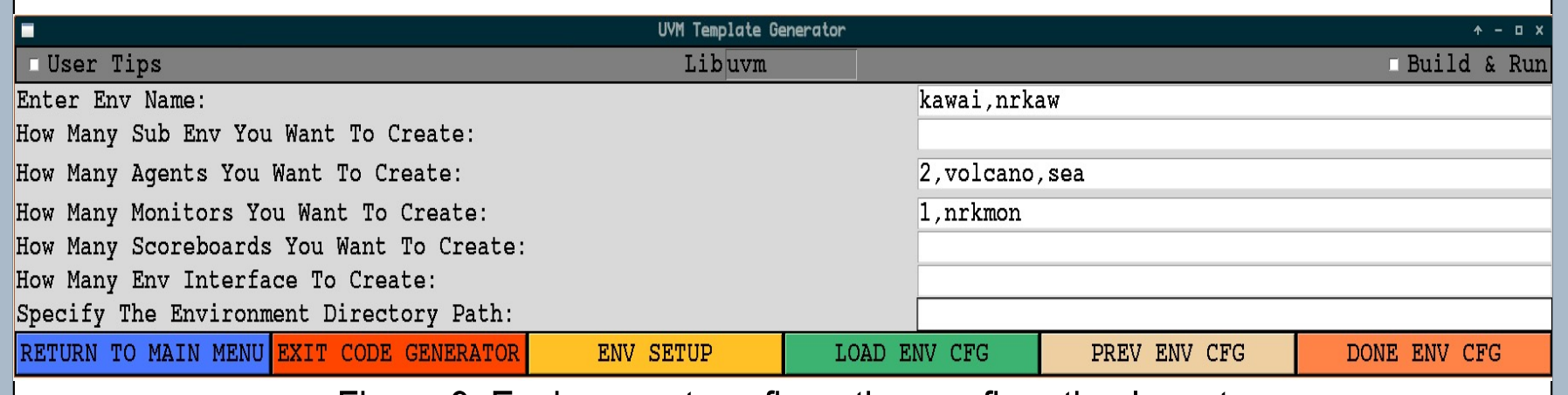
Environment Setup Window

Monitor \ Scoreboard	top,aloha,ascb
top,aloha,master,ruby	1
top,aloha,amon	0

```
top,aloha,master,ruby-top,aloha,ascb
                         Click Drop-Down For Options
RETURN PORT    ADD PORT    STORE PORT
```

Figure 8. Monitor-Scoreboard Connectivity Matrix table

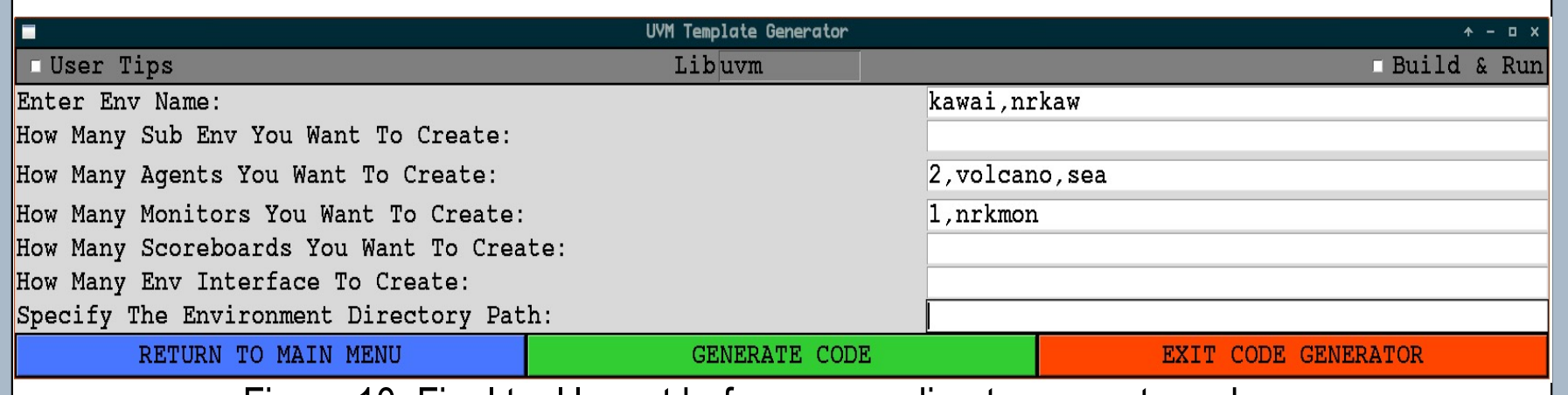
Once the user has provided the required details about all the environments and the monitor-scoreboard connectivity information, the user needs to click the "Done Env Cfg" button as shown in Figure 9 to instruct the tool that the user has confirmed all the testbench setup and it is safe to move ahead.



```
UVM Template Generator
User Tips                      Lib uvm                       Build & Run
Enter Env Name:                                 kawai,nrkaw
How Many Sub Env You Want To Create:
How Many Agents You Want To Create:             2,volcano,sea
How Many Monitors You Want To Create:           1,nrkmon
How Many Scoreboards You Want To Create:
How Many Env Interface To Create:
Specify The Environment Directory Path:
RETURN TO MAIN MENU | EXIT CODE GENERATOR | ENV SETUP | LOAD ENV CFG | PREV ENV CFG | DONE ENV CFG
```

