Firmware bring-up and integration on SystemC VP with COSIDE

COSEDA User Group Meeting 2022

Ana-Maria Plaiseanu(IFRO DCBUC ATV SC D DD)

Agenda

- Background and motivation
- Workflow
  - Block diagram
  - Concept feasibility VP development
  - Firmware integration
  - Testing concept
  - Regression results
  - Benefits

Background and motivation

Background

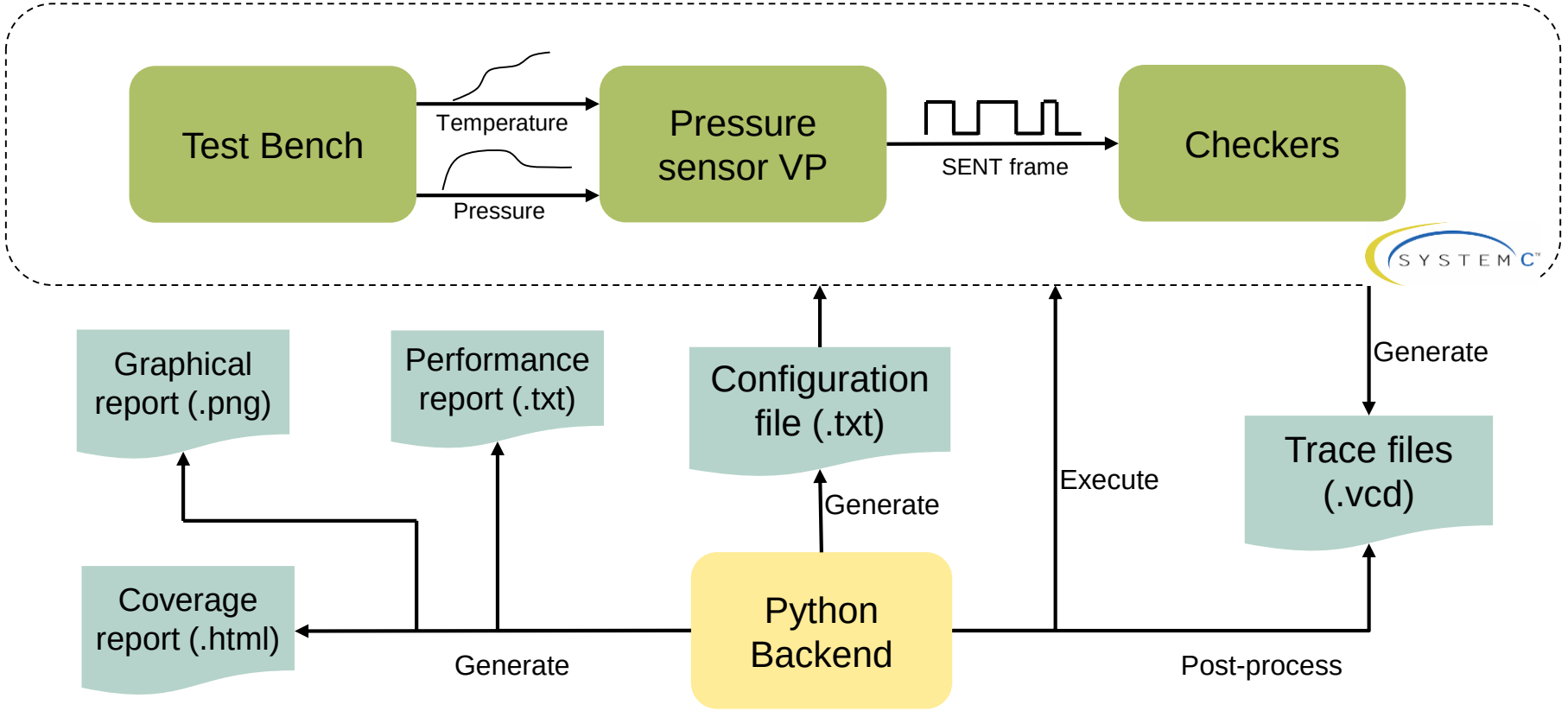
- First step – design SystemC virtual prototype with a Python backend for concept feasibility
  - Performance-driven architectural choices during early chip concept stage
- Next step - refinement of model used in concept feasibility for design team
  - Supports building the design from bottom up - high complexity systems split into smaller, more manageable pieces (modules)

