Unified Test Writing Framework for Pre and Post Silicon Verification

Rahulkumar Patel
Pablo Cholbi
Sivasubrahmanya Evani
Raman K

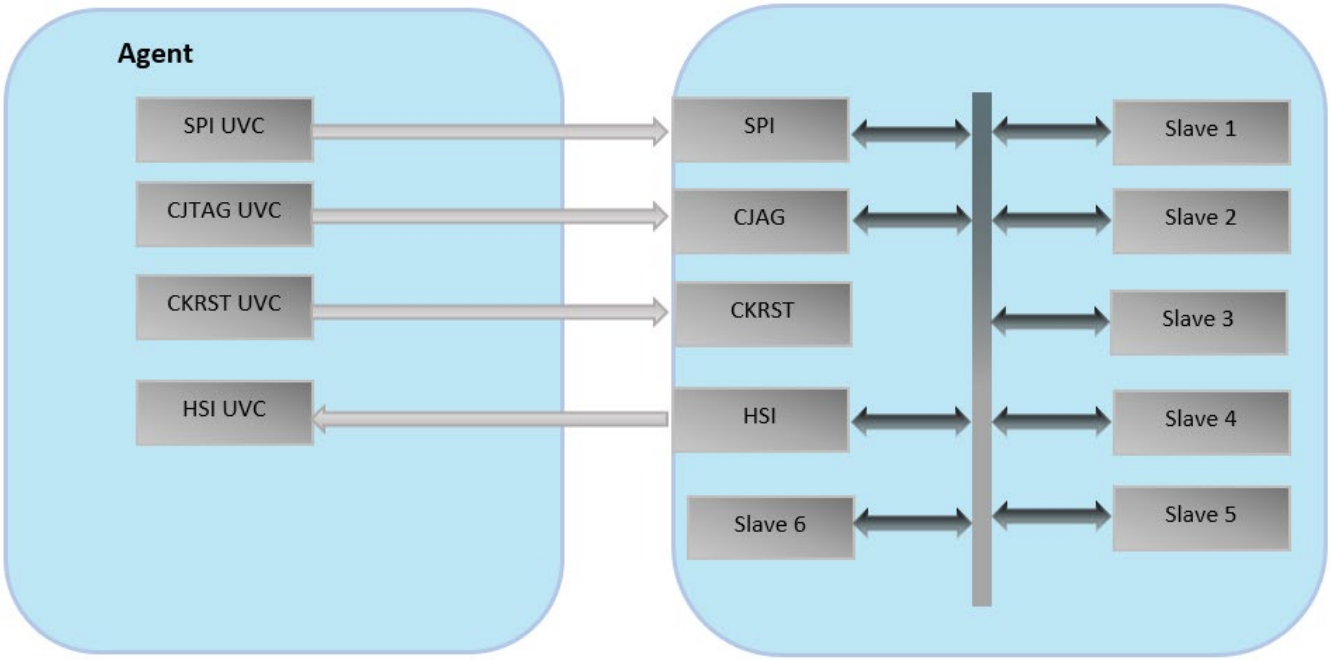
accellera SYSTEMS INITIATIVE™
2019 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION INDIA

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_cnfg 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 ?

```c
#include "basic_op.h"
#include "registers.h"
#include <stdio.h>

int main()
{
setup(); // Configure

spi.spi_interfaceconfigb.diom = 1;
spi_reg_update();

force_digital("`DIGITAL_TOP.muxout_in",0xFFFE);
wait_state("GPIO[1]",0,10);
delay_ns(20);

misc.misc_muxout.muxout_sel = 1;
dev_reg_update();

dev_reg_read_expect(adc_adpll_adpll_stat_addr,0xC656ADA6);

cleanup(); //Finish gracefully
return(0);
}
```

