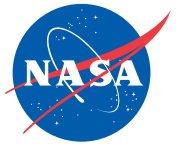


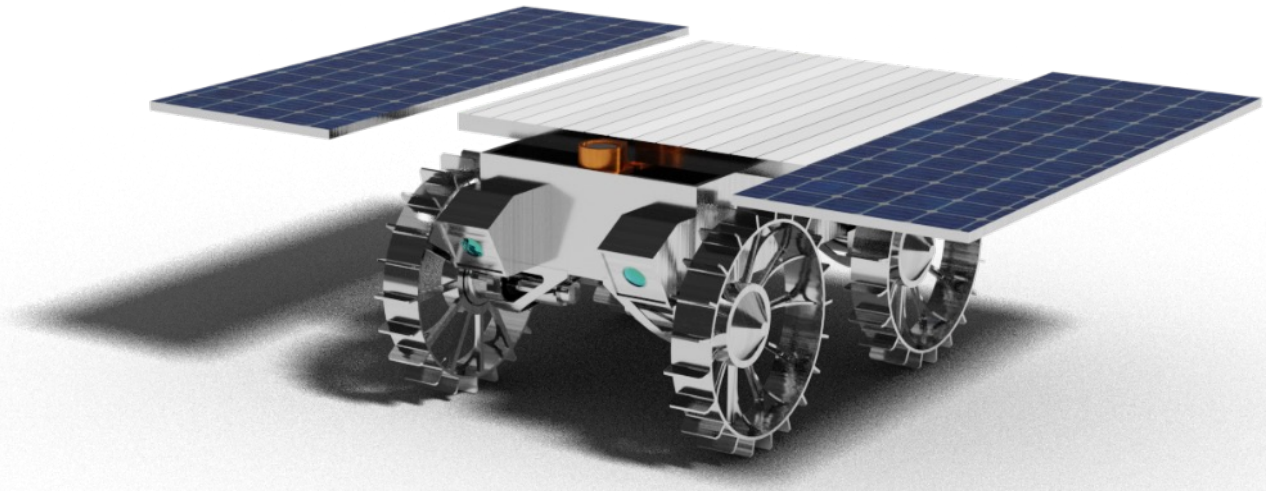
Jet Propulsion Laboratory
California Institute of Technology