Figure 9. Environment configuration confirmation layout

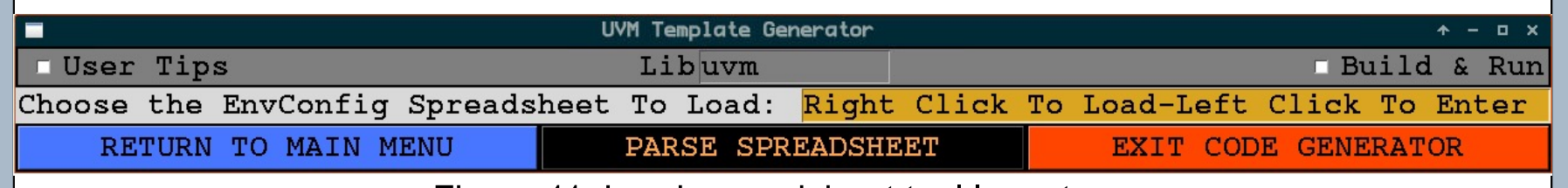
After confirming the environment configuration, the user then clicks the "Generate Code" button as shown in Figure 10. This will instruct the tool to build the testbench codes, necessary files, and directory structures.



```
UVM Template Generator
User Tips                      Lib uvm                       Build & Run
Enter Env Name:                                 kawai,nrkaw
How Many Sub Env You Want To Create:
How Many Agents You Want To Create:             2,volcano,sea
How Many Monitors You Want To Create:           1,nrkmon
How Many Scoreboards You Want To Create:
How Many Env Interface To Create:
Specify The Environment Directory Path:
RETURN TO MAIN MENU        GENERATE CODE        EXIT CODE GENERATOR
```

Figure 10. Final tool layout before proceeding to generate code

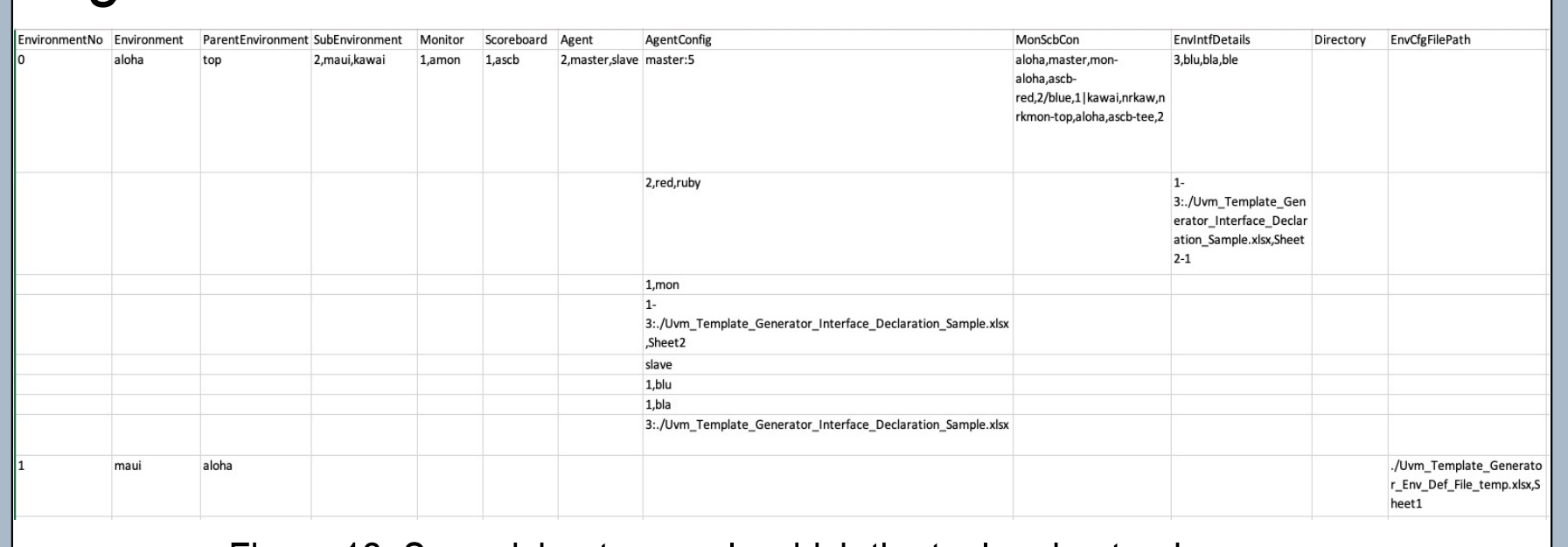
In the complete UVM VIP tool window, when the user chooses the "Load Spreadsheet Approach", the tool pops up with the layout as shown in Figure 11.



```
UVM Template Generator
User Tips                      Lib uvm                       Build & Run
Choose the EnvConfig Spreadsheet To Load: Right Click To Load-Left Click To Enter
RETURN TO MAIN MENU        PARSE SPREADSHEET        EXIT CODE GENERATOR
```