Background and motivation

Motivation of FW developer

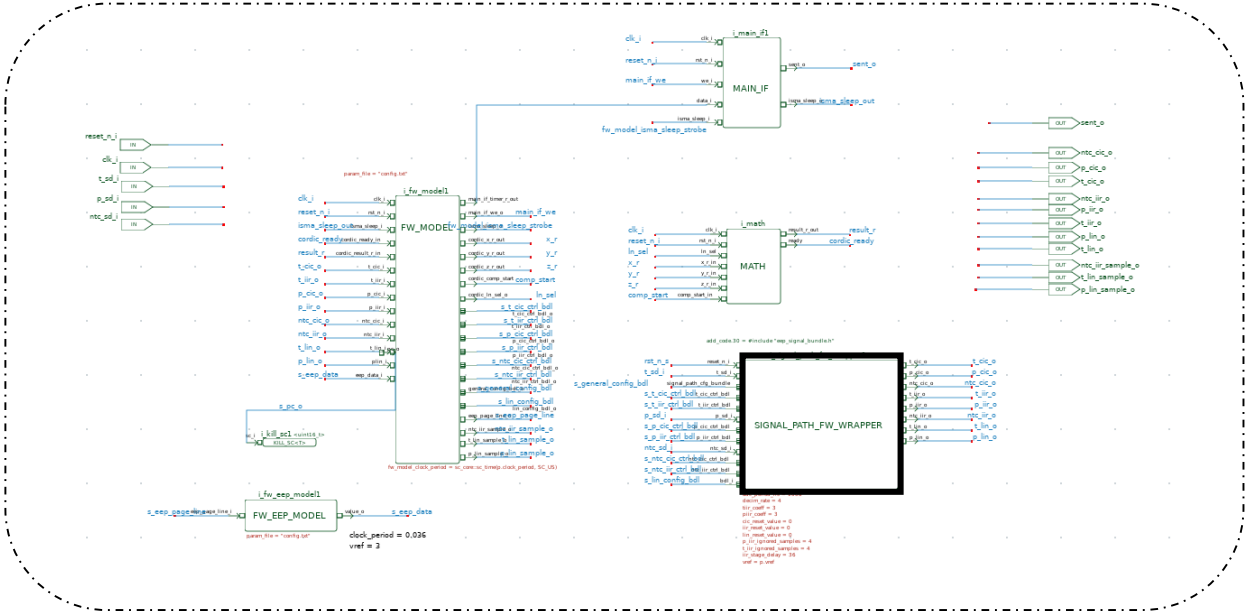
- High complexity bugs hide at the ‘edges’ of the design
  - Interaction between firmware units
  - Interaction between firmware and hardware
- Firmware and digital design activities are not necessarily synchronized
- Effort of verification increases late in design phase – FW and HW activities are done
- Simulation time is slow for Firmware necessities (gate level not necessary for SW)
- FW on top approach shifts integration verification effort to FW team