RTL to Software Model using COSIDE® and SystemC

COSEDA User Group 2022

Ashot Hambardzumyan
FPGA Engineer

Nov 24th, 2022

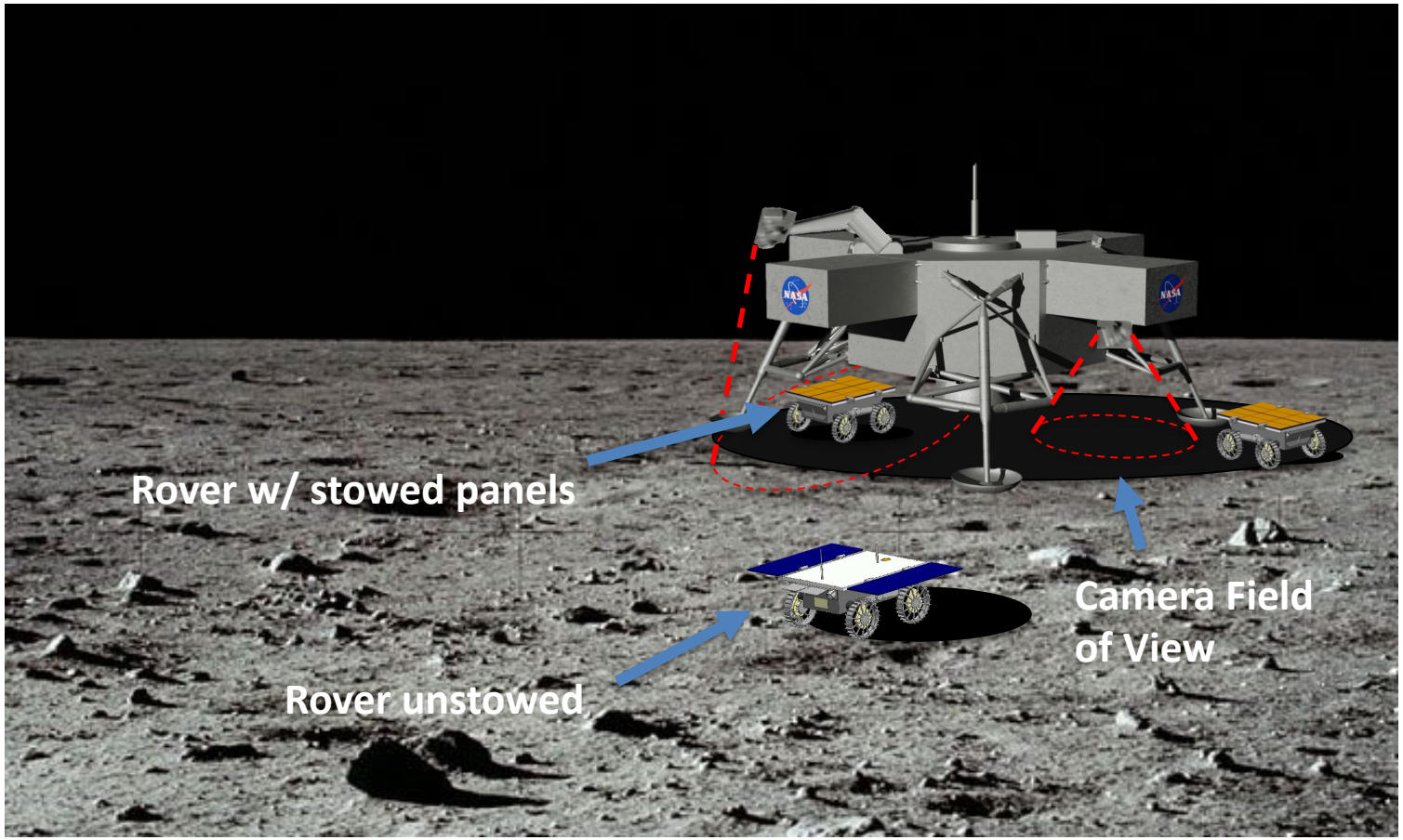


CADRE: Cooperative Autonomous Distributed Robotic Exploration

Agenda

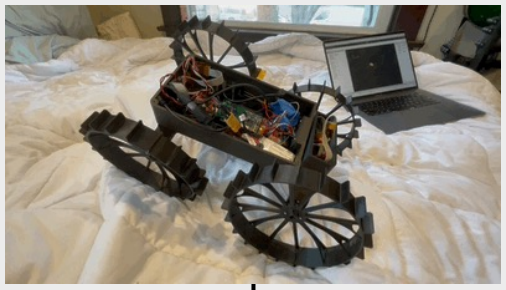
1. CADRE Mission
2. CADRE FPGA Architecture
3. Current process for SW-HW co-development
4. Verilator Support in COSIDE
5. Software Modeling using IPC
6. Software Modeling using SystemC

- CADRE is a flight technology demonstration to demonstrate multi-agent cooperative autonomy, performing distributed measurement, on the surface of the moon.



CADRE Through the Ages

Mar 19, ’21
Proto-Merc7 is Born

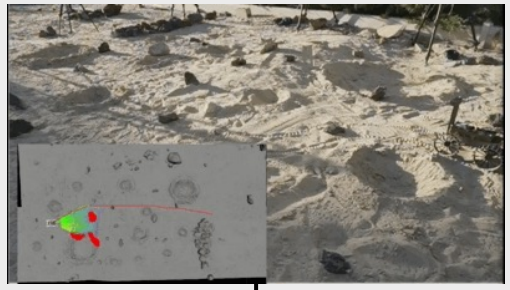


April 14, ’21
Mini-Mars Yard Rocks

June 3, ’21
Driving in GRC1 @JPL

July 23, ’21
Multi-Agent Drive (open)

