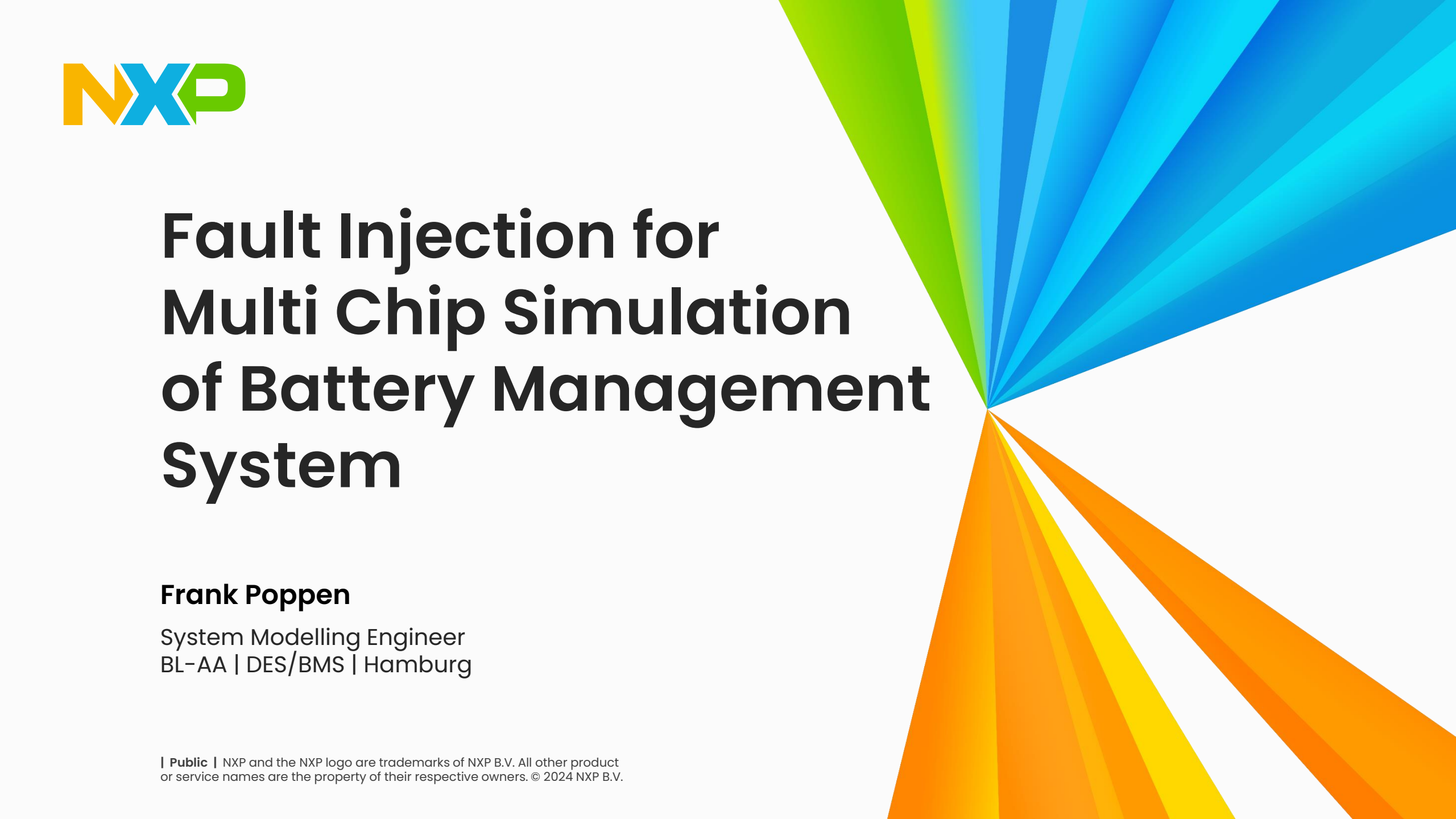


Fault Injection for Multi Chip Simulation of Battery Management System

Frank Poppen

System Modelling Engineer
BL-AA | DES/BMS | Hamburg

Outline

- Motivation
- High Voltage Battery Management System
- Full System Simulation Setup
- Fault Injection
- Conclusion and Future Work

