

Unified Test Writing Framework for Pre and Post Silicon Verification

Rahul Kumar Patel

Pablo Cholbi

Sivasubrahmanya Evani

Raman K

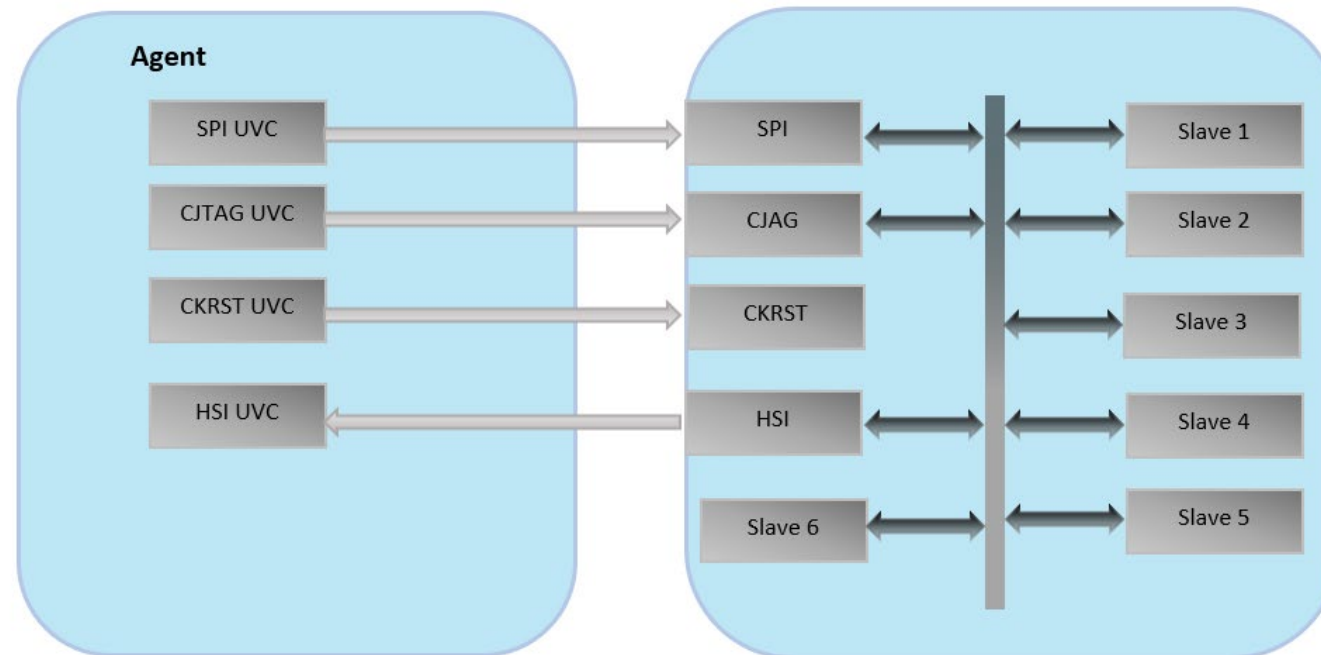


Agenda

- DV Environment overview
- The Journey
- Challenges Faced
- Solutions
- Results

DV Environment Overview

- Top Level view of verification environment



Problem Statement

- Why is this framework needed?
 - P1: Test writing in UVM for non digital folks is a challenge. How do we get them on board (especially for mixed signal simulation)?
 - P2: How can we reuse all/most of the pre silicon infrastructure developed to enable faster post silicon debug?

The Journey

- Problem1: How do we get more people on board ?

```
1 //configure spi
2 spi_cfg 0x01 0x01 ;
3
4 spi_write 0xF001604c 0x04 0x12345678 ;
5 spi_read 0xF001604c 0x04 0x12345678 ;
6
7 force_ivalue_signal `CHIPTOP.muxout_in 0xFFFE ;
8 wait_pin GPIO[1] 0 10 ;
9 delay 20 ;
10 force_ivalue_signal `CHIPTOP.muxout_in 0x01 ;
11 wait_pin GPIO[1] 1 10 ;
```

test.txt

- Pros
 - Simple txt interface, easy to write.
 - Saves Compilation time
- Cons
 - Easy to write but difficult to maintain
 - Vulnerable to regMap changes
 - Difficult to port across products

The Journey

Cont...