Oct 18, ‘21
Autonomy



Nov 4, ’21
GRC Slope Lab Testing

May 2, ’22
Autonomy, Exploration

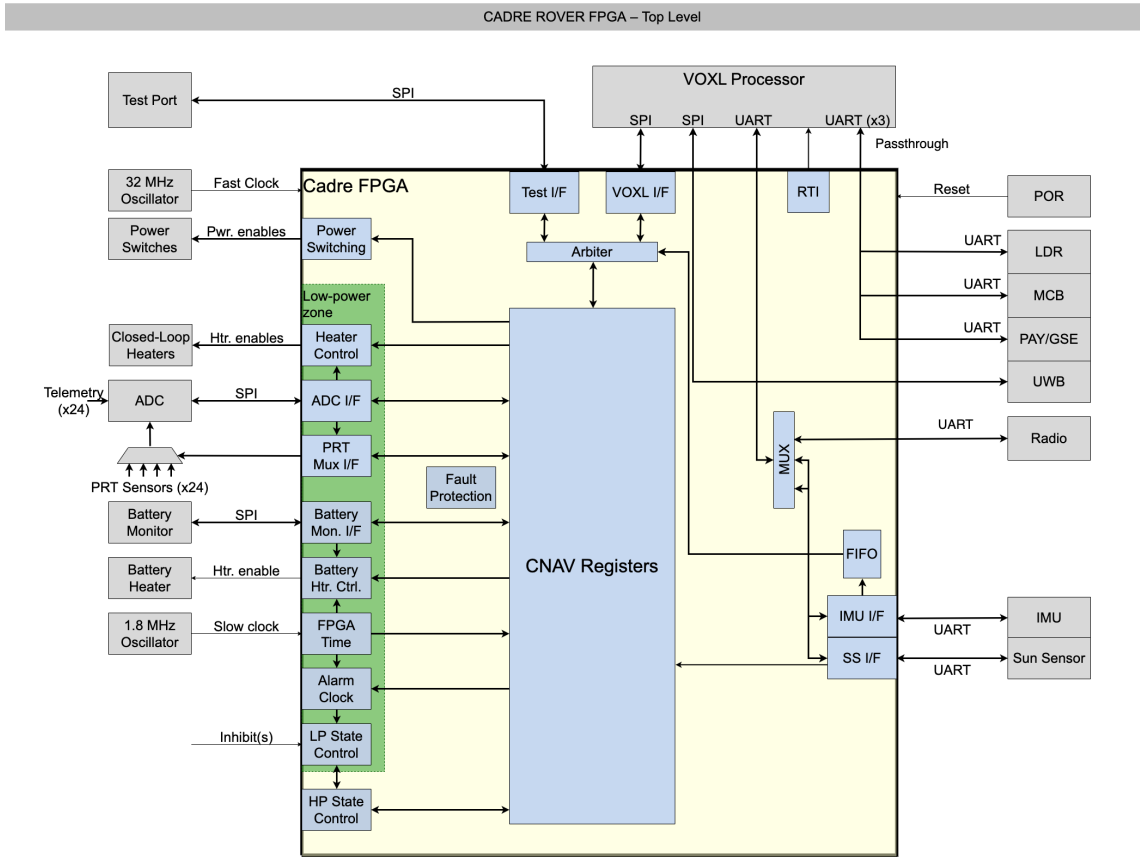
March ’22
Dragon Farm

Oct, ’22
Formation Driving

FPGA Architecture

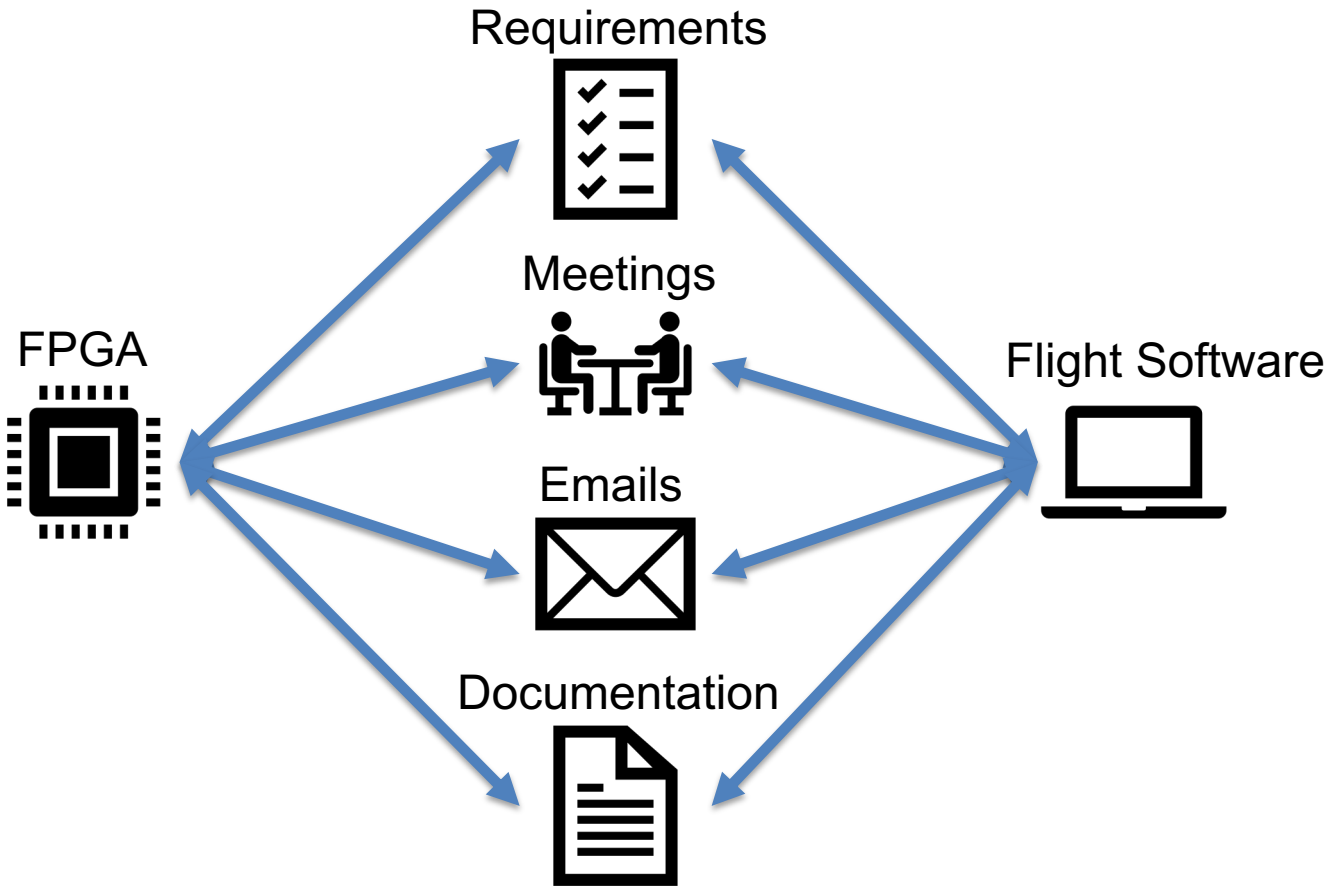
Features:

- Enforced Power States
- Watchdogs, Runout Timer, Alarm Clock
- Closed-Loop Survival Heater Control
- IMU Driver
- Sun Sensor Driver
- Fault Protection
- Low Power core



Current Process

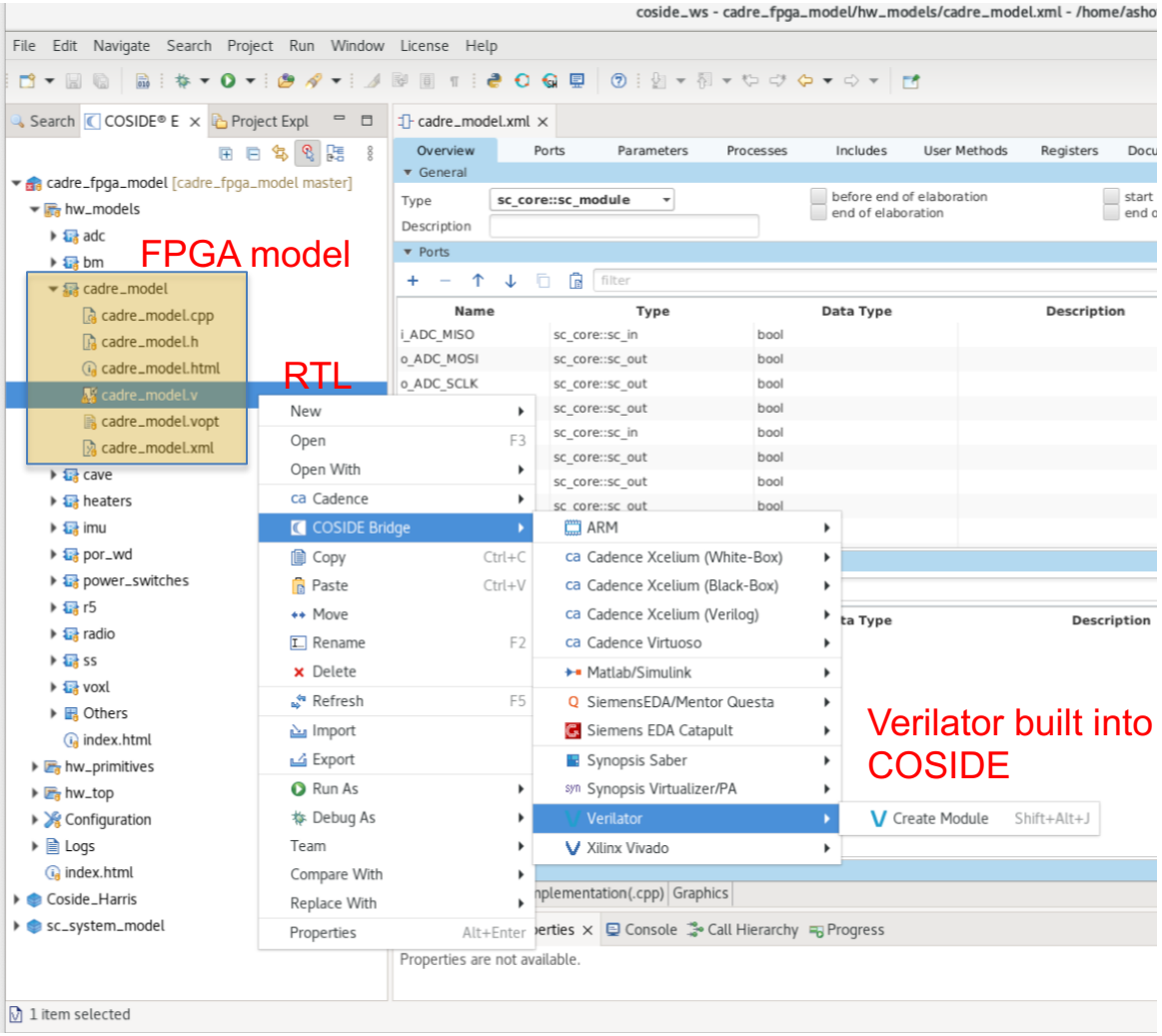
How FPGA and Flight Software co-development communicate today