Figure 11. Load spreadsheet tool layout

In this mode, the user can enter the details in a tool understandable spreadsheet format as shown in the Figure 12.



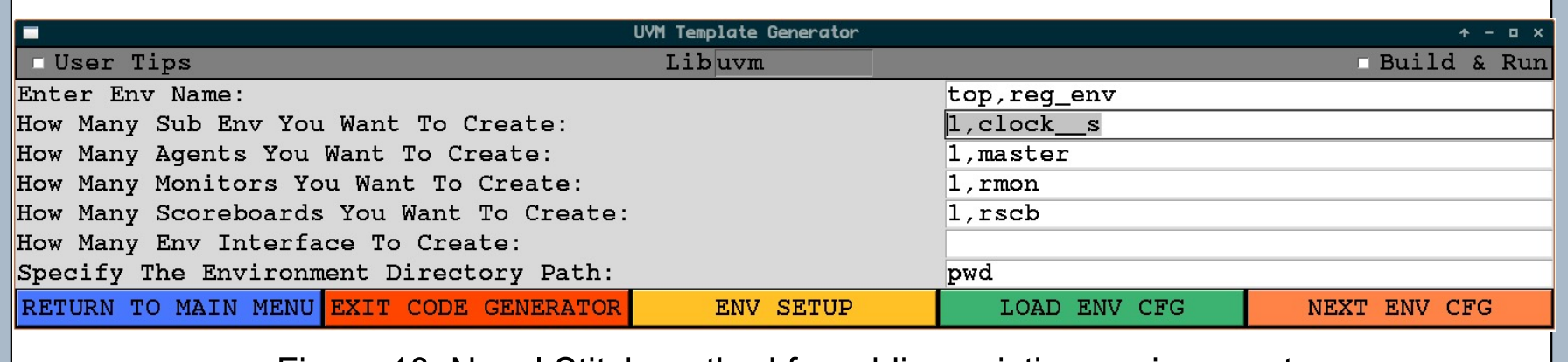
EnvironmentNo	Environment	ParentEnvironment	SubEnvironment	Monitor	Scoreboard	Agent	AgentConfig	MonScbCon	EnvIntfDetails	Directory	EnvCfgFilePath
0	aloha	top	2,maui,kawai	1,amon	1,ascb	2,master,slave	master:5	aloha,master,mon-aloha,ascb-red,2/blue,1|kawai,nrkaw,nrkmon-top,aloha,ascb-tee,2	3,blu,bla,ble		
							2,red,ruby		1-3:./Uvm_Template_Generator_Interface_Declaration_Sample.xlsx,Sheet2-1		
							1,mon				
							1-3:./Uvm_Template_Generator_Interface_Declaration_Sample.xlsx,Sheet2				
							slave				
							1,blu				
							1,bla				
							3:./Uvm_Template_Generator_Interface_Declaration_Sample.xlsx				
1	maui	aloha									./Uvm_Template_Generator_Env_Def_File_temp.xlsx,Sheet1

Figure 12. Spreadsheet example which the tool understands

Novel Stitch, Create & Stitch Feature

The art of developing a testbench doesn't happen in a single day but is a continual long-term process. For example, on day 1, the user might just need to build the environment skeleton. On day 2, the user might end up adding few other components namely agents and environment level monitors. Later the user adds the required scoreboard and connectivity. How does this tool take care of such cases? Well, the tool provides couple of novel features namely, "Stitch", "Create & Stitch" modes which helps in incremental testbench development process.

Stitch Mode: This mode comes in handy if the user has already created an environment with the tool which takes care of generating different kinds of clock sources and now the user wants to build a block level bench which is going to take care of register programming. The user needs to launch the tool, generate the required block level testbench skeleton and then, using the "Stitch mode", the user can stitch the other sub-environments into this block level environment. The user needs to enter the required number of sub-env's wanted to be stitched and add the sub-env's names appended with "__s" as shown in the Figure 13.