- Problem1: How do we get more people on board ?

```
1 #include "basic_op.h"
2 #include "registers.h"
3 #include <stdio.h>
4
5 int main()
6 {
7     setup(); // Configure
8
9     spi_spi_interfaceconfigb.diom = 1;
10    spi_reg_update();
11
12    force_digital("`DIGITAL_TOP.muxout_in", 0xFFFE);
13    wait_state("GPIO[1]", 0, 10);
14    delay_ns(20);
15
16    misc.misc_muxout.muxout_sel = 1;
17    dev_reg_update();
18
19    dev_reg_read_expect(adc_adpll_adpll_stat_addr, 0xC656ADA6);
20
21    cleanup(); //Finish gracefully
22    return(0);
23 }
24
```

test.c

- C wrapper on text based interface
- Bit field Centric
- Pros
 - Easy to write/understand
 - Portable
 - Allows to develop layers of functions
- Cons
 - Unidirectional, no conditional loops/branches

The Journey

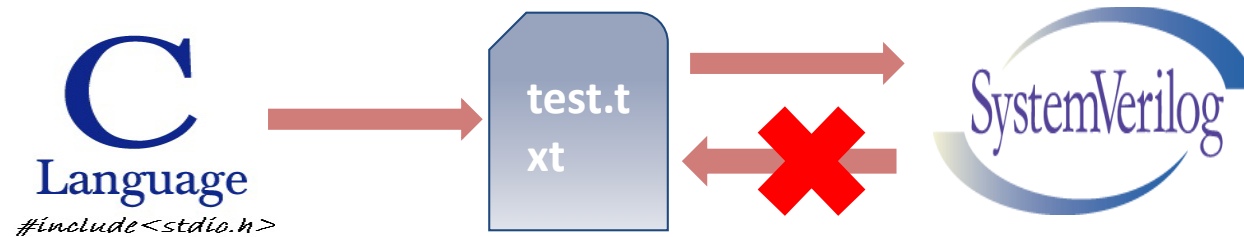
Cont...

- Key Feature Set
 - MMR read/write
 - Memory read/write
 - Analog node voltage probing
 - Force/Release/Wait on digital nodes
 - Waiting for specific voltage on analog node
 - Delay

The Journey

Cont...

- **Recap :**
 - Unidirectional communication

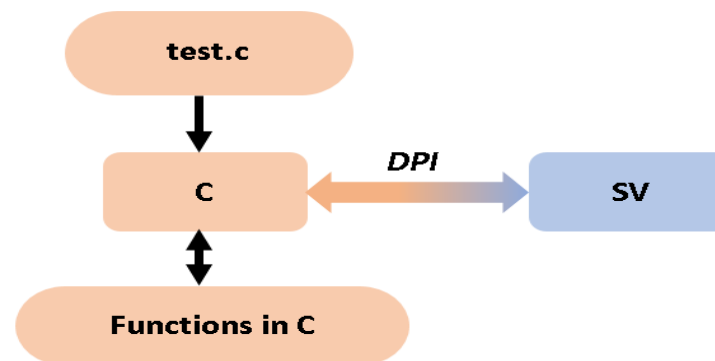
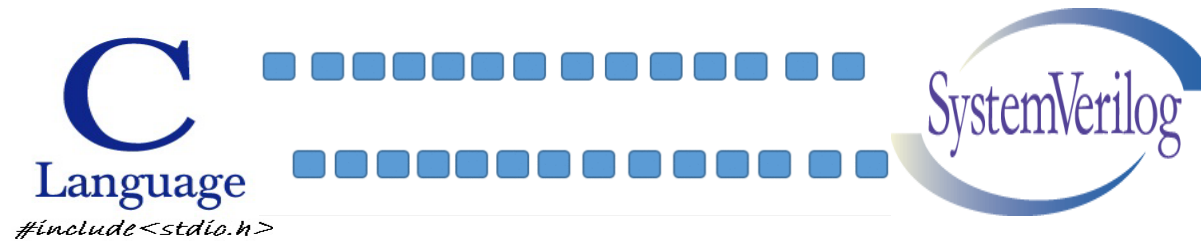


How to mimic the host processor ?