Block diagram – Manifold air pressure application



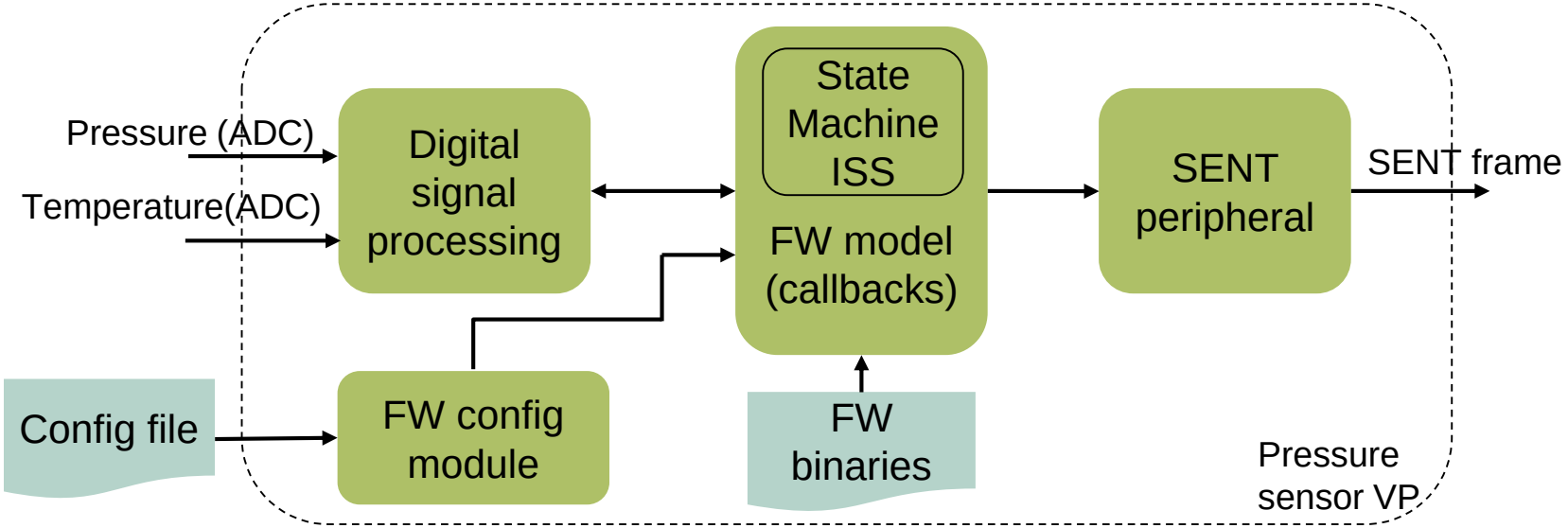
Concept feasibility VP development

- Digital core VP driven by concept requirements
- Model implements digital core of a pressure sensor used for manifold application
- During concept feasibility – only digital signal processing path was of interest



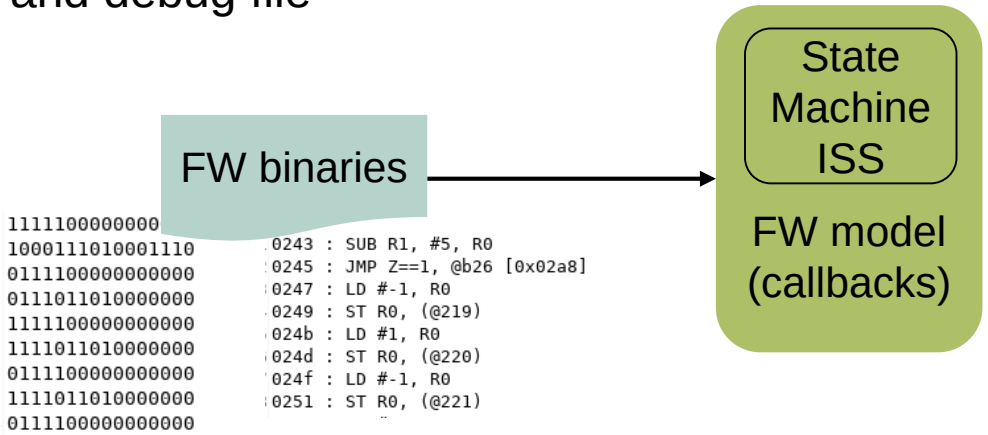
Firmware integration

- Integration of further level of detail and ‘peripherals’ used by FW as SC modules
- Integration of state machine core instruction set simulator
  - Supports performance parameters evaluation emulating target behaviour - Timing analysis is critical for FW development in small ASICs