test.c

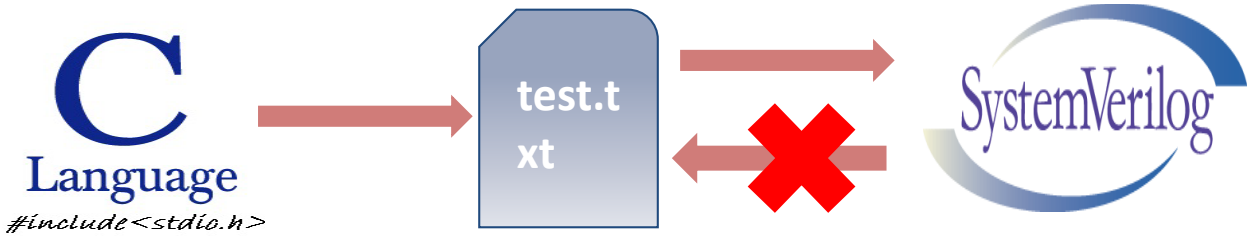
- C wrapper on text based interface
- Bit field Centric
- Pros
  - Easy to write/understand
  - Portable
  - Allows to develop layers of functions
- Cons
  - Unidirectional, no conditional loops/branches

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…

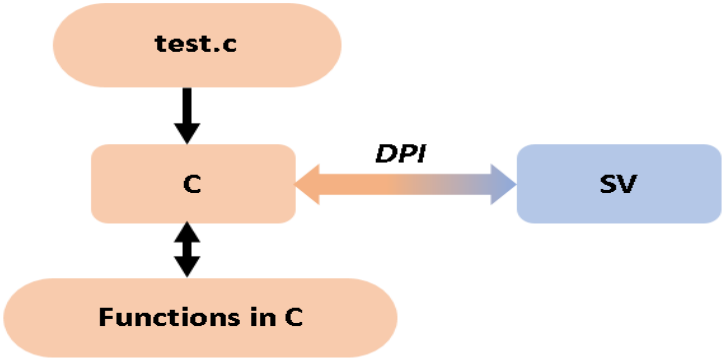
- **Recap** :
  - Unidirectional communication



How to mimic the host processor ?

The Journey Cont…

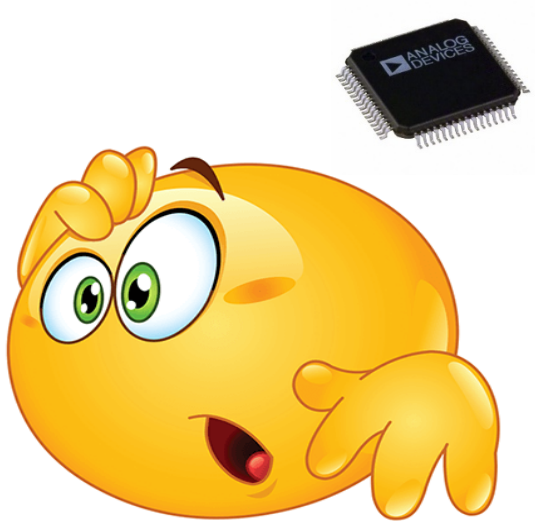
- **Direct Programming Interface (DPI)**
  - Import of the C functions and export the SV functions



The Journey Cont…

Taped Out…!!!

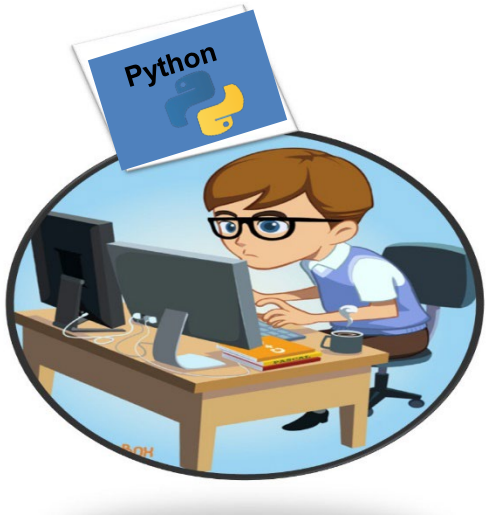
Silicon Arrived…!!!



