Advanced Functional Verification Methodology Using UVM For Complex DSP Algorithms In Mixed Signal RF SoCs

Srinivas Aluri

Jaimin Mehta

TEXAS INSTRUMENTS

Agenda

- About this paper
- Verification architecture
- Matlab integration with HVL
- Use case illustrations
- Conclusion
- Q & A

About this paper

- Complex mixed signal SOC's involving RF front end and digital signal processing blocks create a unique challenge for functional verification.

- Signal processing tools provides powerful built in functions, to realize DSP functionality, signal generation and power spectral analysis.

- This paper illustrates a methodology to integrate commercial signal processing tools into UVM to create a robust self-checking test bench which can thoroughly verify complex DSP algorithms.

Verification architecture

- Test bench is architected using UVM with various UVC's for configuration of signal processing chain and stimulus generation.

- Analog frontend and mixed signal modules which transfer data to DSP are implemented in discrete domain using VerilogAMS to speed up simulations

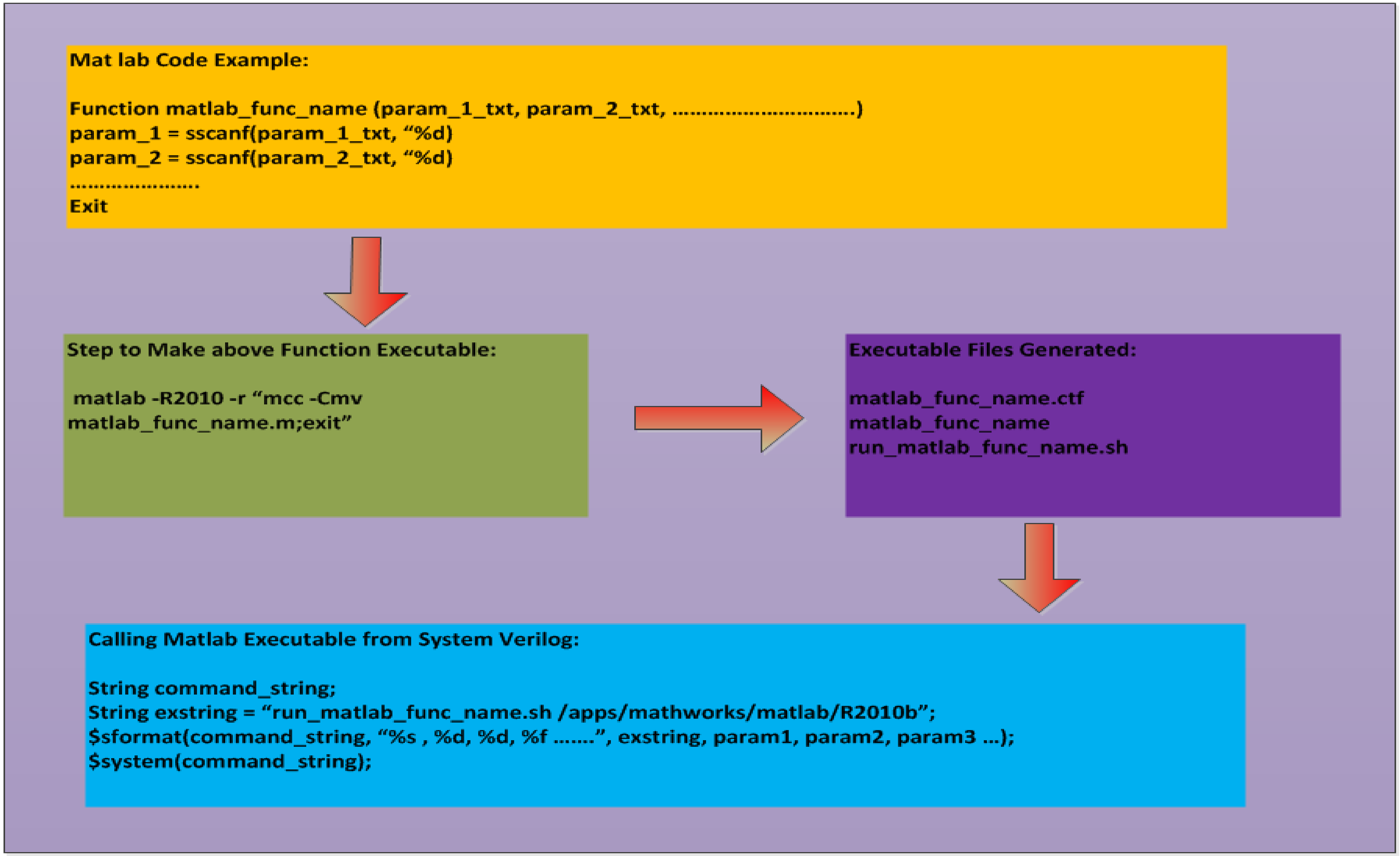
- SV interfaces and monitors are used along the signal chain to pool data at the input and output of each component and transfer data at a transaction level into the scoreboard.