The Journey

Cont...

- **Direct Programming Interface (DPI)**
 - Import of the C functions and export the SV functions



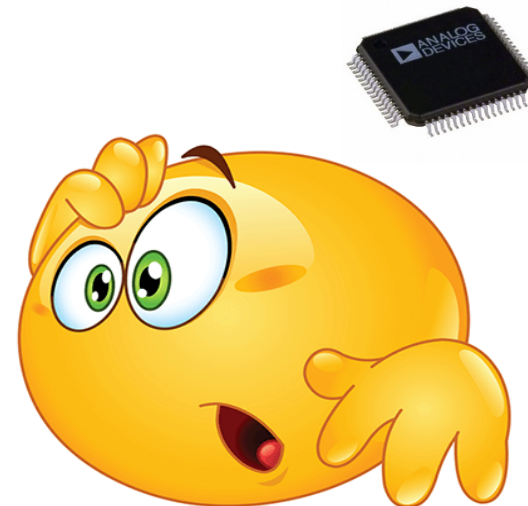
The Journey

Taped Out...!!!



Cont...

Silicon Arrived...!!!



The Journey

Cont...

- Problem2: How can we reuse the pre-silicon infrastructure for post-silicon ?

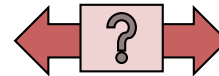
Eval. Engineers:

Develop the test cases in Python



DV Engineers:

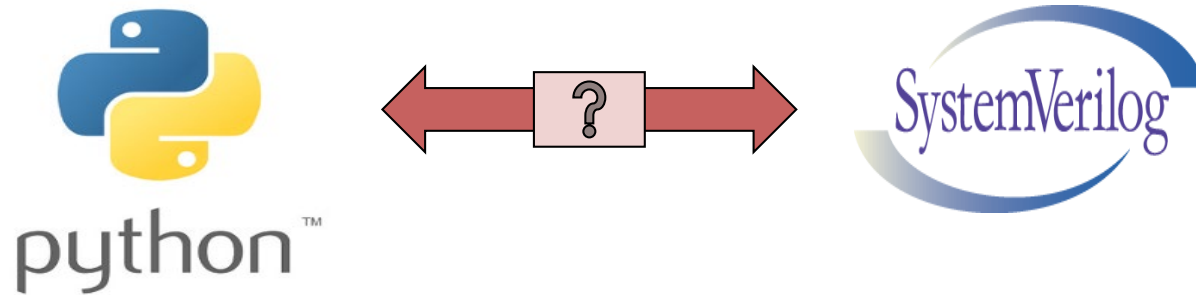
Develop the test cases in C/SV/UVM



- Python being very popular language and lot of open source communities working on it, makes it well suited to communicate with lab equipment
- C is well suited in communicating with the simulators, it is not that well suited to communicate with lab equipment