The Journey Cont…

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

Eval. Engineers:
Develop the test cases in Python



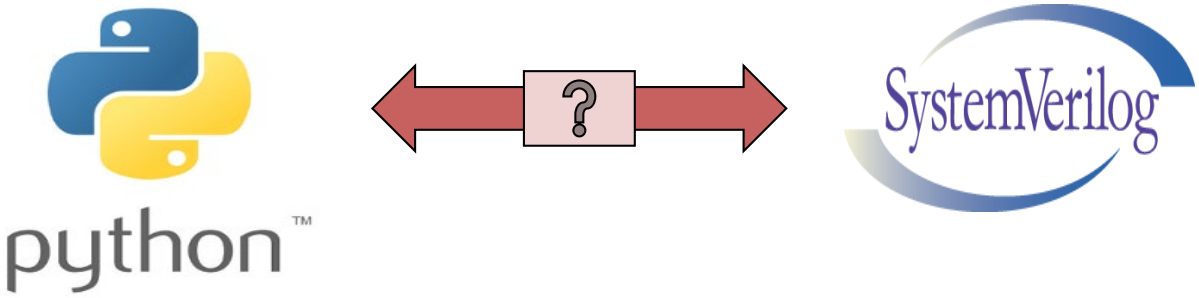
- Python being very popular language and lot of open source communities working on it, makes it well suited to communicate with lab equipment

DV Engineers:
Develop the test cases in C/SV/UVM

- C is well suited in communicating with the simulators, it is not that well suited to communicate with lab equipment