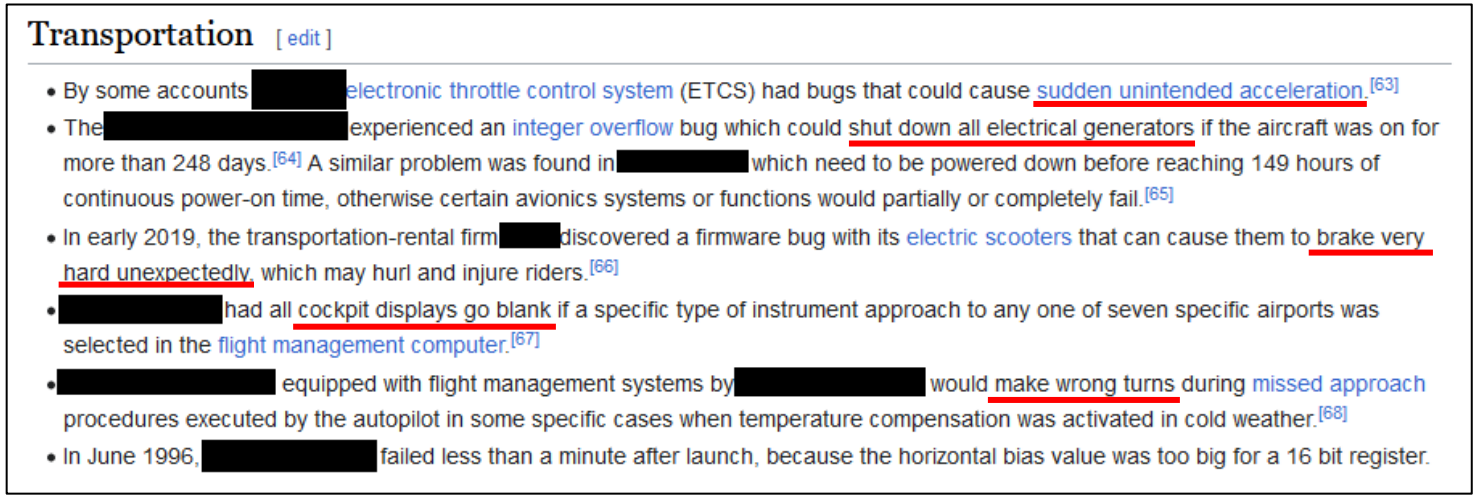
Motivation

Motivation

- Automotive Microelectronic Systems
  - high complexity
  - high degree of interaction inside them
  - high degree of interaction between them

- Strong Requirements
  - functional safety
  - reliability
  - cost
  - time to market

https://en.wikipedia.org/wiki/List_of_software_bugs



Transportation [edit]

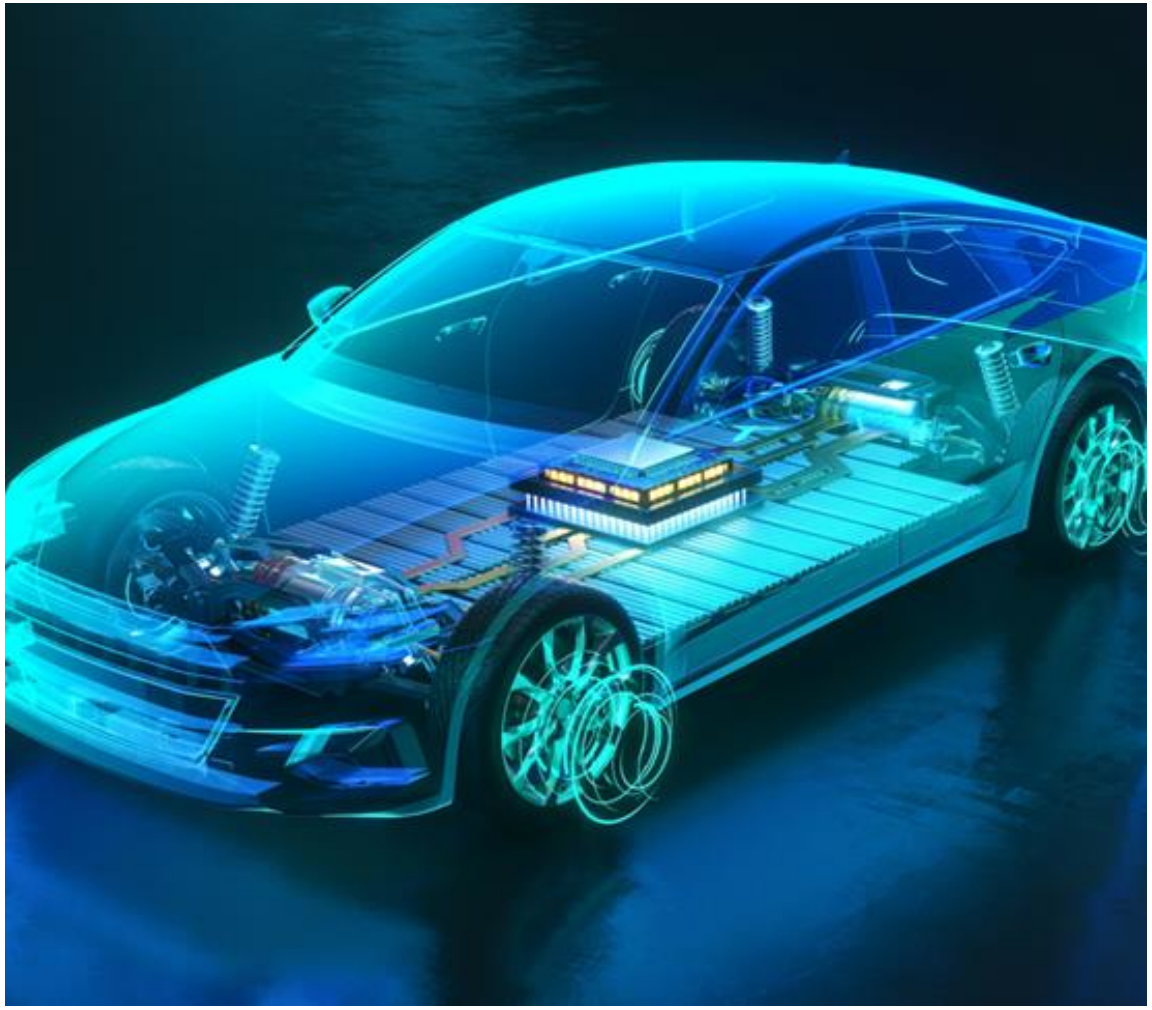
- By some accounts [redacted] electronic throttle control system (ETCS) had bugs that could cause sudden unintended acceleration.[63]
- The [redacted] experienced an integer overflow bug which could shut down all electrical generators if the aircraft was on for more than 248 days.[64] A similar problem was found in [redacted] which need to be powered down before reaching 149 hours of continuous power-on time, otherwise certain avionics systems or functions would partially or completely fail.[65]
- In early 2019, the transportation-rental firm [redacted] discovered a firmware bug with its electric scooters that can cause them to brake very hard unexpectedly, which may hurl and injure riders.[66]
- [redacted] had all cockpit displays go blank if a specific type of instrument approach to any one of seven specific airports was selected in the flight management computer.[67]
- [redacted] equipped with flight management systems by [redacted] would make wrong turns during missed approach procedures executed by the autopilot in some specific cases when temperature compensation was activated in cold weather.[68]
- In June 1996, [redacted] failed less than a minute after launch, because the horizontal bias value was too big for a 16 bit register.