Shortcomings:

- Documents come out of sync
- Written description can be interpreted differently
- Software has to build an FPGA model to test with
- Integration bugs won't be found until hardware is ready to run the software.

COSIDE



Steps:

1. Add the synthesizable Verilog RTL to your COSIDE project
2. Right click the Verilog file and navigate to COSIDE Bridge > Verilator > Create Model
3. COSIDE module will be created
4. Create other components required in your testbench either by importing or coding

- COSIDE Generates SystemC code from RTL

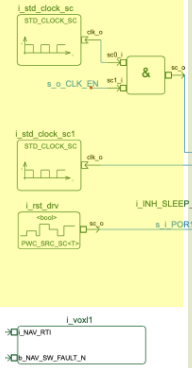
```
module Convert;
  input clk
  input [31:0] data;
  output [31:0] out;

  always @ (posedge clk)
    out <= data;
endmodule
```

```
#include "verilated.h"

class Convert {
  bool    clk;
  uint32_t  data;
  uint32_t  out;

  void eval();
}
```

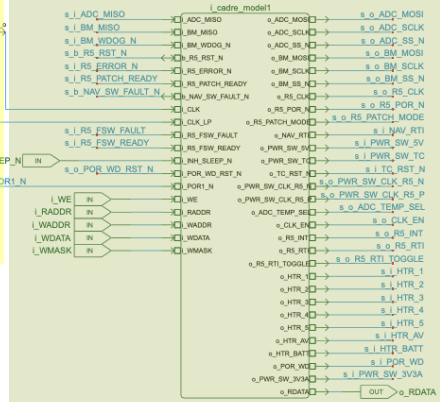
COSIDE TB

Testbench

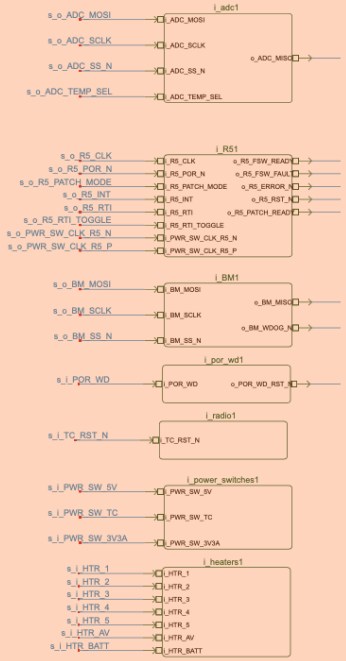
COSIDE Primitives



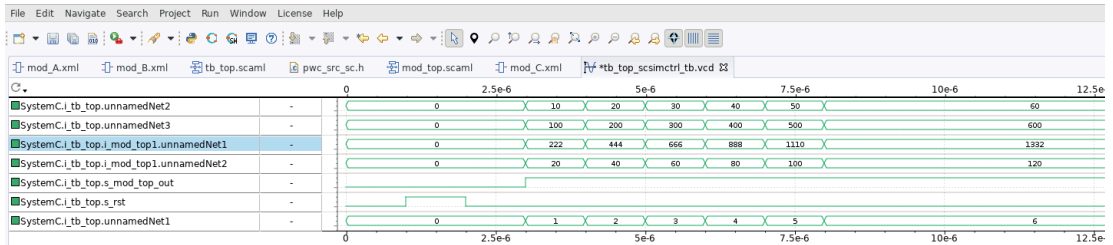
CADRE FPGA Model



Board component modeled in SystemC



Simulation



Steps:

1. Create a TB with peripheral device models
2. Simulate to verify proper functionality

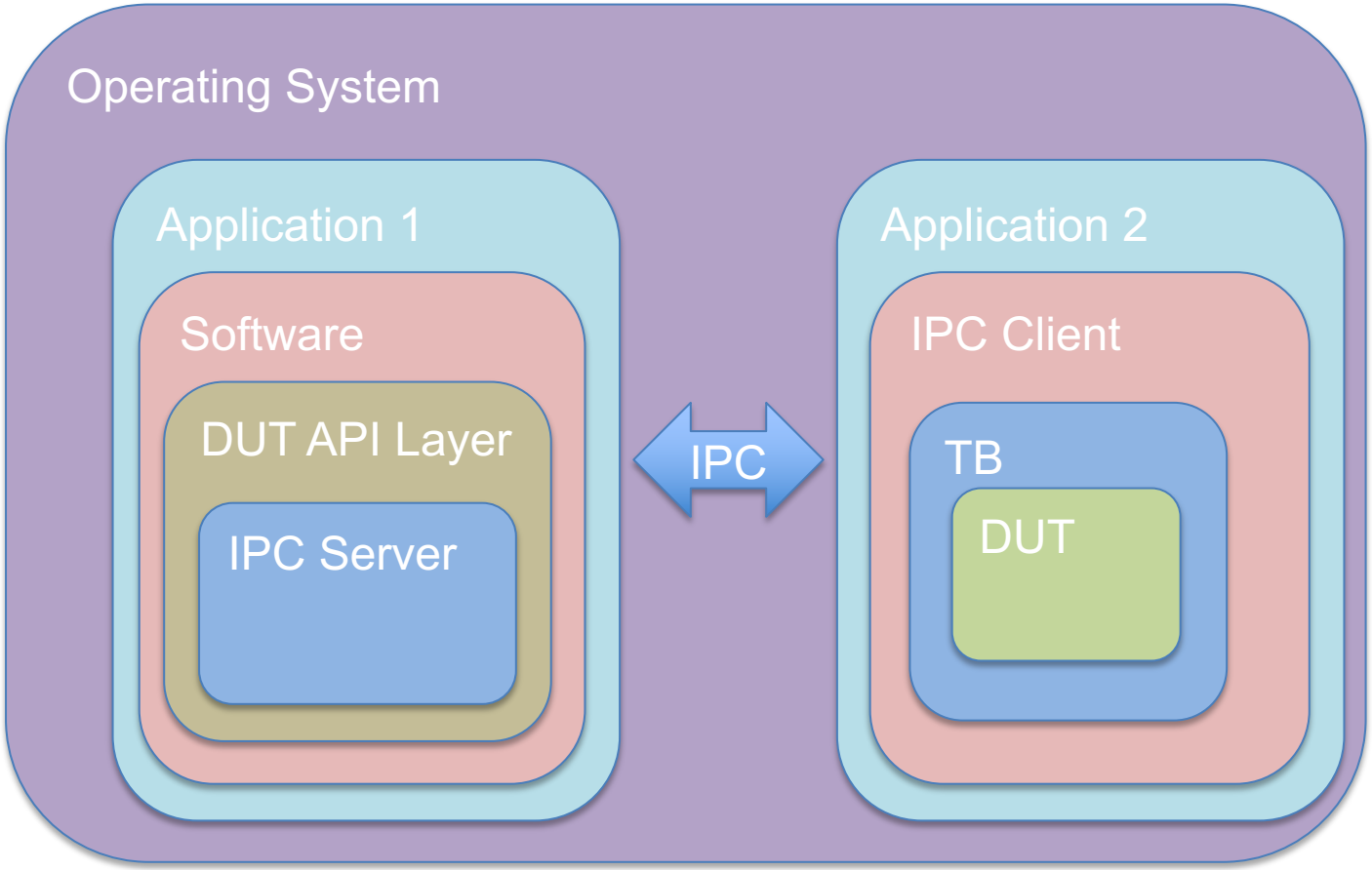
Software Model (IPC method)