Challenges

- **Python <=> SV**
  - SV can’t communicate directly with Python

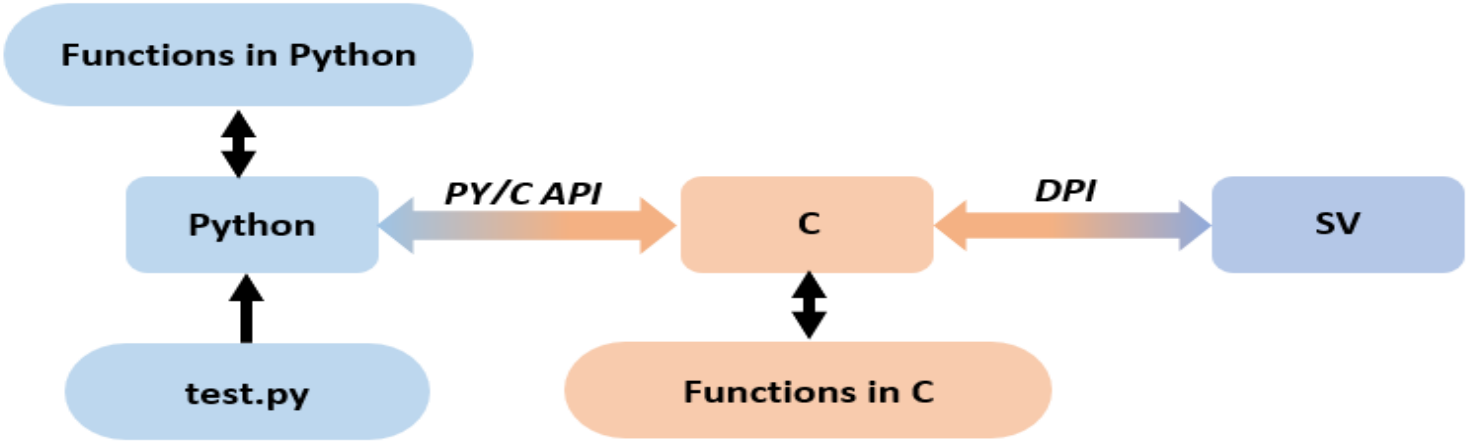


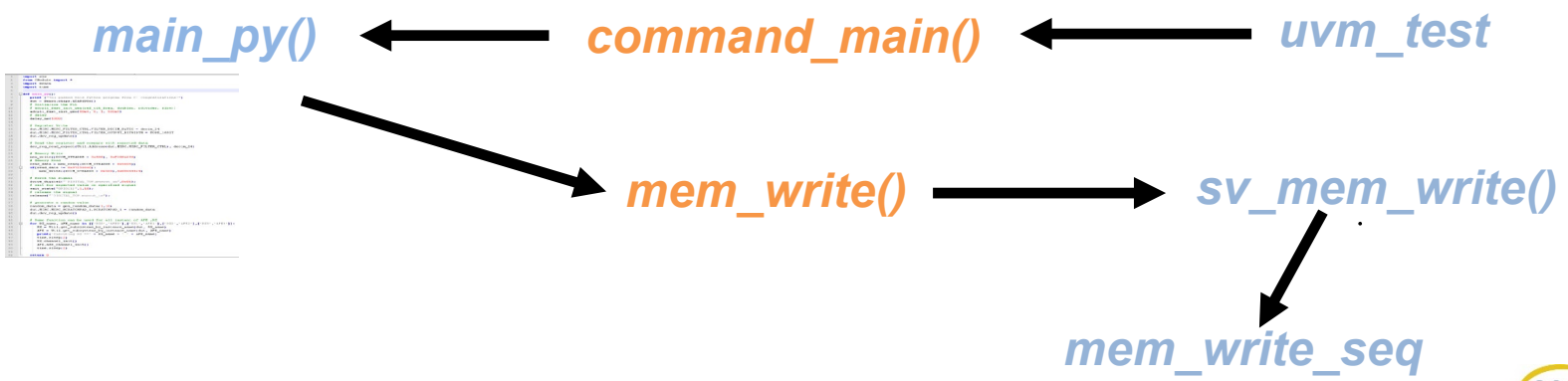
Solution

- 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

```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;
```

```c
/* 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
}
```

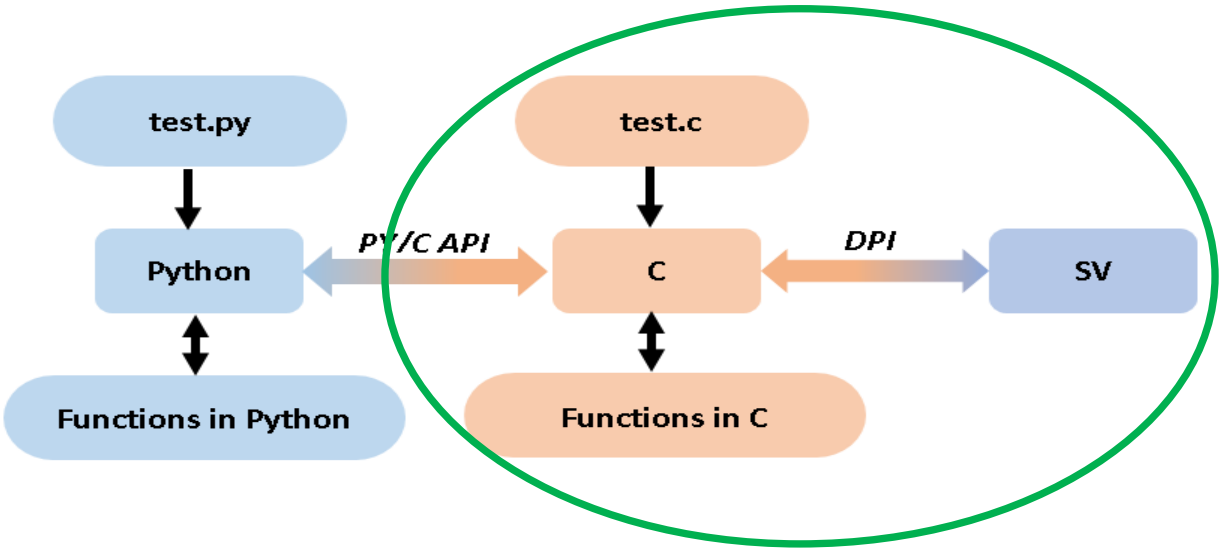
Extending Python: Create a module which contain set of C functions

```c
//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}
};
```

```c
#if 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);
}
```

