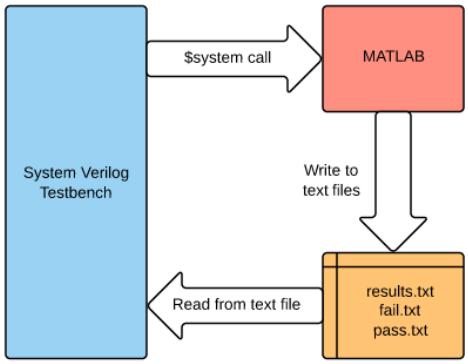
INTERFACING PYTHON WITH A SYSTEMVERILOG TEST BENCH

Lakshay Grover[1] Kaushal Modi[2]

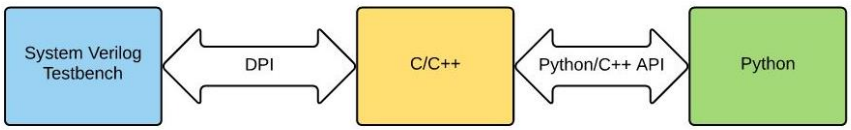
[1] Analog Devices, Inc., 3 Technology Way, Norwood, MA 0206, lakshay.grover@analog.com,
[2] kaushal.modi@analog.com.

1. Introduction: As a design verification engineer, there is a constant need to use foreign languages like C, C++ or MATLAB for complex calculations that SystemVerilog may not be able to handle easily. MATLAB is widely used to create signal processing reference models because of the availability of numerous mathematical libraries. Apart from its libraries, it can be used to easily plot graphs and calls to a MATLAB function can be made from a SystemVerilog test bench. This paper discusses how Python can be used with SystemVerilog as an alternative to MATLAB.

2. Problem Statement: Even after the benefits of MATLAB, it has the inherent overhead of acquiring a license, requires tailored knowledge of using it and cannot be interfaced with SystemVerilog seamlessly. We can invoke MATLAB through the test bench using a *$system* call, but that is as good as calling MATLAB from the terminal. If output is needed from the MATLAB function, then it has to be dumped to a file which can then be read by SystemVerilog test bench. This becomes a performance issue if we are dealing with a large data set, and if the frequency of file access is high. Alternatively, a C/SystemVerilog model can be generated from MATLAB but that requires an extra MATLAB compiler license which has limitations of being low level, difficult to maintain and extend.



3. Proposed Solution: Python is one of the most widely used scientific programming languages. A plethora of Python libraries like numpy, matplotlib and scipy are available which makes it possible to write code functionally equivalent to the MATLAB version. Python is open source and has a number of IDEs available. Also we can have checkers implemented in Python itself. Although Python cannot be directly interfaced with SystemVerilog, but C++ can be, and C++ can be interfaced with SystemVerilog. Using a cascade of ***Direct Programming Interface (DPI)*** for SystemVerilog–C++, and ***Python-C Application Programmer's Interface (PythonC API)*** for Python–C++, we can have a near-seamless interface between Python and SystemVerilog.



4. Technologies and Services Employed

4.1. Direct Programming Interface (DPI)

DPI is an interface between SystemVerilog and a foreign language (C/ C++/ SystemC). The beauty of using DPI is that both the SystemVerilog and the foreign language layer are isolated from each other.

4.1.1. Usage of DPI

> SystemVerilog test bench would require an import of the C++ function that can be defined in a C++ library.
> This function can be called from the test bench using compatible data types.
> Use of classes is not permitted, but structures can be used.

4.2 Python-C API

PyhtonC API provides access to python in two ways, extending or embedding it. In this case the python is embedded into the C++ code, which means that a python interpreter can be initialized and call backs to a python code at specific times.

4.2.1 Usage of Python-C API

> The PythonC API is incorporated in the C++ source file by including the header "Python.h". A python interpreter is initialized in the C++ code.
> Further, the name of the Python file and function is specified.
> The data is then sent in the form of a tuple. If data needs to be returned we can receive it in a tuple, from where the data can be copied to C-compatible data types.
> After this is no further use is required, the Python interpreter can be closed.

- **Additions to SV test bench.**

```
import "DPI" function void SV_to_CPP_to_Python (input struct struct_IN,
output struct struct_OUT);
SV_to_CPP_to_Python(structObj_in, structObj_out);
```

- **Creation of C++ library.**

```
#include "svdpi.h"
#include <Python.h>
extern "C" void  SV_to_CPP_to_Python(struct* C_Obj_In, struct* C_Obj_Out ) {
  PyObject *pName, *pModule, *pFunc;
  PyObject *tuple_in, *tuple_out;
  Py_Initialize();
  PyRun_SimpleString("import sys");

PyRun_SimpleString("sys.path.append(\"/home/lgrover/sandbox2/SCP/DVCon\")");
  pName   = PyString_FromString(<Python file name>);
  pModule = PyImport_Import(pName);
  pFunc   = PyObject_GetAttrString(pModule,<Python Function>);

  tuple_in = PyTuple_New(<num of arguments to be send>);
  PyTuple_SetItem (tuple_in, i, PyInt/Float/String_FromLong/Double/String
(C_Obj_In-> intVar) );

  tuple_out = PyTuple_New(num of arguments to be received);
  tuple_out = PyObject_CallObject(pFunc, tuple_in);
 C_Obj_Out-> intVar = PyInt/Float/String_AsLong/Double/String (
PyTuple_GetItem(tuple_out, i) );

  Py_Finalize(); }
```

- **Creation of Python Model.**

5. Advantages of the proposed solutions

Data can be returned by python to SystemVerilog unlike MATLAB. The python code can be made self-checking which can reduce the burden on the testbench. Another advantage of this method is that unlike the MATLAB approach, we need not compile the Python code when a change is made as it interpreted during compilation of the test bench. Also, the C++ code only acts as a wrapper and needs to be changed only when adding new variables. If no changes are made, the previous compiled object can be used.

Case Study: For a number of Fast Fourier Transform calculations, as reference, we noticed a speedup of 24.9% for our simulations. The output was checked for both the versions and were found to be consistent. For using the python approach, a python model had to be created to mimic the MATLAB version of fft calculaions. Creation of python models can be encouraged if this interface is employed. For further analysis, ways to directly convert existing MATLAB models to python ones is also being researched.

Limitations: This approach does reduce the overhead of using MATLAB but does comes with a few limitations that need to be worked around. Firstly, the use of classes is not permitted with DPI. The debugging of python comes as an overhead as the errors are not provided directly. For this a dummy python wrapper can be used which can be used to call the library python function. If the python model doesn't exist it need to be created and validated.