Challenges

- **Python \Leftrightarrow SV**
 - SV can't communicate directly with Python



Solution

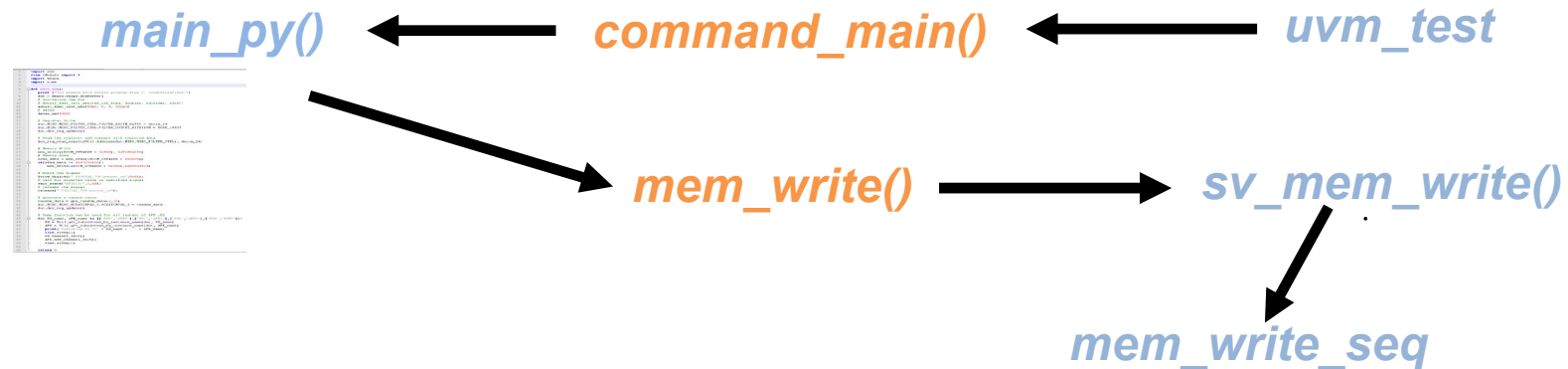
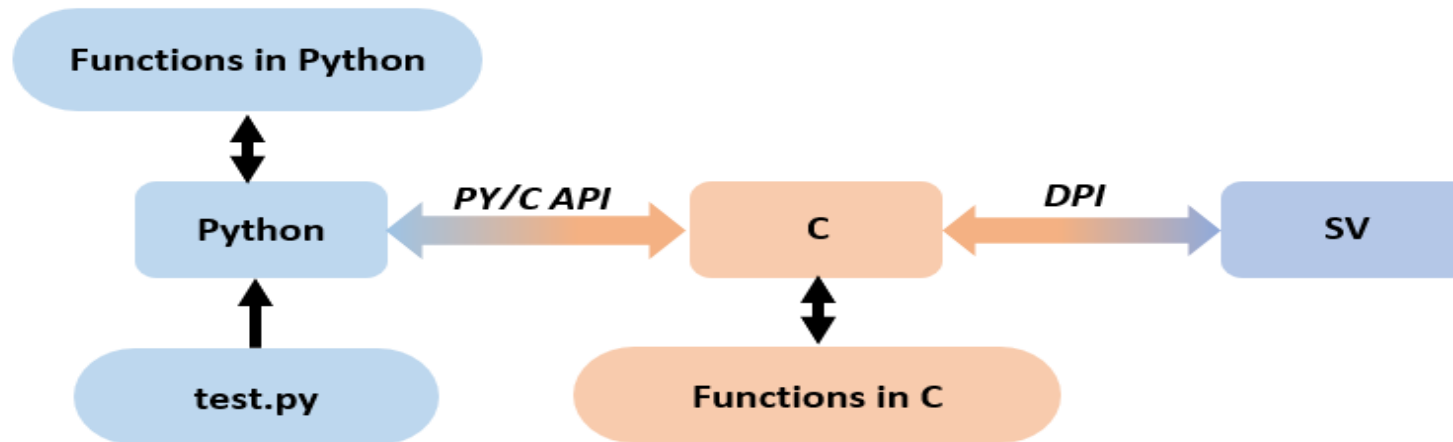
- API-DPI Cascade Bridge
 - **Direct Programming Interface (DPI)** for SystemVerilog–C
 - **Python–C API** for Python–C



- API provides access to the Python interpreter from C code.
 - **Embedding Python:** Inserting calls to Python interpreter into your C application and Calling Python code at specific time
 - **Extending Python:** Python interpreter loads the set of C functions as part of import statement

Solution

Cont...