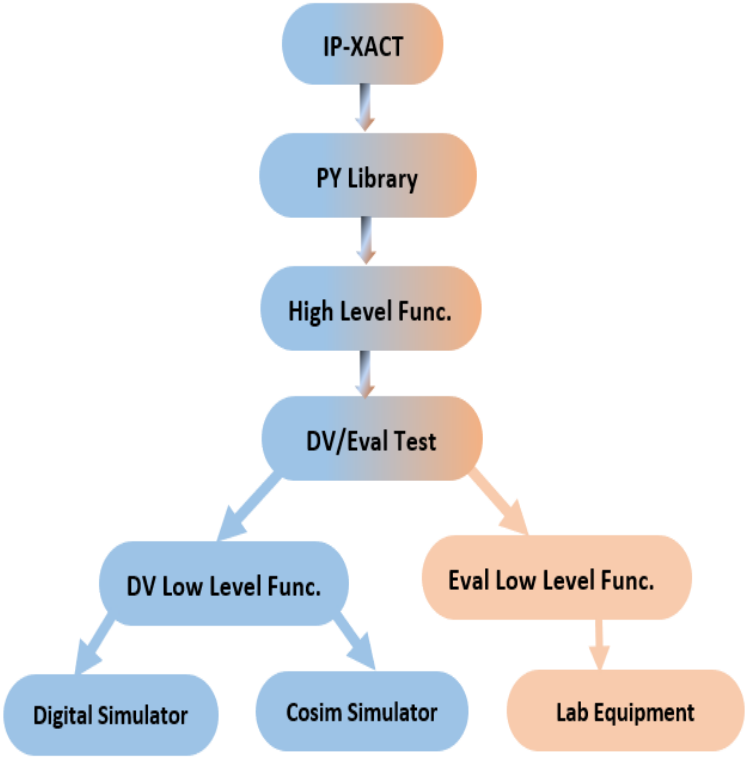
Byproduct

- 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

```python
import sys
from CModule import *
from chips import Util
import Bench
import time

def main_py():
    print ("You passed this Python program from C! Congratulations!")
    dut = Bench.chips.ADAR690x()
    # Initialize the PLL
    dut.ADC_ADPLL.adcpll_fast_init_gen(80e6, 0, 1, 500e3)
    # delay
    delay_ns(1000)

    # Register write
    dut.MISC.MISC_FILTER_CTRL.FILTER_DECIM_RATIO = 0x20
    dut.MISC.MISC_FILTER_CTRL.FILTER_OUTPUT_BITWIDTH = 0x1
    dut.dev_reg_update()

    ## Read the register and compare with excepted data
    dut.comms.read_expect(Util.Address(dut.MISC.MISC_SCRATCHPAD_0), 0x30, 0x04)

    ## Memory Write
    dut.comms.write((DCCM_STRADDR+0x500), 0x12345678, 0x04)

    ## Memory Read
    read_data = dut.comms.read((DCCM_STRADDR+0x500), 0x04)
    if(read_data != 0x87654321):
        read_data = dut.comms.read((DCCM_STRADDR+0x500), 0x04)
    print (read_data)

    #force the signal
    force_digital("`DIGITAL_TOP.muxout_in",0x01)
    #wait for expected value on specificed signal
    wait_state("GPIO[1]",1,10)
    #release the signal
    release("`DIGITAL_TOP.muxout_in")

    #generate a random value
    random_data = gen_random_data(1,10)
    dut.MISC.MISC_SCRATCHPAD_1.SCRATCHPAD_1 = random_data
    dut.dev_reg_update()

    #Same function can be used for all instand of AFE,RX
    for RX_name, AFE_name in [['RX0','AFE0'],['RX1','AFE1'],['RX2','AFE2'],['RX3','AFE3']]:
        RX = Util.get_subsystems_by_instance_name(dut,RX_name)
        AFE = Util.get_subsystems_by_instance_name(dut,AFE_name)
        RX.channel_init()
        AFE.afe_channel_init()

    return 0
```