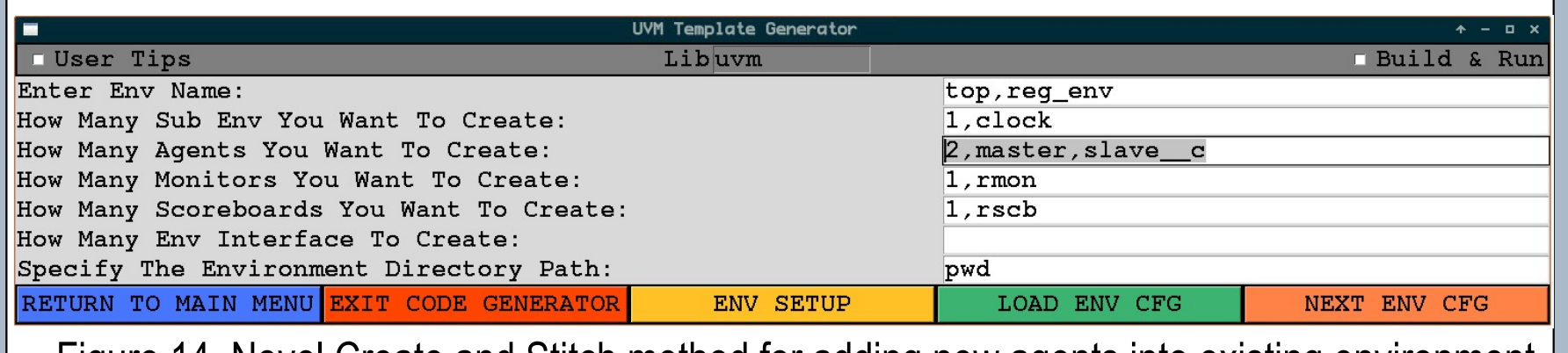


```
UVM Template Generator
User Tips                      Lib uvm                       Build & Run
Enter Env Name:                                 top,reg_env
How Many Sub Env You Want To Create:            1,clock__s
How Many Agents You Want To Create:             1,master
How Many Monitors You Want To Create:           1,rmon
How Many Scoreboards You Want To Create:        1,rscb
How Many Env Interface To Create:
Specify The Environment Directory Path:         pwd
RETURN TO MAIN MENU | EXIT CODE GENERATOR | ENV SETUP | LOAD ENV CFG | NEXT ENV CFG
```

Figure 13. Novel Stitch method for adding existing environment

Create & Stitch Mode: If, in the above block level environment, the user wants to add a new agent, the user can do so by launching the tool and loading the block level environment using "Load Spreadsheet" Mode. Next, the user can add the details about new agent i.e., number of agents followed by the name of agent appended with "__c", as shown in Figure 14 and then clicking the "Generate Code" button.

The tool knows the user is adding new components onto an existing block level environment, so the tool generates only the new components and then stitches them onto the already existing environment in all the necessary places.



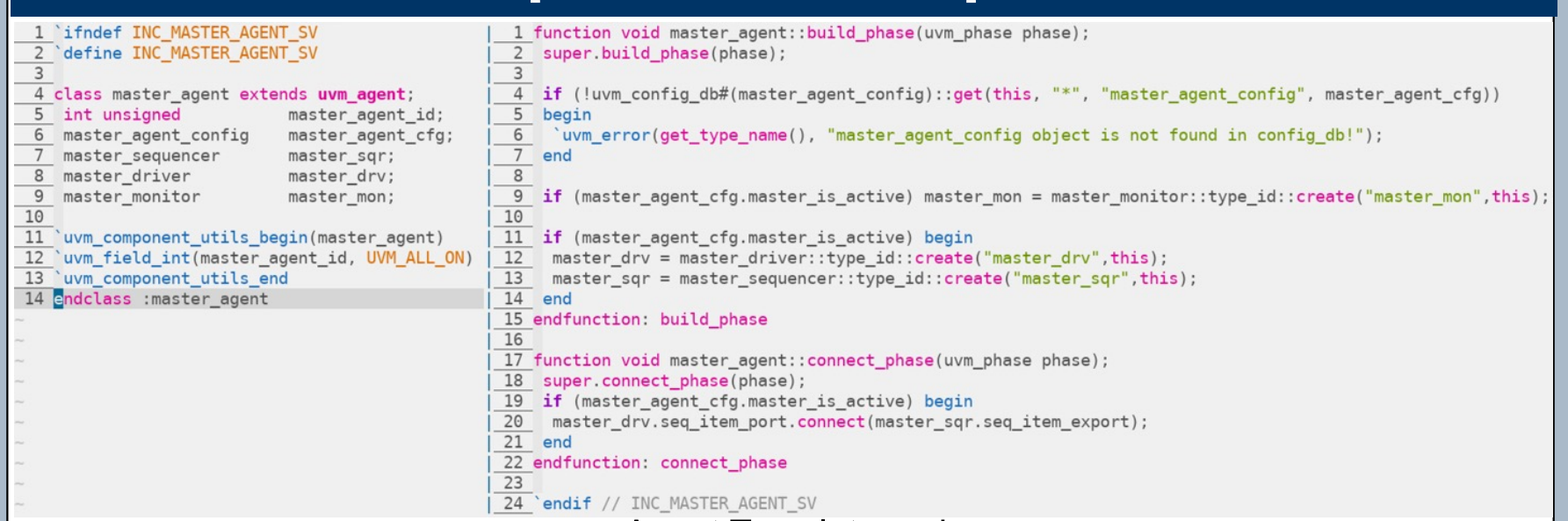
```
UVM Template Generator
User Tips                      Lib uvm                       Build & Run
Enter Env Name:                                 top,reg_env
How Many Sub Env You Want To Create:            1,clock
How Many Agents You Want To Create:             2,master,slave__c
How Many Monitors You Want To Create:           1,rmon
How Many Scoreboards You Want To Create:        1,rscb
How Many Env Interface To Create:
Specify The Environment Directory Path:         pwd
RETURN TO MAIN MENU | EXIT CODE GENERATOR | ENV SETUP | LOAD ENV CFG | NEXT ENV CFG
```