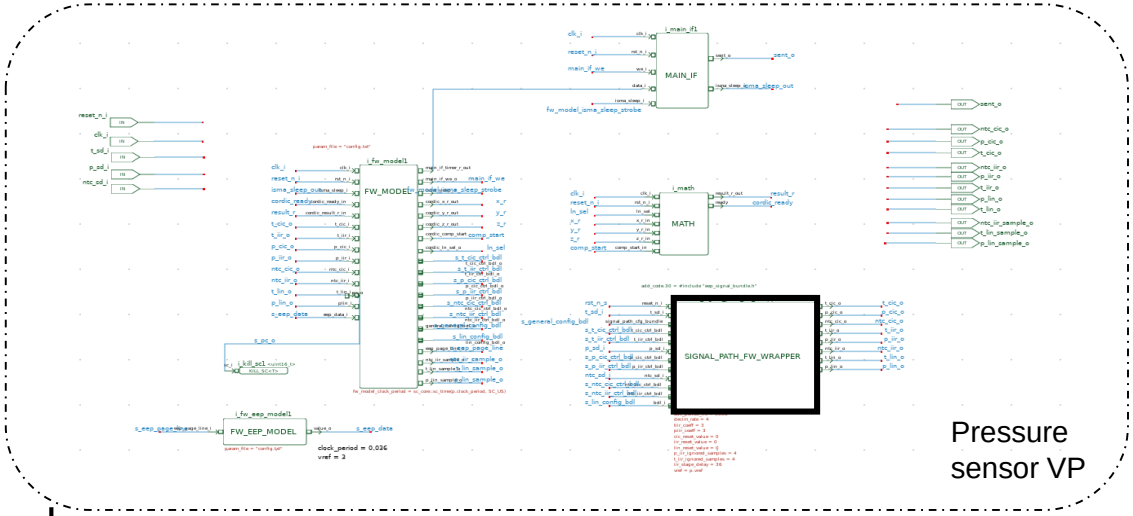
Firmware integration

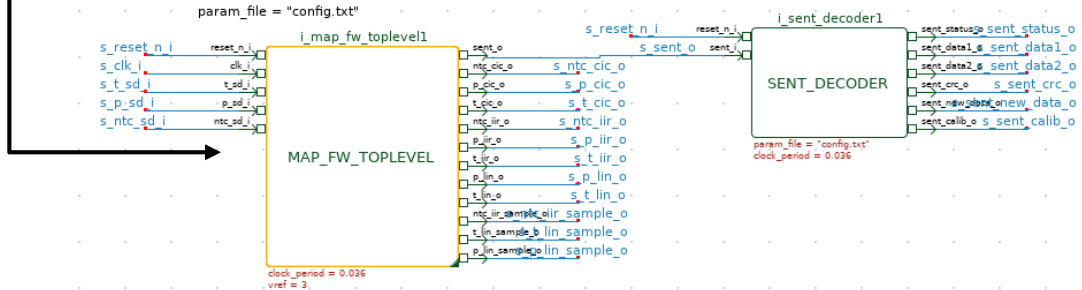
- Coupling between ISS and FW code done through sc_module wrapper implementing FW model callbacks
  - Separate callbacks for program counter iteration as well as read/write access to internal registers and RAM
- Firmware code thus integrated as precompiled binaries using specific compiler and debug file




```
void fw_model::construct()
{
    int counter;
    int prio[] = {1};
    int start_addr[] = {0x0000};
    int end_addr[] = {0xFFFF};
    std::ifstream debug_file("rom_code.txt");
    char *rom_files[] = {"romcode1.cod", "romcode2.cod"};

    iss_parameters iss_device_info;
    iss_device_info.NR_CORES = 1;
    iss_device_info.NR_MEMS = 1;
    iss_device_info.priorities = prio;
    iss_device_info.start_addresses = start_addr;
    iss_device_info.end_addresses = end_addr;
    iss_device_info.rom_filenames = rom_files;

    init_fw_model(this);

    setup_eep_data();

#ifdef MEMORY_CALLBACKS
    mem_callback* memory_callbacks_table[] = {memory_callback};
    iss_wrap_instance = new iss_wrapper("i_iss_wrapper1", iss_device_info, m
#else
    iss_wrap_instance = new iss_wrapper("i_iss_wrapper1", iss_device_info);
#endif
```