Extending Python

Function declared In C

Register Write

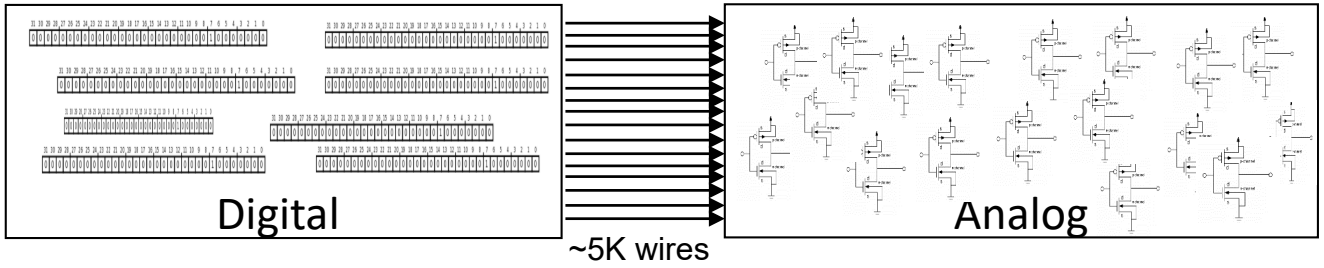
Register Read

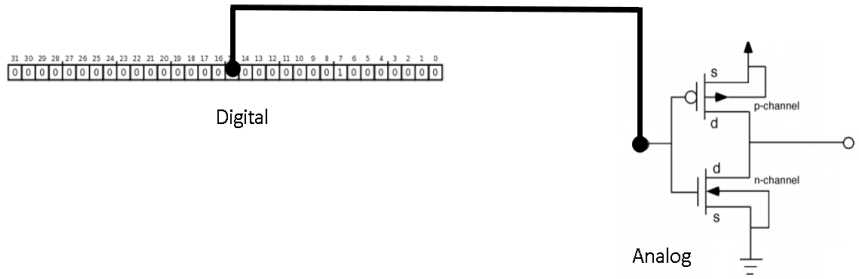
Memory Wr/Rd

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…

```
import sys
from CModule import *
from chips import Util
import Bench
import time

def main_py():

    dut = Bench.chips.ADAR690x()

    comment("Testing afe0_adc_bias_ctrl0.adc_dac23_ncs_ctrl")
    # local_hier[] = "<digimmic.die.rx_section.rxchannel0.afe>adc_dac23_ncs_ctrl";
    signal="<DIGIMMIC.DIE.RX_SECTION.RXCHANNEL0.AFE>ADC_DAC23_NCS_CTRL"
    comment ("signal="+str(signal)) # FOUND HIERARCHY OF AFE0.ADC_BIAS_CTRL0.ADC_DAC23_NCS_CTRL
    # Bus check
    for i in range(2):
        dut.AFE0.ADC_BIAS_CTRL0.ADC_DAC23_NCS_CTRL = 0X1 << i; dut.dev_reg_update();
        dut.comms.read_expect(Util.Address(dut.AFE0.ADC_BIAS_CTRL0), 0x1<<(i+24),0x04);
        probe_bus_expect(signal, 0x1 << i);

    dut.AFE0.ADC_BIAS_CTRL0.ADC_DAC23_NCS_CTRL = 0X0; dut.dev_reg_update();
    dut.comms.read_expect(Util.Address(dut.AFE0.ADC_BIAS_CTRL0), 0x0,0x04);
    probe_bus_expect(signal, 0x0);

    comment("Testing afe0_adc_bias_ctrl0.adc_amp3_stg1_bias")
    # local_hier[] = "<digimmic.die.rx_section.rxchannel0.afe>adc_amp3_stg1_bias";
    signal="<DIGIMMIC.DIE.RX_SECTION.RXCHANNEL0.AFE>ADC_AMP3_STG1_BIAS"
    comment ("signal="+str(signal)) # FOUND HIERARCHY OF AFE0.ADC_BIAS_CTRL0.ADC_AMP3_STG1_BIAS
    # Bus check
    for i in range(2):
        dut.AFE0.ADC_BIAS_CTRL0.ADC_AMP3_STG1_BIAS = 0X1 << i; dut.dev_reg_update();
        dut.comms.read_expect(Util.Address(dut.AFE0.ADC_BIAS_CTRL0), 0x1<<(i+8),0x04);
        probe_bus_expect(signal, 0x1 << i);

    dut.AFE0.ADC_BIAS_CTRL0.ADC_AMP3_STG1_BIAS = 0X0; dut.dev_reg_update();
    dut.comms.read_expect(Util.Address(dut.AFE0.ADC_BIAS_CTRL0), 0x0,0x04);
    probe_bus_expect(signal, 0x0);

    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
  - UltrFlex 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.