Embedding Python: Call python method *main_py* of *test.py* from C

```
#include <Python.h>
#include <stdlib.h>
extern void PyInit_CModule();
int command_main() {
    setenv("PYTHONPATH", ".", 1);

    /* Add a built-in module, before Py_Initialize */
    PyImport_AppendInittab("CModule", PyInit_CModule);

    /* Initialize the Python Interpreter */
    Py_Initialize();

    /* Get a reference to the test.main_py function */
    PyObject *pFunc, *u_name, *module;
    PyObject *args;
    PyObject *kwargs;
    PyObject *result = 0;
    int retval;

    /* Get a reference to the test.main_py function */
    u_name = PyUnicode_FromString("test");
    module = PyImport_Import(u_name);
    Py_DECREF(u_name);
    pFunc = PyObject_GetAttrString(module, "main_py");

    /* Make sure we own the GIL(global interpreter
    lock) */
    PyGILState_STATE state = PyGILState_Ensure();

    /* Verify that func is a proper callable */
    if (!PyCallable_Check(pFunc)) {
        fprintf(stderr, "call_func: expected a
        callable\n");
        goto fail;
    }

    /* Build arguments */
    args = Py_BuildValue("()");
    kwargs = NULL;

    /* Call the function */
    result = PyObject_Call(pFunc, args, kwargs);
    Py_DECREF(args);
    Py_XDECREF(kwargs);

    /* Check for Python exceptions (if any) */
    if (PyErr_Occurred()) {
        PyErr_Print();
        goto fail;
    }

    /* Verify the result is a int object */
    if (!PyLong_Check(result)) {
        fprintf(stderr, "call_func: callable
        didn't return a Long\n");
        goto fail;
    }

    /* Create the return value */
    retval = PyLong_AsLong(result);
    Py_DECREF(result);

    /* Restore previous GIL state and return */
    PyGILState_Release(state);

    /* Done */
    Py_DECREF(pFunc);
    Py_Finalize();

    return 0;

fail:
    Py_XDECREF(result);
    abort(); // Change to something more
    appropriate
}
```