- Modeling and Simulation-Based Systems Engineering

Motivation

Models are the Basis for Engineering

- Contain more detail than text specifications
  - no need to reread (reinterpret) text specifications repeatedly
  - less ambiguous (execute and observe)
- Can be shared
  - virtual integration of component models build larger systems (reuse and integrate)
  - early customer involvement
- Evaluate safety functions
  - fault injection
- Enable hardware/software co-design
  - improve time-to-market (shift left)

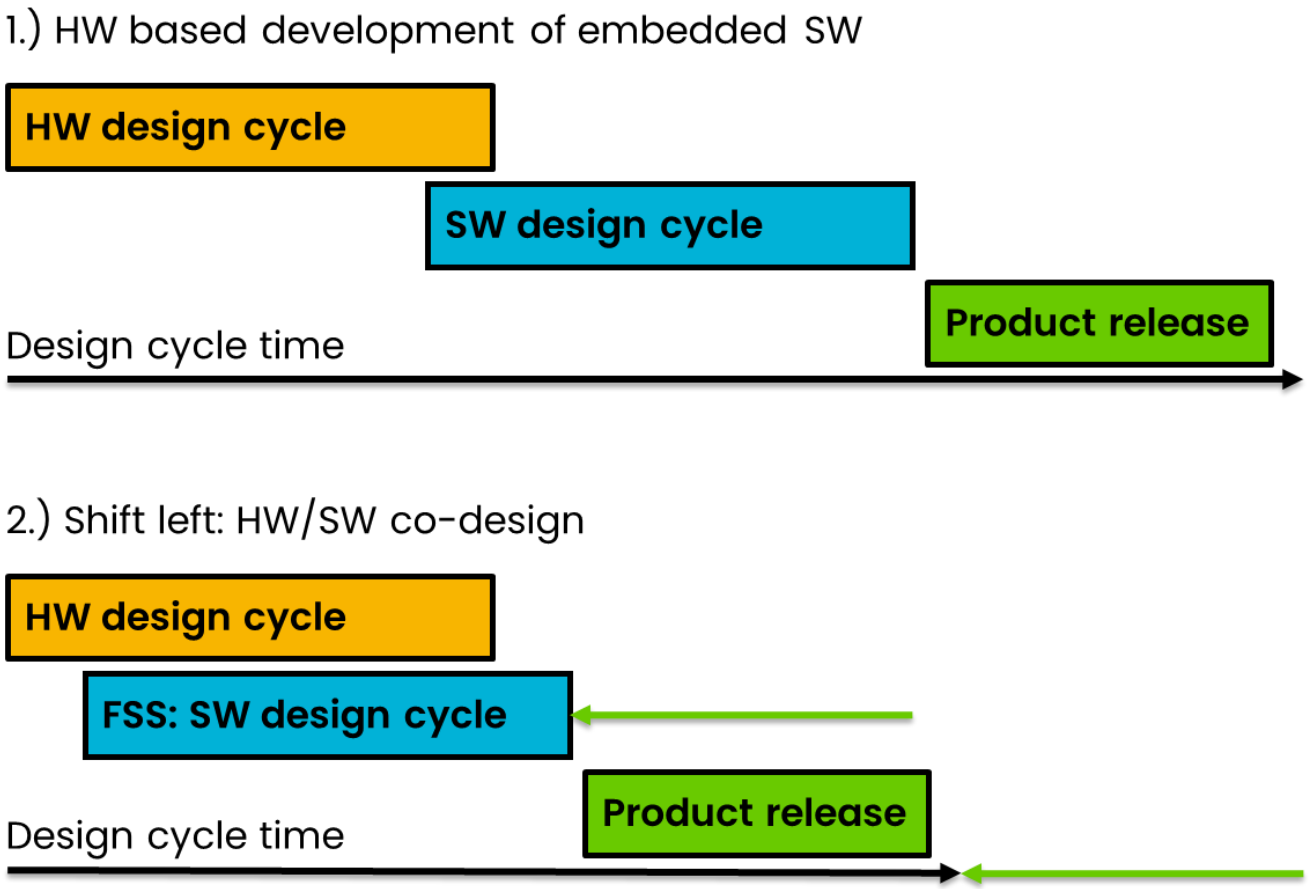


Motivation

Shift Left with HW/SW Co-Design

- Developing HW and SW in parallel
  - traditional: first develop ECU then SW
  - faster: start early SW development with virtual prototype
  - supports collaboration and quality of results reducing time to production

- Early SW testing
  - HW and lab space limit test cycles
  - fast and more test cycles enable increased coverage
  - fault injection to test safety features

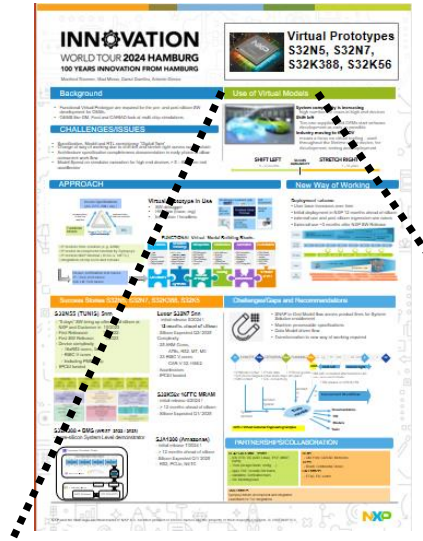


Motivation

Shift Left with HW/SW Co-Design

- Developing HW and SW in parallel
  - traditional: first develop ECU then SW
  - faster: start early SW development with virtual prototype
  - supports collaboration and quality of results reducing time to production

- Early SW testing
  - HW and lab space limit test cycles
  - fast and more test cycles enable increased coverage
  - fault injection to test safety features



Use of Virtual Models

System complexity is increasing
high numbers of cores in high end devices

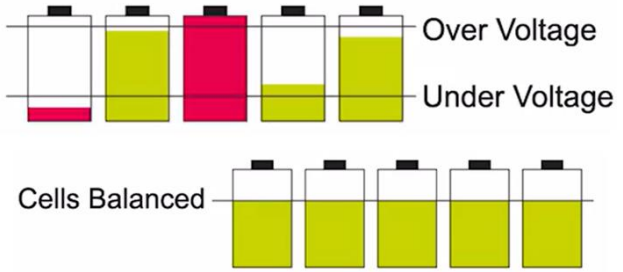
Shift left
Tier-one suppliers and OEMs start software development as early as possible

Industry moving to the SDV
means a focus on virtual tooling – used throughout the lifetime of the device, for development, testing and deployment