- Digital signal path consisting of typical signal processing blocks like decimators, interpolators, gain etc.. are verified extensively by integrating signal processing checkers and reference model into test bench environment.

Matlab integration with HVL

- Reference model/checker should be coded in matlab with .m file extension.
- Input parameters need to be made string type for the matlab function.
- Using matlab compiler convert .m file into executable shell script.
- Create a command string with executable shell script along with its necessary parameters.
- Execute the command string from HVL using $system() function.

Matlab integration flow chart



Mat lab Code Example:

```
Function matlab_func_name (param_1_txt, param_2_txt, ..............................)
param_1 = sscanf(param_1_txt, “%d)
param_2 = sscanf(param_2_txt, “%d)
......................
Exit
```

Step to Make above Function Executable:

```
 matlab -R2010 -r “mcc -Cmv
matlab_func_name.m;exit”
```

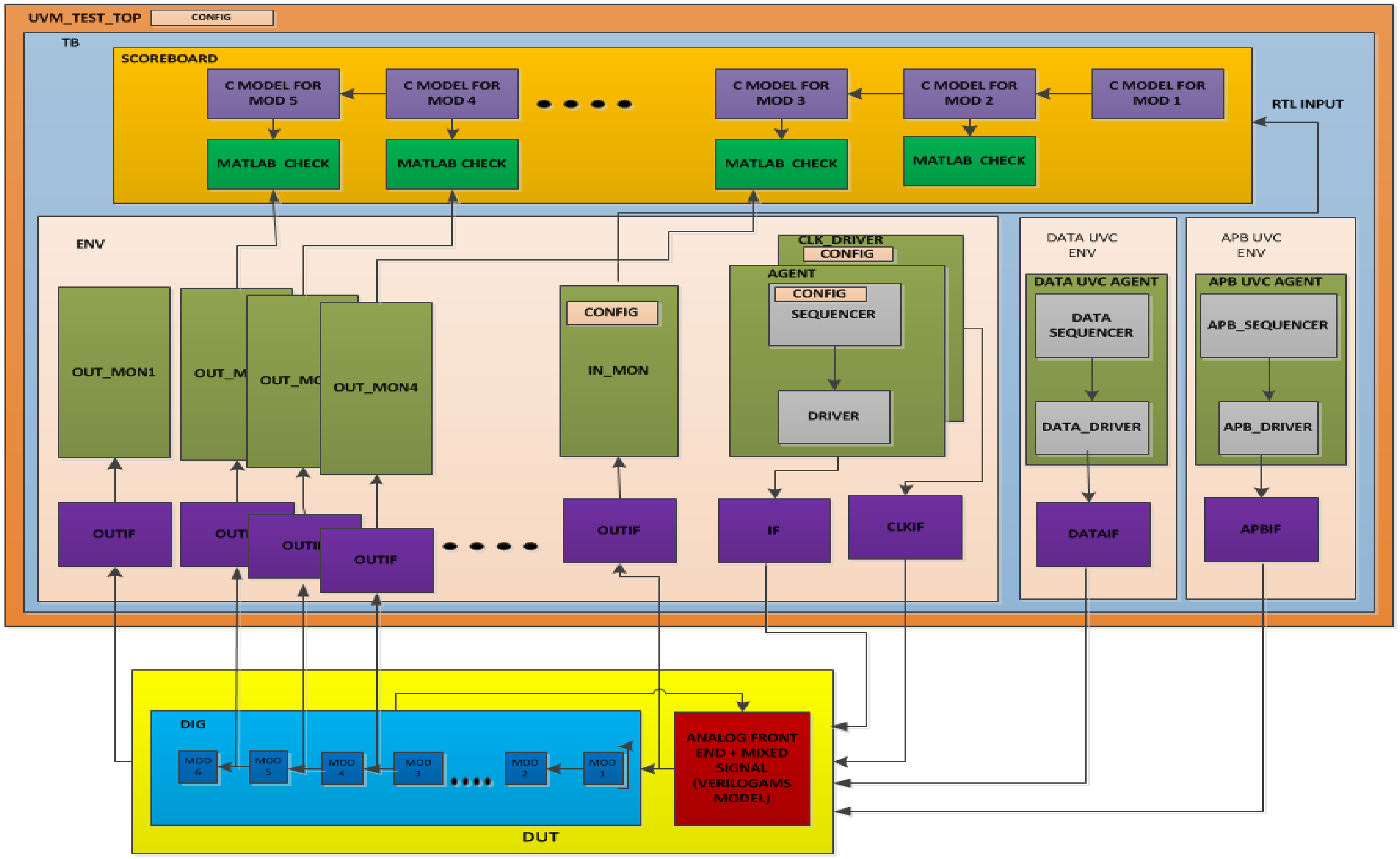
Executable Files Generated:

```
matlab_func_name.ctf
matlab_func_name
run_matlab_func_name.sh
```