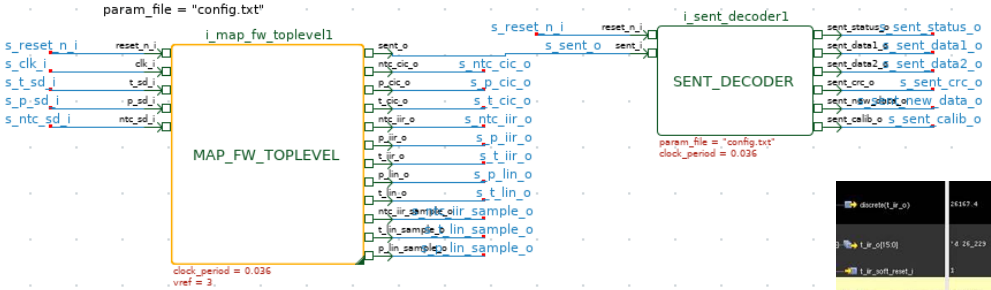
Testing concept - General

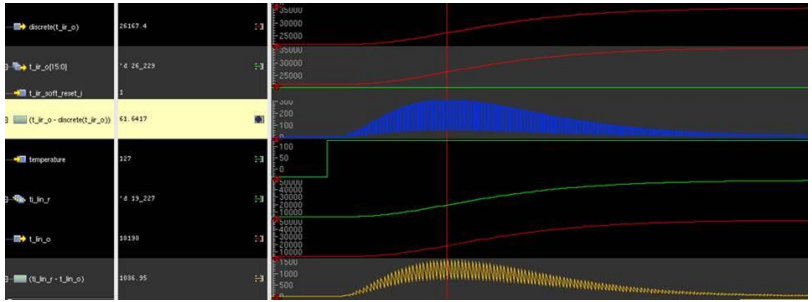


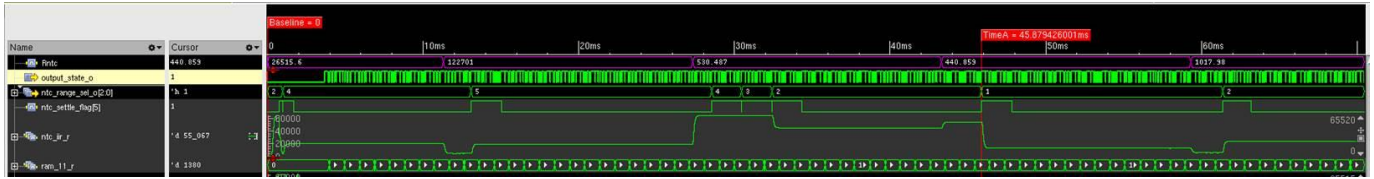


- Simple test bench instantiating VP and SENT decoder fulfilling ‘checkers’ role
- Stimuli driven programmatically in multiple stimuli files based on application (signal ramp, mid range, signal step etc.)
- Selection of stimuli done in regression based on implemented test case
- Rest of VP configuration done through *config.txt* file coupled to model parameters as well as configuration signals in dedicated module