High Voltage Battery Management System

High Voltage Battery Management System

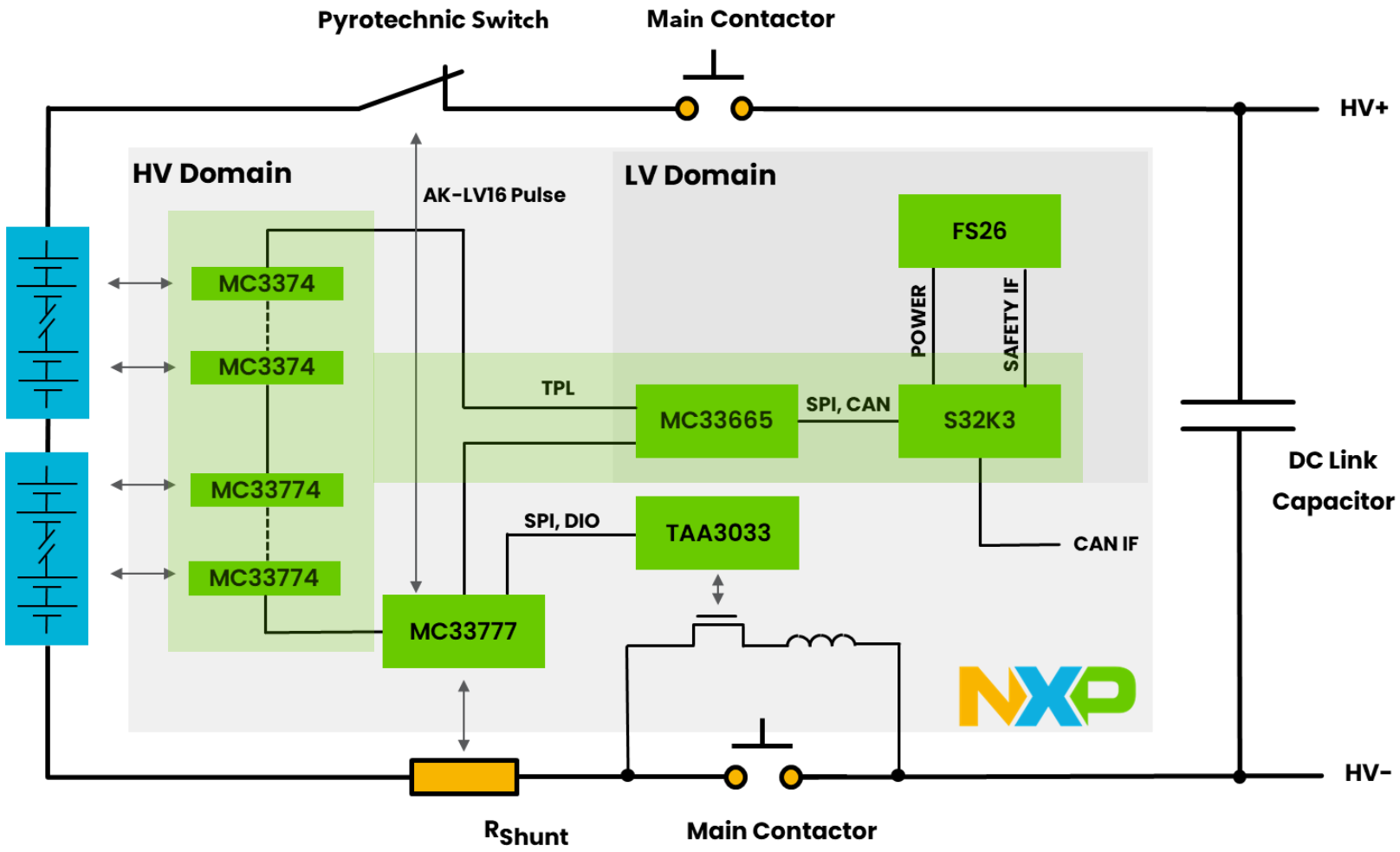
- No one size fits all BMS solution
  - scooters, motorcycles, buses, trucks, trains, boats or airplanes
  - battery packs as low as 14V or high voltages of 800V and more
- BCC directly connected to cells
  - no means of “turning off and on again”
  - low power consumption to avoid discharge of the cells
- Fire in an electric vehicle is an ASIL-D hazard
- Supervises voltage and temperature of cells
- Cell balancing
  - lowest voltage cell limits discharge
  - highest voltage cell limits charging



High Voltage Battery Management System

System Solution of a HV BMS

- **MCU**
  Microcontroller Unit
- **GTW**
  Gateway
- **BCC**
  Battery Cell Controller
- **TPL**
  Transport Protocol Link
- **BJB**
  Battery Junction Box
- **PRE**
  Precharge
- **SBC**
  Safety System Basis Chip