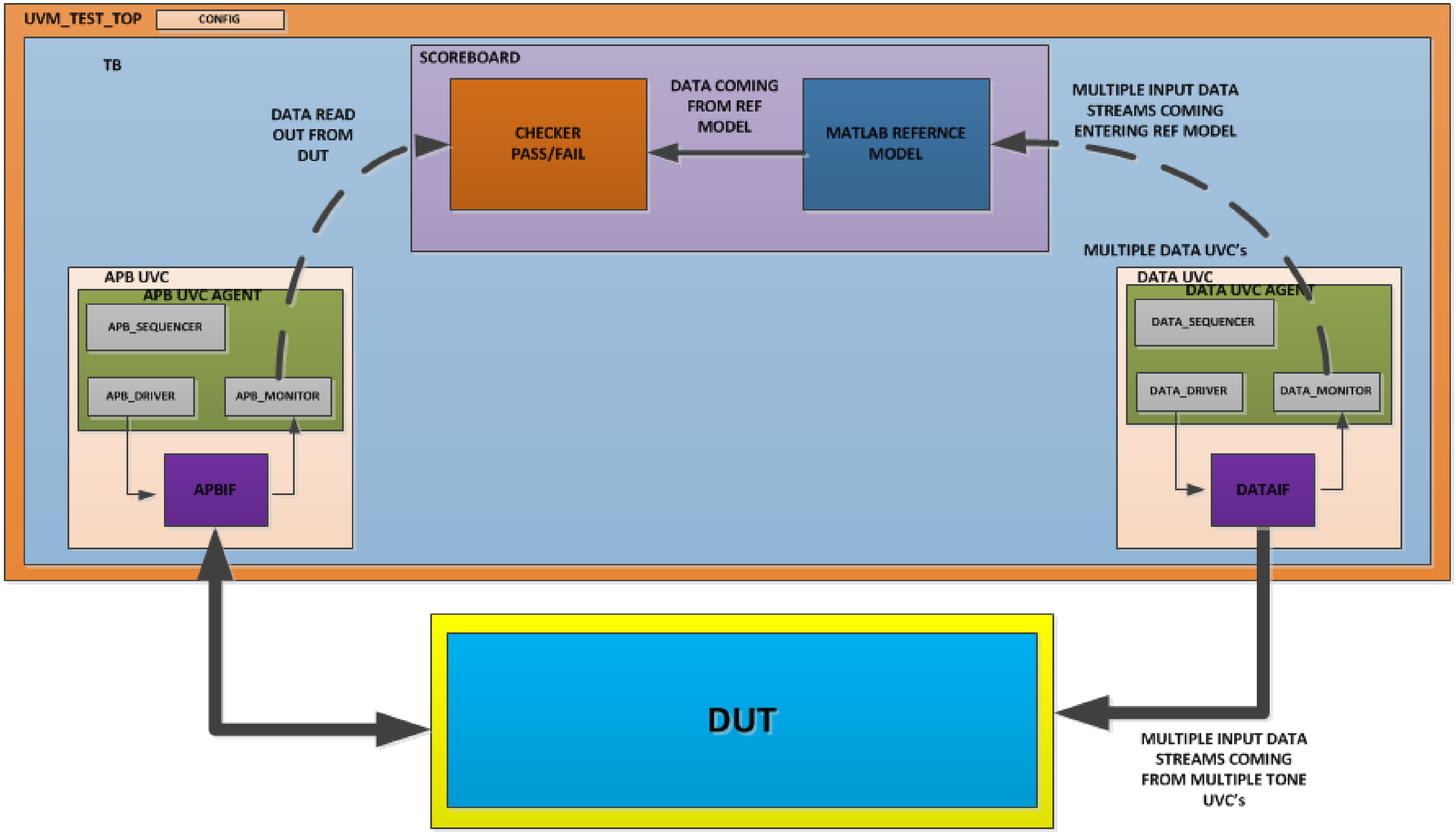
Calling Matlab Executable from System Verilog:

```
String command_string;
String exstring = “run_matlab_func_name.sh /apps/mathworks/matlab/R2010b”;
$sformat(command_string, “%s , %d, %d, %f .......”, exstring, param1, param2, param3 ...);
$system(command_string);
```

TB: illustration -1



TB: illustration - II



Conclusion

- Mentioned procedure was successfully used to call complex mat lab checkers, reference model created in mat lab.
- This methodology helped us in creating effective self-checking test bench architecture, and enabled us to re-use most of the system level functions and reference model thus reducing verification cycle time.
- Complex signal processing functions can easily be coded in matlab and called from systemverilog, which reduces the complexity and many lines of code needed if the same is implemented in HVL's.
- Finally this methodology also has proven effective when running lot of regressions, since it does not need to invoke the mat lab engine.

Thanks for your time!!

Email: s-aluri@ti.com