Figure 14. Novel Create and Stitch method for adding new agents into existing environment

The granularity level at which the work can start from can be:

- Adding single or multiple port connectivity's between monitor and scoreboard
- Adding new environment level monitor's and scoreboard's in already existing environment with updated connectivity
- Adding driver/monitor inside already existing agents and adding new agents into existing environment
- Adding an environment using create & stitch or just stitch process into an already existing environment

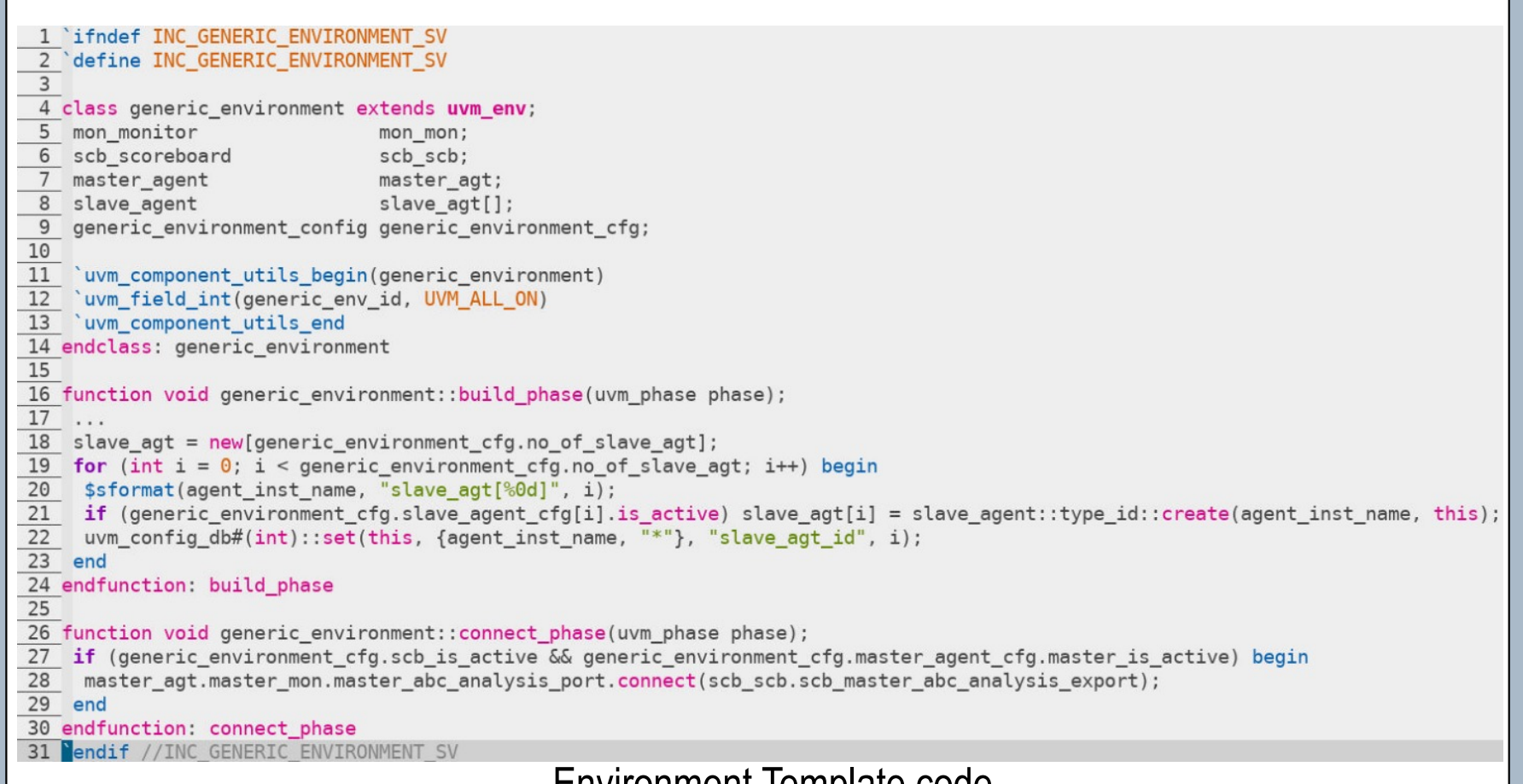
Sample Code Snapshots



```
`ifndef INC_MASTER_AGENT_SV
`define INC_MASTER_AGENT_SV

class master_agent extends uvm_agent;
 int unsigned           master_agent_id;
 master_agent_config    master_agent_cfg;
 master_sequencer       master_sqr;
 master_driver          master_drv;
 master_monitor         master_mon;

`uvm_component_utils_begin(master_agent)
`uvm_field_int(master_agent_id, UVM_ALL_ON)
`uvm_component_utils_end
endclass :master_agent
```