Extending Python: Create a module which contain set of C functions

```
//initCModule.c

#include <Python.h>
#include "basic_op.h"

/* This is a wrapper function for C function "mem_write". */
static PyObject* py_mem_write(PyObject* self, PyObject*
args)
{
    uint32_t addr;
    uint32_t data;
    PyArg_ParseTuple(args, "II", &addr, &data);
    // part of basic_op.h which call the SV
    mem_write(addr, data);
    return Py_BuildValue("");
}

/* This is a wrapper function for C function "mem_read". */
static PyObject* py_mem_read(PyObject* self, PyObject* args)
{
    uint32_t return_val;
    uint32_t addr;
    PyArg_ParseTuple(args, "I", &addr);
    // part of basic_op.h which call the SV
    return_val = mem_read(addr);
    return Py_BuildValue("I", return_val);
}

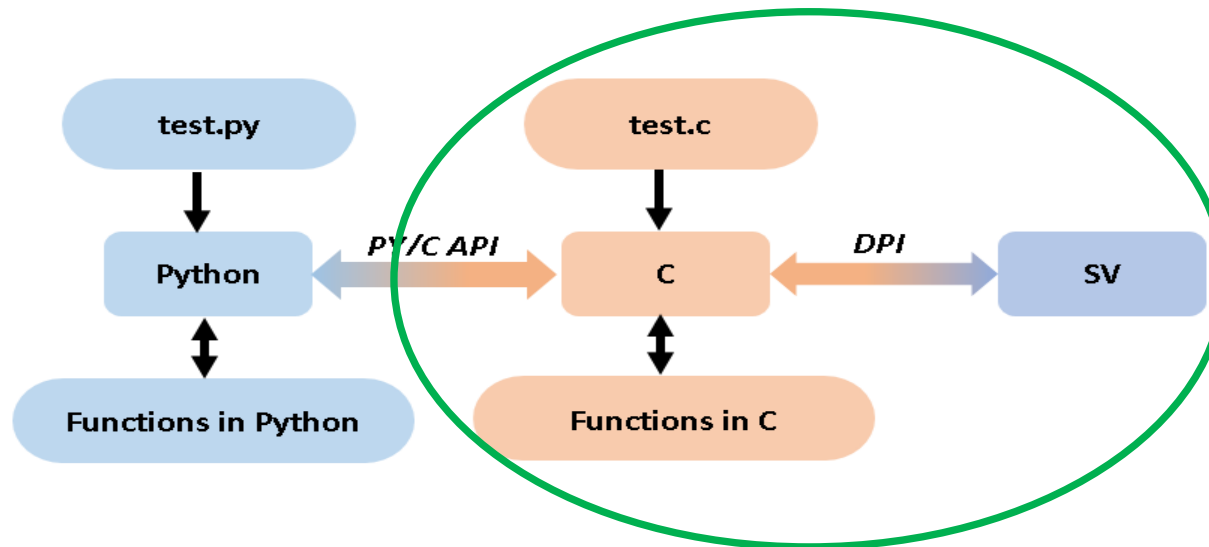
/* Bind Python function names to our C functions */
static PyMethodDef CModule_methods[] = {
    {"mem_write", py_mem_write, METH_VARARGS},
    {"mem_read", py_mem_read, METH_VARARGS},
    {NULL, NULL}
};

#ifdef PY_MAJOR_VERSION >= 3
static struct PyModuleDef moduledef = {
    PyModuleDef_HEAD_INIT, /* m_base */
    "CModule",             /* m_name */
    NULL,                  /* m_doc */
    -1,                    /* m_size */
    CModule_methods        /* m_methods */
};
#endif

/* Python calls this to let us
initialize our module */
PyMODINIT_FUNC
PyInit_CModule(void)
{
    return PyModule_Create(&moduledef);
}
```

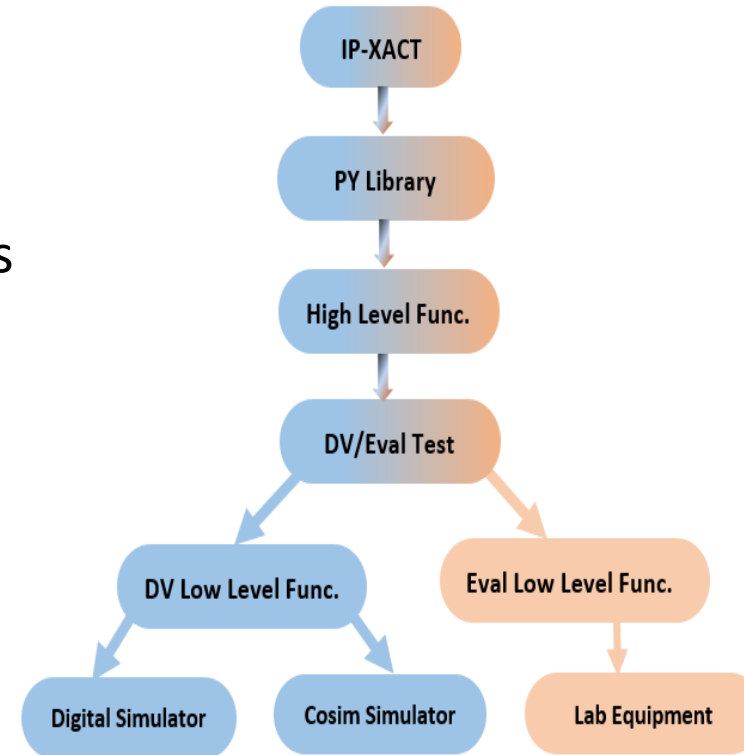

Byproduct

- Pre-Silicon Test suite developed is still valid ...!!!