Testing concept – simulation and co-simulation

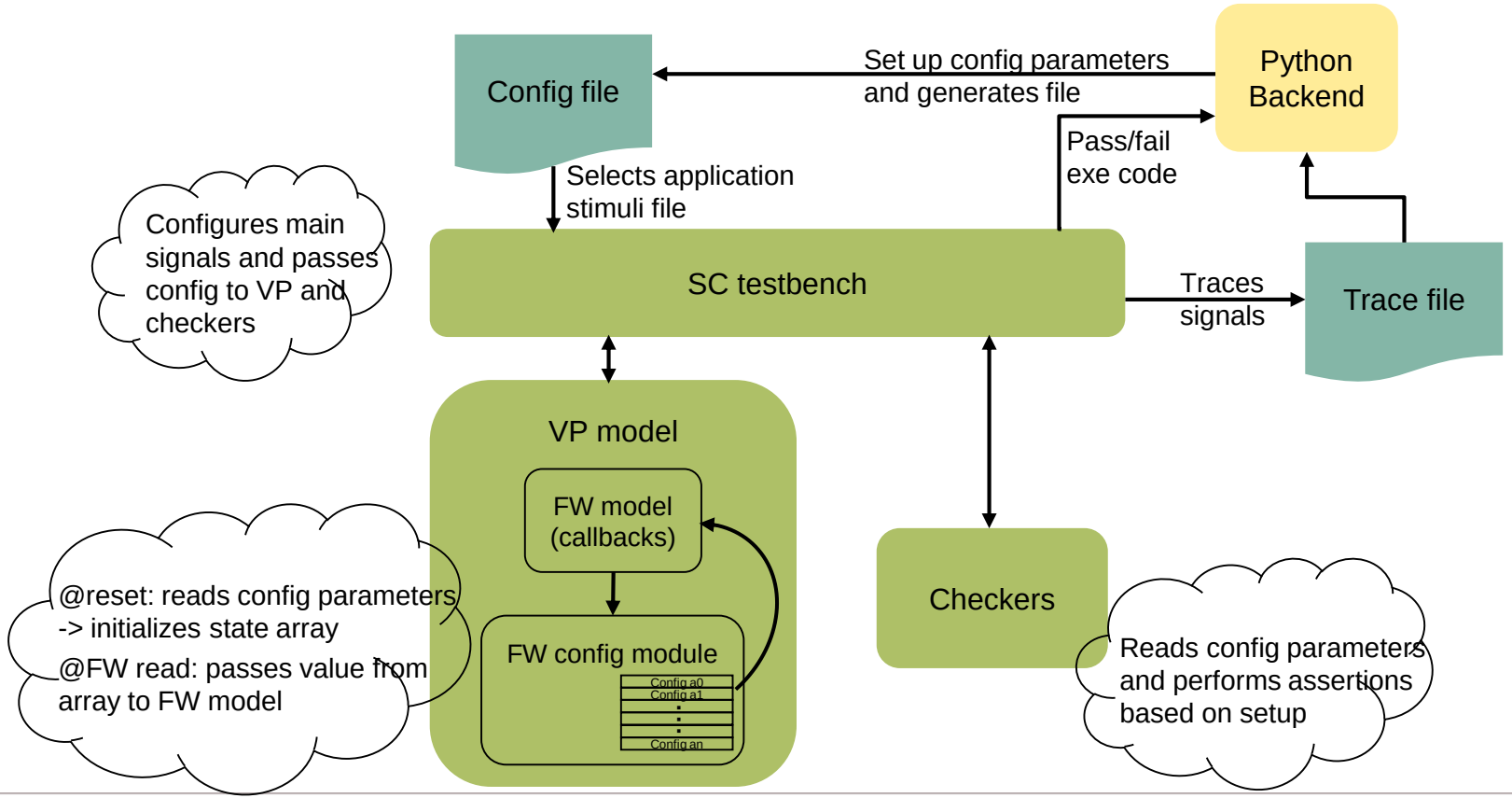






- Each stimuli TB setup checked in simulation also co-simulation with Xcellium coupling

Testing concept – from testbench to regression



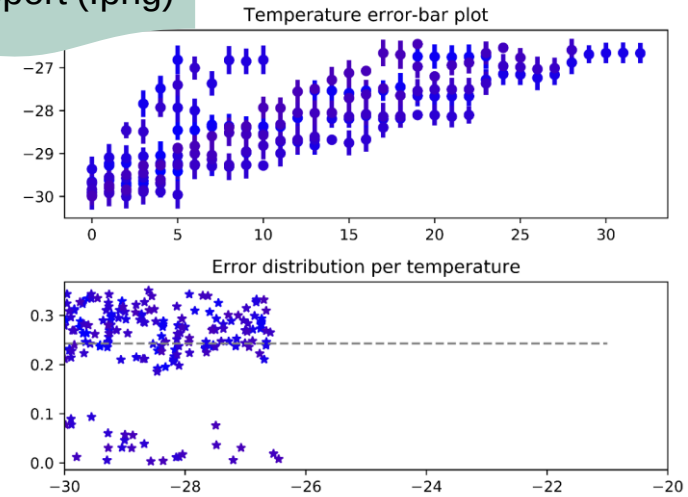
Python backend – a summary

- Configures the VP, handles and sets up the environment parameters through the configuration file.
- Launches multiple VP simulations in parallel on separate CPU cores or dispatches them on Unix LSF machines.
- Handles post-processing of results after simulation runs and supports creation of both tabular and graphical format reports.
- Enables easy regression runs and automated testing.