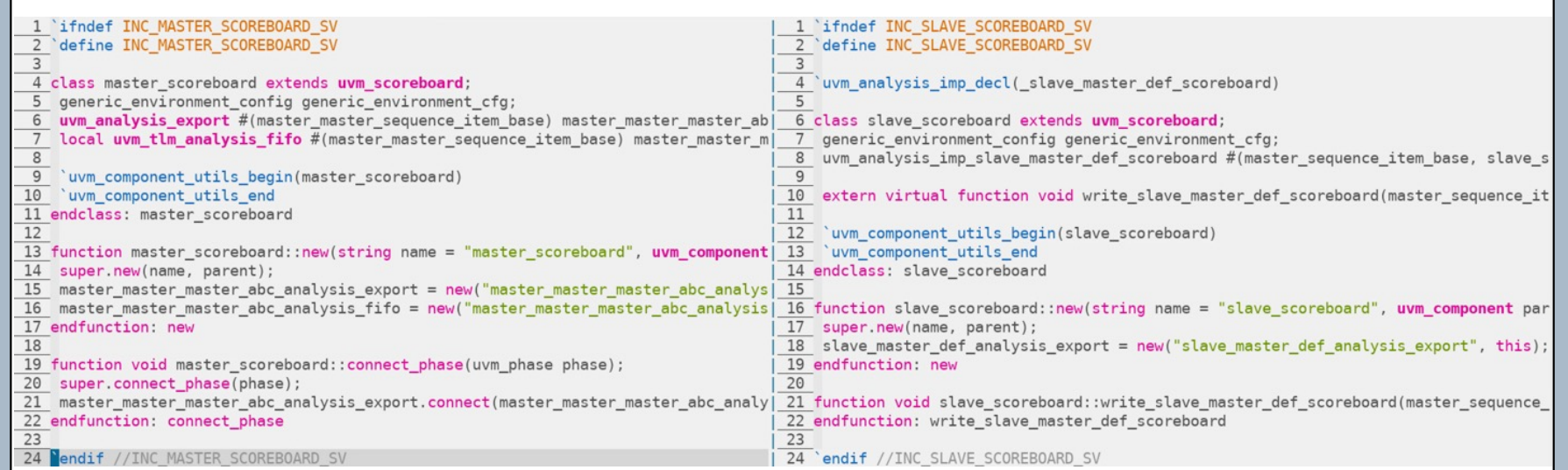
```
function void master_agent::build_phase(uvm_phase phase);
super.build_phase(phase);

 if (!uvm_config_db#(master_agent_config)::get(this, "*", "master_agent_config", master_agent_cfg))
 begin
  `uvm_error(get_type_name(), "master_agent_config object is not found in config_db!");
 end

 if (master_agent_cfg.master_is_active) master_mon = master_monitor::type_id::create("master_mon",this);

 if (master_agent_cfg.master_is_active) begin
  master_drv = master_driver::type_id::create("master_drv",this);
  master_sqr = master_sequencer::type_id::create("master_sqr",this);
 end
endfunction: build_phase

function void master_agent::connect_phase(uvm_phase phase);
super.connect_phase(phase);
 if (master_agent_cfg.master_is_active) begin
  master_drv.seq_item_port.connect(master_sqr.seq_item_export);
 end
endfunction: connect_phase

 `endif // INC_MASTER_AGENT_SV
```