Framework Overview

- Most of the framework auto dumped form YODA through custom script
- Test cases / functions are reused across all the stage of product development



Sample test Case

test.py

```
1 import sys
2 from CModule import *
3 from chips import Util
4 import Bench
5 import time
6
7 def main_py():
8     print ("You passed this Python program from C! Congratulations!")
9     dut = Bench.chips.ADAR690x()
10    # Initialize the PLL
11    dut.ADC_ADPLL.adcp11_fast_init_gen(80e6, 0, 1, 500e3)
12    # delay
13    delay_ns(1000)
14
15    # Register write
16    dut.MISC.MISC_FILTER_CTRL.FILTER_DECIM_RATIO = 0x20
17    dut.MISC.MISC_FILTER_CTRL.FILTER_OUTPUT_BITWIDTH = 0x1
18    dut.dev_reg_update()
19
20    ## Read the register and compare with expected data
21    dut.comms.read_expect(Util.Address(dut.MISC.MISC_SCRATCHPAD_0), 0x30, 0x04)
22
23    ## Memory Write
24    dut.comms.write((DCCM_STRADDR+0x500), 0x12345678, 0x04)
25
26    ## Memory Read
27    read_data = dut.comms.read((DCCM_STRADDR+0x500), 0x04)
28    if(read_data != 0x87654321):
29        read_data = dut.comms.read((DCCM_STRADDR+0x500), 0x04)
30    print (read_data)
31
32    #Force the signal
33    force_digital("`DIGITAL_TOP.muxout_in",0x01)
34    #wait for expected value on specified signal
35    wait_state("GPIO[1]",1,10)
36    #release the signal
37    release("`DIGITAL_TOP.muxout_in")
38
39    #generate a random value
40    random_data = gen_random_data(1,10)
41    dut.MISC.MISC_SCRATCHPAD_1.SCRATCHPAD_1 = random_data
42    dut.dev_reg_update()
43
44    #Same function can be used for all instand of AFE,RX
45    for RX_name, AFE_name in [['RX0', 'AFE0'], ['RX1', 'AFE1'], ['RX2', 'AFE2'], ['RX3', 'AFE3']]:
46        RX = Util.get_subsystems_by_instance_name(dut, RX_name)
47        AFE = Util.get_subsystems_by_instance_name(dut, AFE_name)
48        RX.channel_init()
49        AFE.afe_channel_init()
50
51    return 0
```

Extending Python

Function declared In C

Register Write

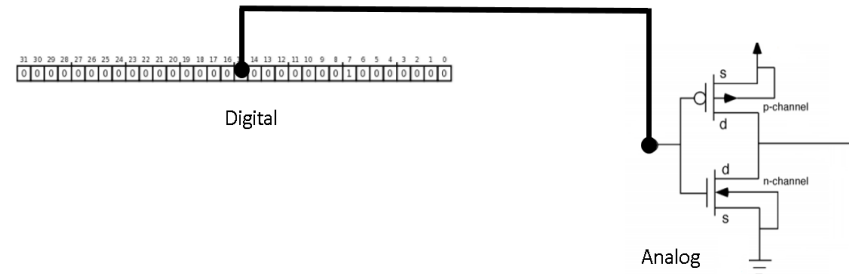
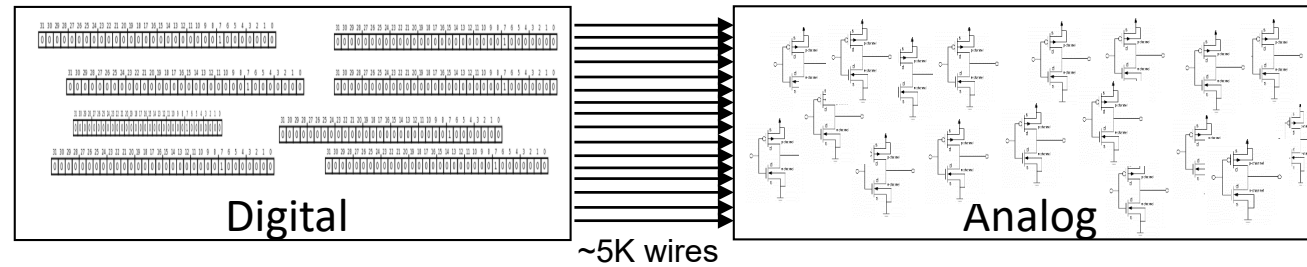
Register Read

Memory Wr/Rd

Force/Wait/Release

Extended Solution

- Automated Checks for the connectivity from digital register bits to the relevant analog nodes in mixed signal simulation.