Regression results

- Regressions run on average **32.000** tests – dispatching on LSF 50 jobs at a time
  - Runtime for one such regression is ~1.5-2 days
- Performance report extracted with number of fails and configuration files saved for failed tests
- Graphical reports as well as coverage reports generated

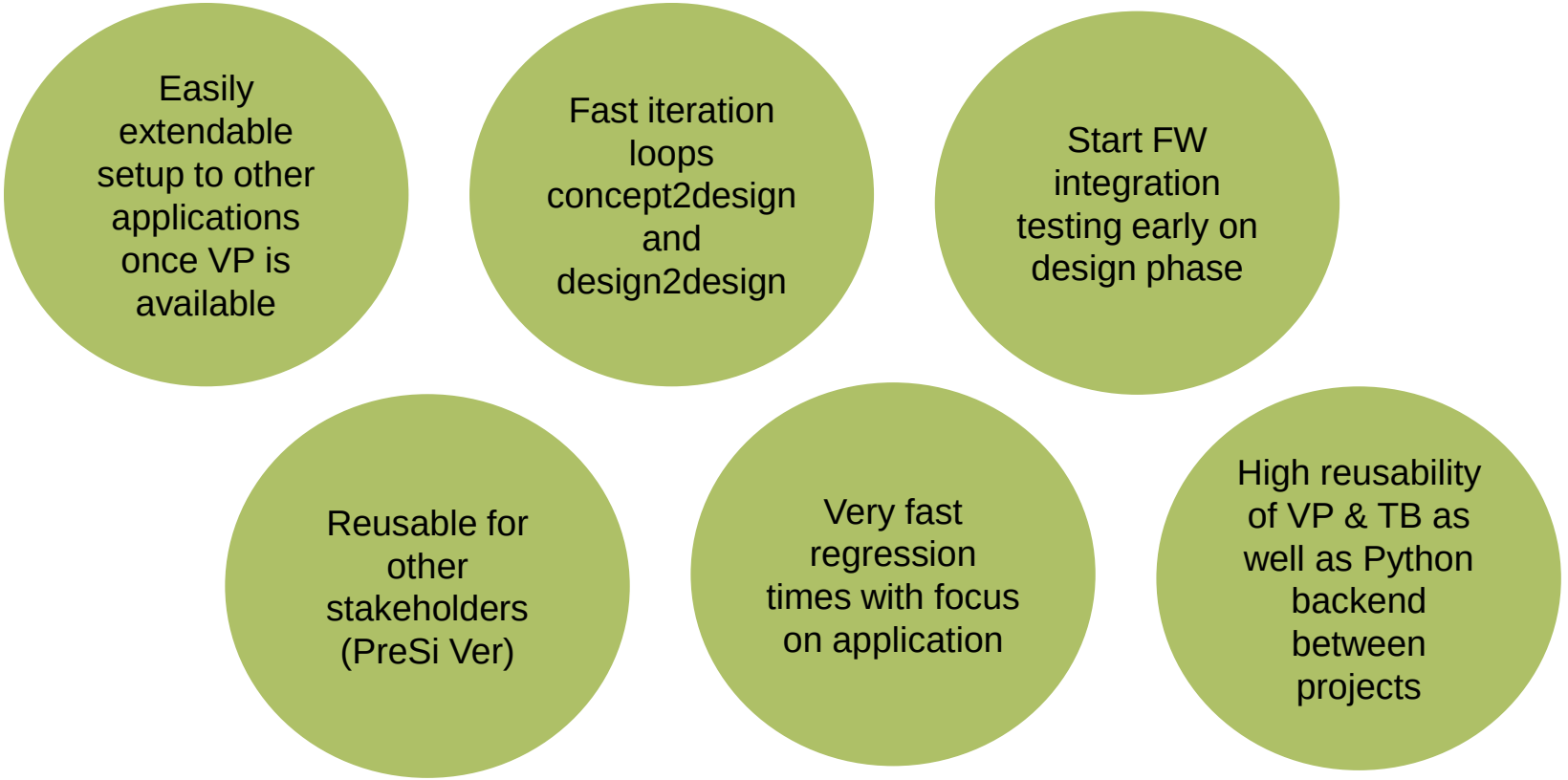
Graphical report (.png)



Coverage report (.html)

```
uint16_t math_parity(uint16_t input_value)
:  {
175  :    uint16_t result;
176  :
177  :    result = input_value;
178  :
179  :    /* This is a quick algorithm to calculate the
180  :    result = result ^ (result >> 8u);
181  :    result = result ^ (result >> 4u);
182  :    result = result & 0x000Fu;
183  :    result = (0x6996u >> result);
184  :    result = result & 1u;
185  :
186  :    return  result;
187  :  }
188  :
```

Benefits



Part of your life. Part of tomorrow.