Agent Template code



```
`ifndef INC_GENERIC_ENVIRONMENT_SV
`define INC_GENERIC_ENVIRONMENT_SV

class generic_environment extends uvm_env;
 mon_monitor                 mon_mon;
 scb_scoreboard              scb_scb;
 master_agent                master_agt;
 slave_agent                 slave_agt[];
 generic_environment_config generic_environment_cfg;

 `uvm_component_utils_begin(generic_environment)
 `uvm_field_int(generic_env_id, UVM_ALL_ON)
 `uvm_component_utils_end
endclass: generic_environment

function void generic_environment::build_phase(uvm_phase phase);
 ...
 slave_agt = new[generic_environment_cfg.no_of_slave_agt];
 for (int i = 0; i < generic_environment_cfg.no_of_slave_agt; i++) begin
  $sformat(agent_inst_name, "slave_agt[%0d]", i);
  if (generic_environment_cfg.slave_agent_cfg[i].is_active) slave_agt[i] = slave_agent::type_id::create(agent_inst_name, this);
  uvm_config_db#(int)::set(this, {agent_inst_name, "*"}, "slave_agt_id", i);
 end
endfunction: build_phase

function void generic_environment::connect_phase(uvm_phase phase);
 if (generic_environment_cfg.scb_is_active && generic_environment_cfg.master_agent_cfg.master_is_active) begin
  master_agt.master_mon.master_abc_analysis_port.connect(scb_scb.scb_master_abc_analysis_export);
 end
endfunction: connect_phase
`endif //INC_GENERIC_ENVIRONMENT_SV
```

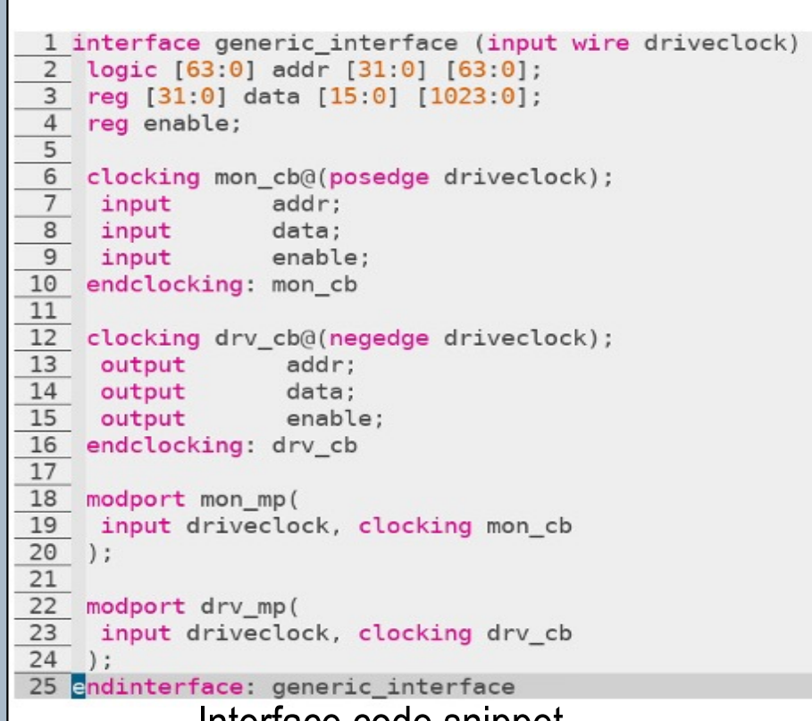
Environment Template code



```
`ifndef INC_MASTER_SCOREBOARD_SV
`define INC_MASTER_SCOREBOARD_SV

class master_scoreboard extends uvm_scoreboard;
 generic_environment_config generic_environment_cfg;
 uvm_analysis_export #(master_master_sequence_item_base) master_master_master_ab
 local uvm_tlm_analysis_fifo #(master_master_sequence_item_base) master_master_m

 `uvm_component_utils_begin(master_scoreboard)
 `uvm_component_utils_end
endclass: master_scoreboard

function master_scoreboard::new(string name = "master_scoreboard", uvm_component
 super.new(name, parent);
 master_master_master_abc_analysis_export = new("master_master_master_abc_analys
 master_master_master_abc_analysis_fifo = new("master_master_master_abc_analysis
endfunction: new

function void master_scoreboard::connect_phase(uvm_phase phase);
 super.connect_phase(phase);
 master_master_master_abc_analysis_export.connect(master_master_master_abc_analy
endfunction: connect_phase

`endif //INC_MASTER_SCOREBOARD_SV
```