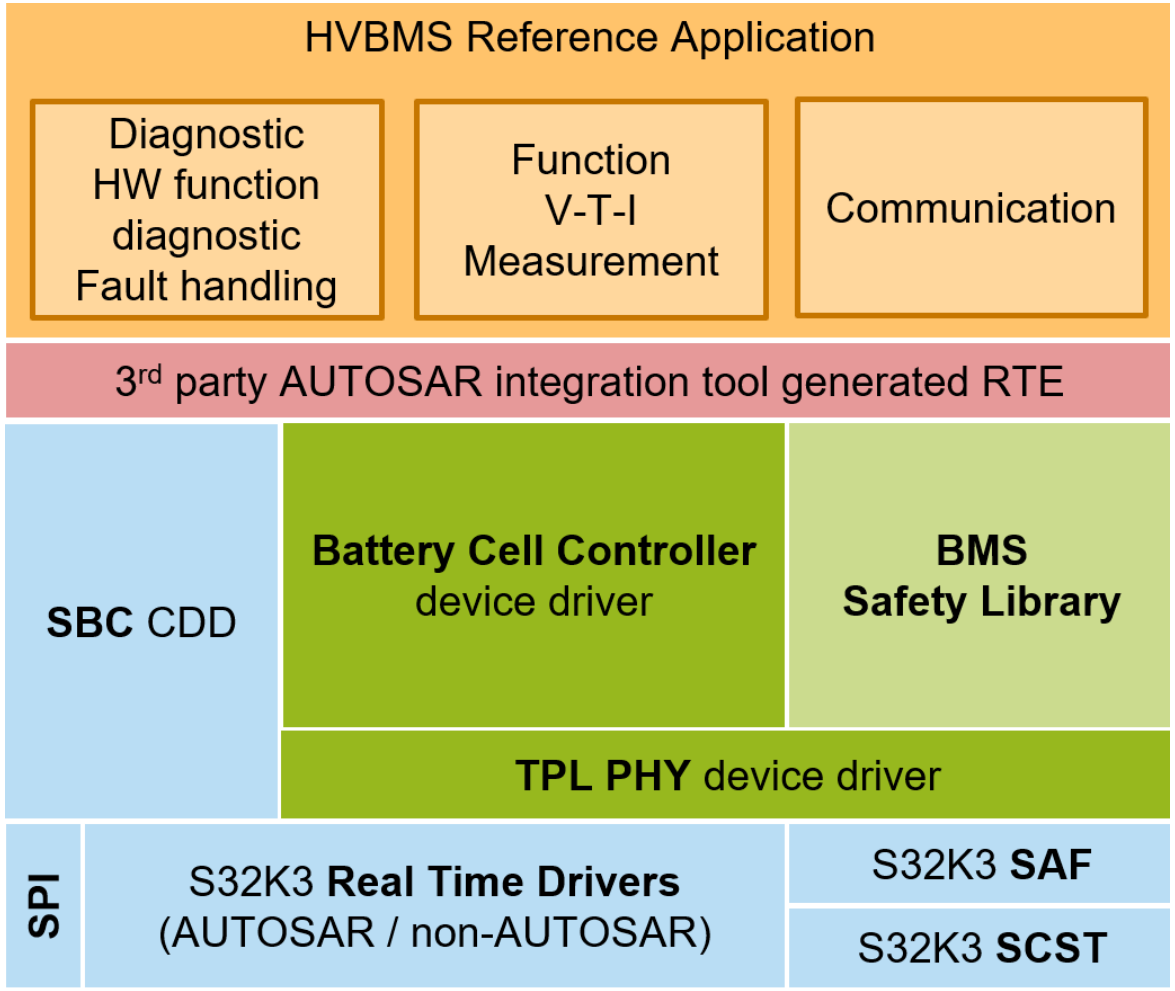
Full System Simulation Setup

Full System Simulation Setup

Software Stack

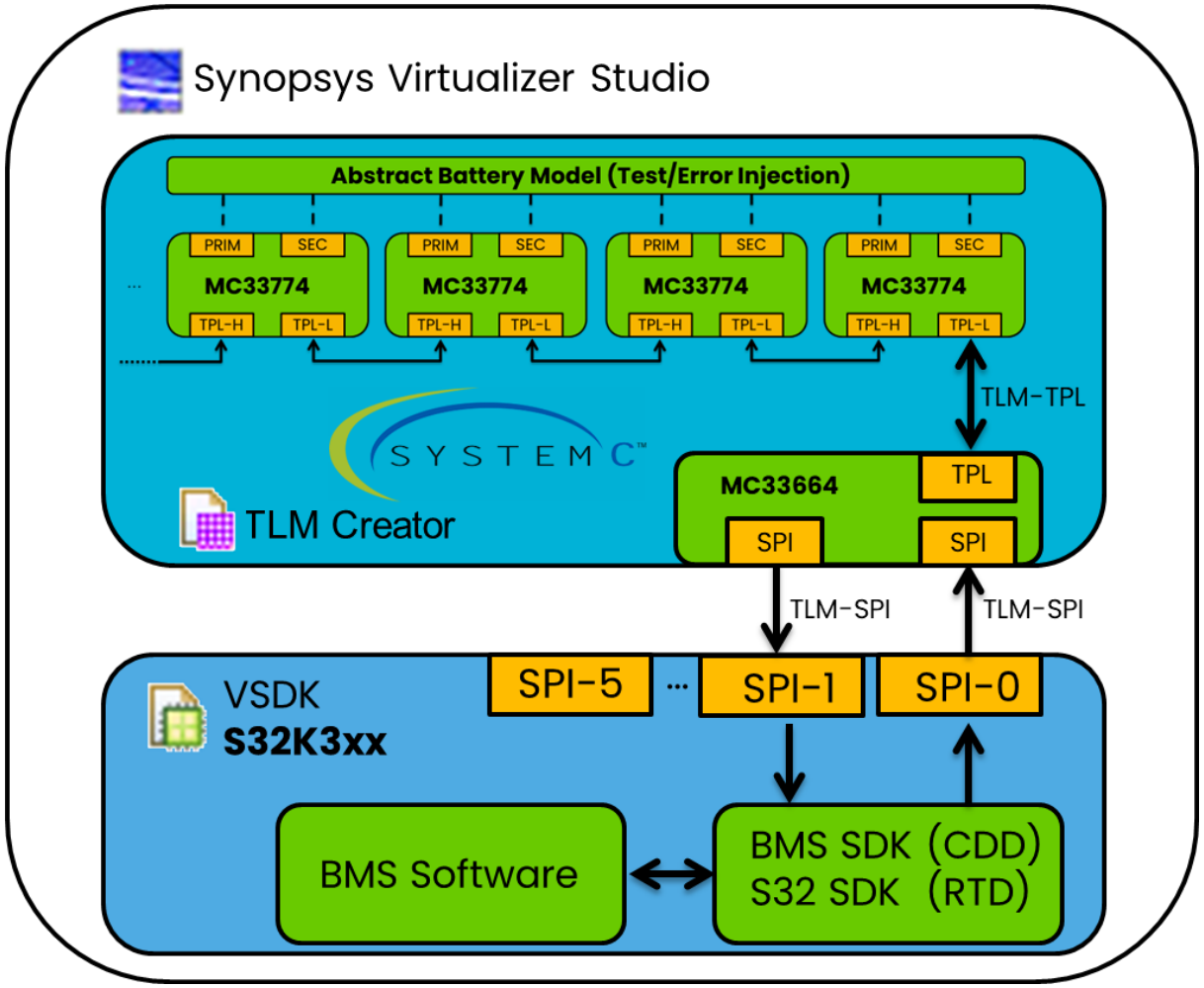
- Real time drivers
  - real time communication
- Complex device drivers
  - system basis chips
  - battery cell controller
  - …
- Safety library
- AUTOSAR
- Application