Pros:

1. No library dependence for Application 2 since it's compiled on COSIDE.
2. Very easy to setup. COSIDE automatically finds all input/output ports in the TB
3. Application 1 can be in any language. COSIDE provides IPC Server in Python and C++

Cons:

1. COSIDE IPC only supports wiggling TB ports. (signal level communication)
2. Need to poll the DUT, can't create event triggers
3. Expect slowdown due to IPC overhead
4. API Layer falls on Software side



Software Model (IPC method)

How To:

1. Setup SystemC Client

```
////////////////////////////////////////////////////////////////////////////
/// configure to connect as Client to C-main                              //
////////////////////////////////////////////////////////////////////////////
cos_mat_engine_namespace::cos_systemc_client hdl;
if(hdl.attach())
{
        SC_REPORT_ERROR("top_simple_tb","Failed to attach to C-main master");
}

hdl.start();
```

2. Setup Server (Python shown, C++ is similar)

```
########################################
print("\nSetup connection...")
hdl=sc.cos_systemc_handle()
hdl.open("top_level_cadre_ipc.exe")
```

3. Run. Read all Registers

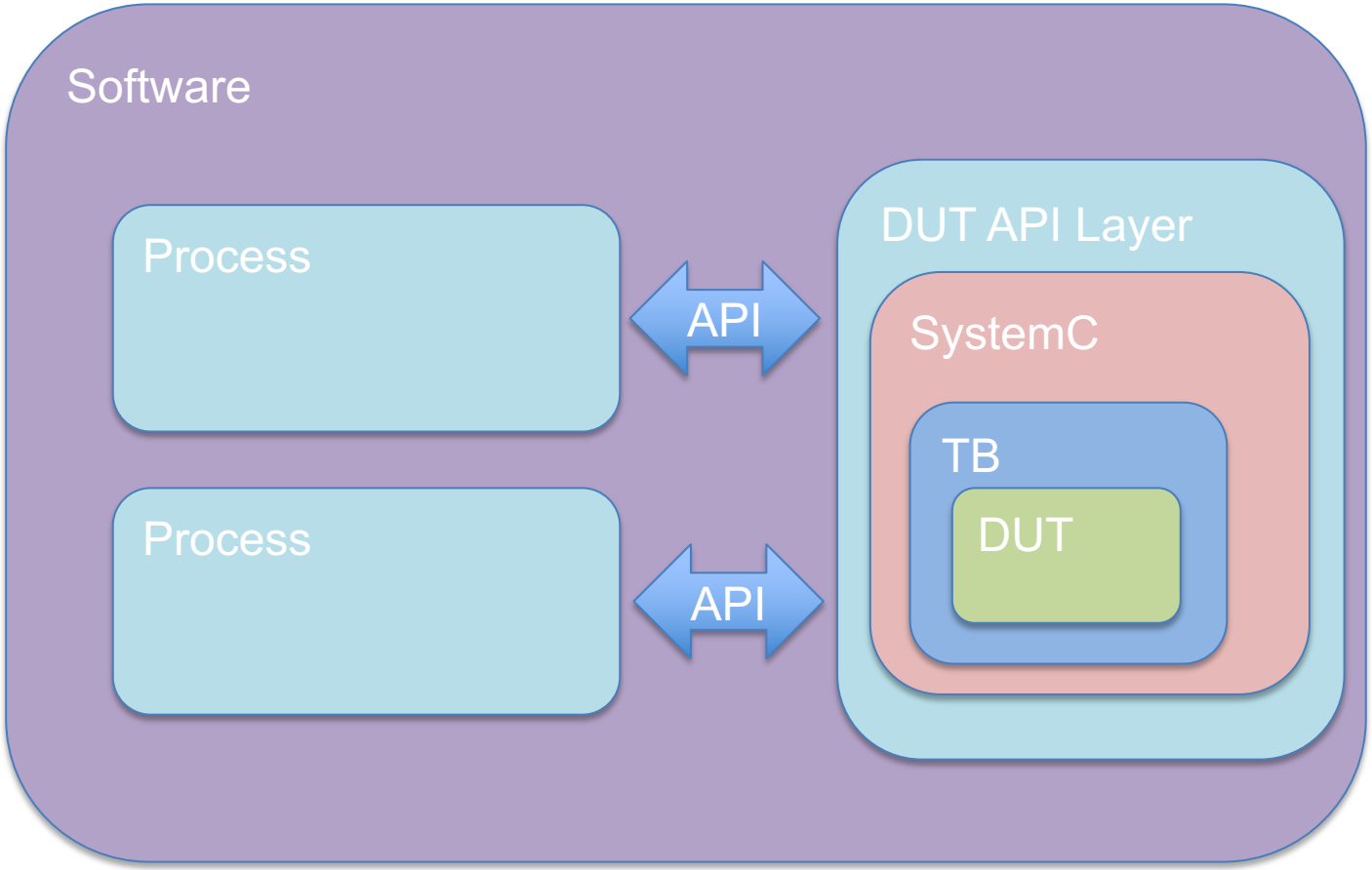
```
print("\nStart simulation...")
# create quick access handle
i_RADDR_s = hdl.get_object_handle("i_RADDR_s")
i_WADDR_s   = hdl.get_object_handle("i_WADDR_s")
i_WDATA_s = hdl.get_object_handle("i_WDATA_s")
i_WE_s   = hdl.get_object_handle("i_WE_s")
i_WMASK_s = hdl.get_object_handle("i_WMASK_s")
i_INH_SLEEP_N_s   = hdl.get_object_handle("i_INH_SLEEP_N_s")
o_RDATA_s = hdl.get_object_handle("o_RDATA_s")

# write i_INH_SLEEP_N using handle
hdl.write(i_INH_SLEEP_N_s, "0")

hdl.run(0.250)     # run simulation for 250ms
print("Current time: %3.2e" %hdl.get_time_in_seconds())

for i in range(20):
    #read reg
    hdl.write(i_RADDR_s, str(i))
    hdl.run(10*1e-6/32.0+1e-12); #run 10 clock cycles plus small delta

    val = hdl.read(o_RDATA_s)
    print("Read Address is {:x} Read Data is {:x}".format(i, int(val)))

hdl.run(1) #run for 1s
```