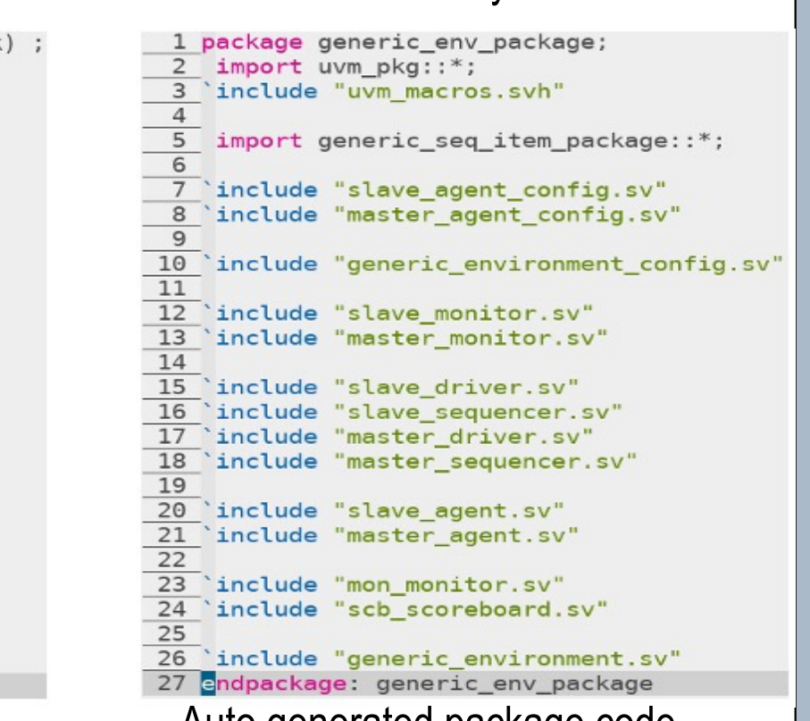
```
`ifndef INC_SLAVE_SCOREBOARD_SV
`define INC_SLAVE_SCOREBOARD_SV

`uvm_analysis_imp_decl(_slave_master_def_scoreboard)

class slave_scoreboard extends uvm_scoreboard;
 generic_environment_config generic_environment_cfg;
 uvm_analysis_imp_slave_master_def_scoreboard #(master_sequence_item_base, slave_s

 extern virtual function void write_slave_master_def_scoreboard(master_sequence_it

 `uvm_component_utils_begin(slave_scoreboard)
 `uvm_component_utils_end
endclass: slave_scoreboard

function slave_scoreboard::new(string name = "slave_scoreboard", uvm_component par
 super.new(name, parent);
 slave_master_def_analysis_export = new("slave_master_def_analysis_export", this);
endfunction: new

function void slave_scoreboard::write_slave_master_def_scoreboard(master_sequence_
endfunction: write_slave_master_def_scoreboard

 `endif //INC_SLAVE_SCOREBOARD_SV
```

Code difference between FIFO based and Write function-based connectivity in Scoreboard



```
interface generic_interface (input wire driveclock) ;
 logic [63:0] addr [31:0] [63:0];
 reg [31:0] data [15:0] [1023:0];
 reg enable;

 clocking mon_cb@(posedge driveclock);
  input       addr;
  input       data;
  input       enable;
 endclocking: mon_cb

 clocking drv_cb@(negedge driveclock);
  output       addr;
  output       data;
  output       enable;
 endclocking: drv_cb

 modport mon_mp(
  input driveclock, clocking mon_cb
 );

 modport drv_mp(
  input driveclock, clocking drv_cb
 );
endinterface: generic_interface
```

Interface code snippet



```
package generic_env_package;
 import uvm_pkg::*;
 `include "uvm_macros.svh"

 import generic_seq_item_package::*;

 `include "slave_agent_config.sv"
 `include "master_agent_config.sv"

 `include "generic_environment_config.sv"

 `include "slave_monitor.sv"
 `include "master_monitor.sv"

 `include "slave_driver.sv"
 `include "slave_sequencer.sv"
 `include "master_driver.sv"
 `include "master_sequencer.sv"

 `include "slave_agent.sv"
 `include "master_agent.sv"

 `include "mon_monitor.sv"
 `include "scb_scoreboard.sv"

 `include "generic_environment.sv"
endpackage: generic_env_package
```

Auto generated package code

Performance Evaluation

Comparison Points	Easier UVM Code Generator	Open Titan UVM Generator	Novel GUI Based UVM testbench Template Builder
License	Open source	Open source	Open source
Support GUI	No	No	Yes
Generation of UVM class code	Yes	Yes	Yes
Generation of complete UVM environment	No	Yes	Yes
Generation of multi-instance of agents, monitors, environments, etc. for complex testbench	No	No	Yes
Smart monitor and scoreboard connectivity	No	No	Yes
Incremental testbench development	No	No	Yes
Environment Integration	No	No	Yes
Open-source documentation formatting	No	No	Yes

Summary

The UVM template generator provides the user to create any component template or the entire VIP dynamically in matter of minutes. The generator helps in standardization of code development and re-usability of the code across the projects and helps in complete integration of the verification collateral. With the template generator's unique 'Create & Stitch' feature, the tool can add new components, add connection between components or append sub environment VIPs to the already existing code and help in incremental enhancement of the testbench. Hence, this tool indeed improved verification productivity and showcased its performance in complex streamlined products.