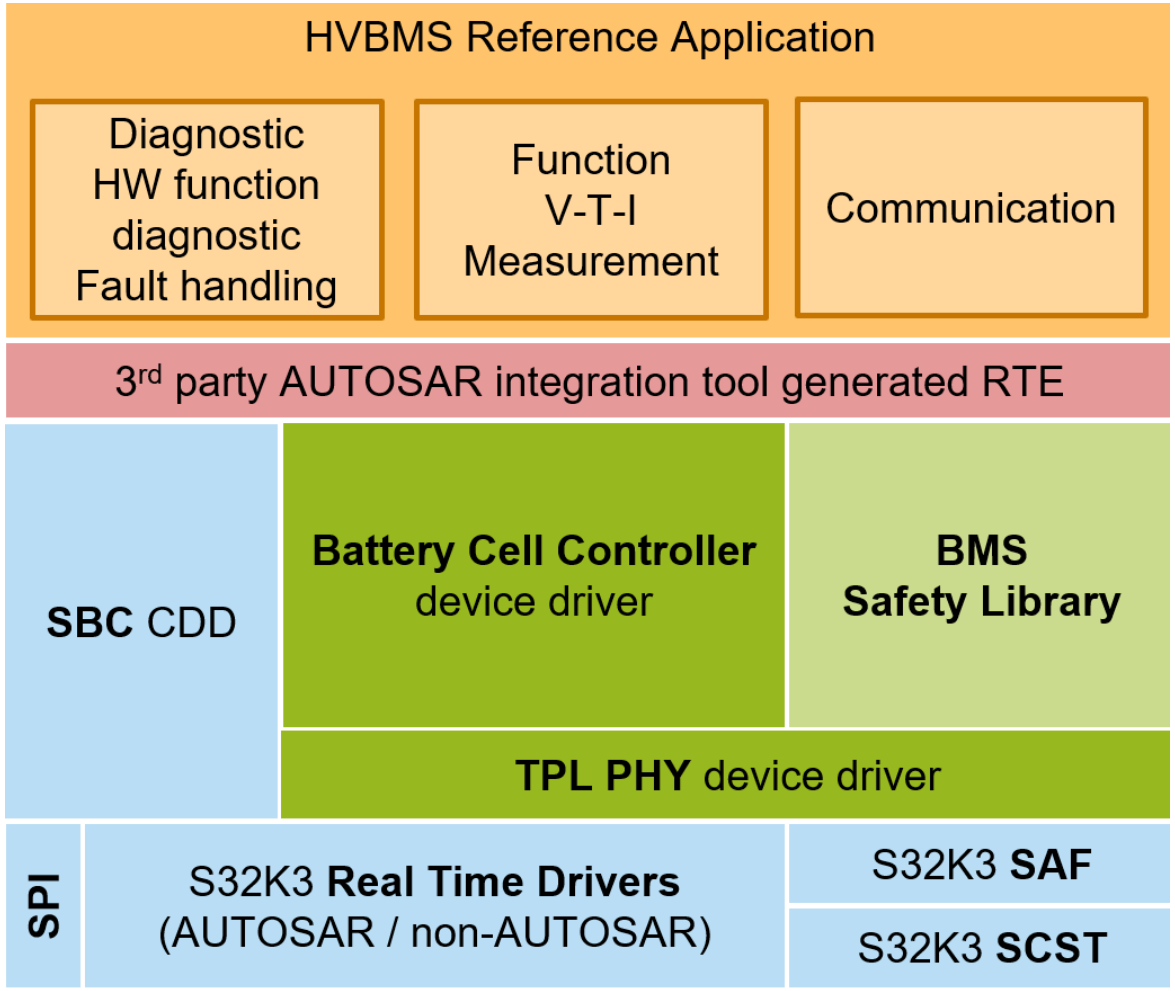
Software Stack Based on AUTOSAR Standard