Extended Solution

Cont...

```
1  import sys
2  from CModule import *
3  from chips import Util
4  import Bench
5  import time
6
7  def main_py():
8
9      dut = Bench.chips.ADAR690x()
10
11      comment("Testing afe0_adc_bias_ctrl0.adc_dac23_ncs_ctrl")
12      # local_hier[] = "<digimmic.die.rx_section.rxchannel0.afe>adc_dac23_ncs_ctrl";
13      signal="<DIGIMMIE.DIE.RX_SECTION.RXCHANNEL0.AFE>ADC_DAC23_NCS_CTRL"
14      comment ("signal="+str(signal)) # FOUND HIERARCHY OF AFE0.ADC_BIAS_CTRL0.ADC_DAC23_NCS_CTRL
15      # Bus check
16      for i in range(2):
17          dut.AFE0.ADC_BIAS_CTRL0.ADC_DAC23_NCS_CTRL = 0x1 << i; dut.dev_reg_update();
18          dut.comms.read_expect(Util.Address(dut.AFE0.ADC_BIAS_CTRL0), 0x1<<(i+24),0x04);
19          probe_bus_expect(signal, 0x1 << i);
20
21      dut.AFE0.ADC_BIAS_CTRL0.ADC_DAC23_NCS_CTRL = 0x0; dut.dev_reg_update();
22      dut.comms.read_expect(Util.Address(dut.AFE0.ADC_BIAS_CTRL0), 0x0,0x04);
23      probe_bus_expect(signal, 0x0);
24
25      comment("Testing afe0_adc_bias_ctrl0.adc_amp3_stg1_bias")
26      # local_hier[] = "<digimmic.die.rx_section.rxchannel0.afe>adc_amp3_stg1_bias";
27      signal="<DIGIMMIE.DIE.RX_SECTION.RXCHANNEL0.AFE>ADC_AMP3_STG1_BIAS"
28      comment ("signal="+str(signal)) # FOUND HIERARCHY OF AFE0.ADC_BIAS_CTRL0.ADC_AMP3_STG1_BIAS
29      # Bus check
30      for i in range(2):
31          dut.AFE0.ADC_BIAS_CTRL0.ADC_AMP3_STG1_BIAS = 0x1 << i; dut.dev_reg_update();
32          dut.comms.read_expect(Util.Address(dut.AFE0.ADC_BIAS_CTRL0), 0x1<<(i+8),0x04);
33          probe_bus_expect(signal, 0x1 << i);
34
35      dut.AFE0.ADC_BIAS_CTRL0.ADC_AMP3_STG1_BIAS = 0x0; dut.dev_reg_update();
36      dut.comms.read_expect(Util.Address(dut.AFE0.ADC_BIAS_CTRL0), 0x0,0x04);
37      probe_bus_expect(signal, 0x0);
38
39      return 0
```

Other benefits

- Functions and test re-use across digital and mixed signal simulation.
- Enables the verification infrastructure of complex data-paths present in the system
 - Using power of Python packages like numpy to compute FFTs etc.
- Enabled system boot-up infrastructure with minimal effort.
- Enabled designers to get involved in test development.
- Accelerating debug, evaluation and demo creation.

Results

- Eval and Apps support
 - Easy debug of configurations issues from Evaluation and Applications
- Tester support
 - UltraFlex tester was used which has a Visual Basic (VB) Front End and all the chip configurations that were developed in Py/C were translated as VB code with minimal effort.