Software Model (RAW SystemC)

Pros:
1. No IPC overhead
2. Software has access to source code
3. Can react to DUT events
4. More control over TB

Cons:
1. Requires some trickery of SystemC Main
2. Software has to resolve SystemC and other hardware library dependencies

Performance: 1 sec sim = 1 min wall clock



Software Model (RAW SystemC)

Read a register from the FPGA model

```
////////////////////////////////////////////////////////////////////////////
/// time domain simulation                                                //
////////////////////////////////////////////////////////////////////////////
cadre_sw_model i_cadre_sw_model; //create the SW Model of FPGA

int rdata;
i_cadre_sw_model.start_sim();
i_cadre_sw_model.set_sleep_inh(true);  //Force Wake up
i_cadre_sw_model.run_clk(10000);       //Run
rdata = i_cadre_sw_model.read_reg(0);  //Read FPGA ID:  11668590
cout<<"FPGA Version Register is expected:0x11668590 actual:"<<std::hex<<rdata<<endl;
```

Set the battery temperature in Battery Monitor model

```
// set BM return value
// Battery temperatures
// These are 12-bit DN values corresponding to specific battery temperatures.
// The values come from Bob's MathCad tool.
i_cadre_sw_model.i_components->i_BM1.ext_temp1 = BM_TEMP_POS30C;
// Read BM
cout<<"BM Temps"<<endl;
for (int i = 0x13; i < 0x15; i++)
{
    rdata = i_cadre_sw_model.read_reg(i);
    cout << "Register address:0x" << std::hex << i << " data:0x" << rdata << endl;
}
```

APIs:

- Read

```
int cadre_sw_model::read_reg(uint address){
        i_CNAV_RADDR.write(address);
        run_clk(10);
        int rdata = o_CNAV_RDATA.read();
        return rdata;
}
```

- Write

```
void cadre_sw_model::write_reg(uint address, int data){
        i_CNAV_WADDR.write(address);
        i_CNAV_WMASK.write(0xffffffff);
        i_CNAV_WDATA.write(data);
        i_CNAV_WE.write(true);
        run_clk(1);
        i_CNAV_WE.write(false);
        run_clk(50);
}
```

- Run Clock

```
void cadre_sw_model::run_clk(uint cycles){
        double one_clk_period = 1e-6/32.0;
        double sim_time = cycles * one_clk_period + 1e-12;
        sc_core::sc_start(sim_time, sc_core::SC_SEC);

}
```