Full System Simulation Setup

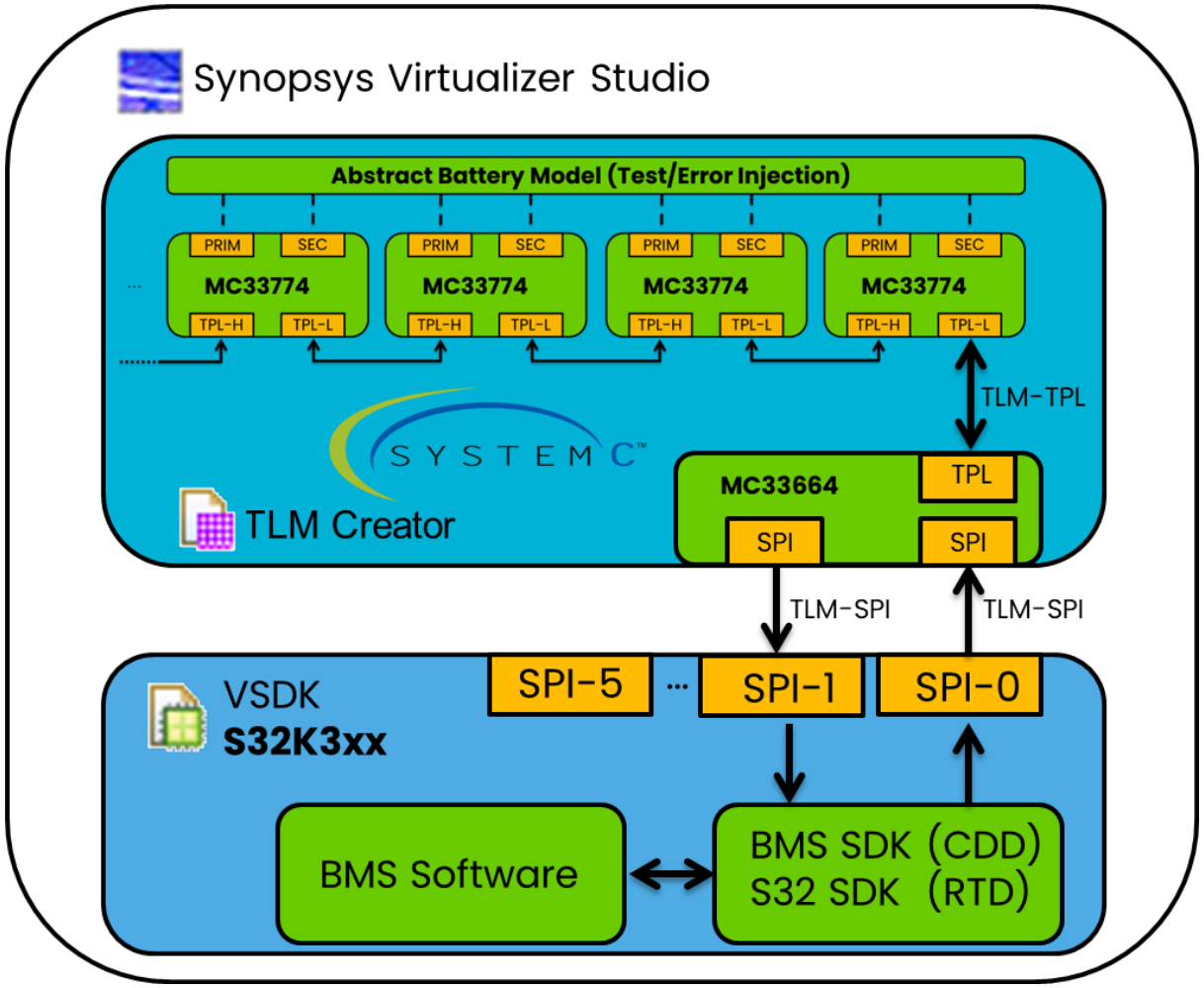
Hardware





Full System Simulation Setup

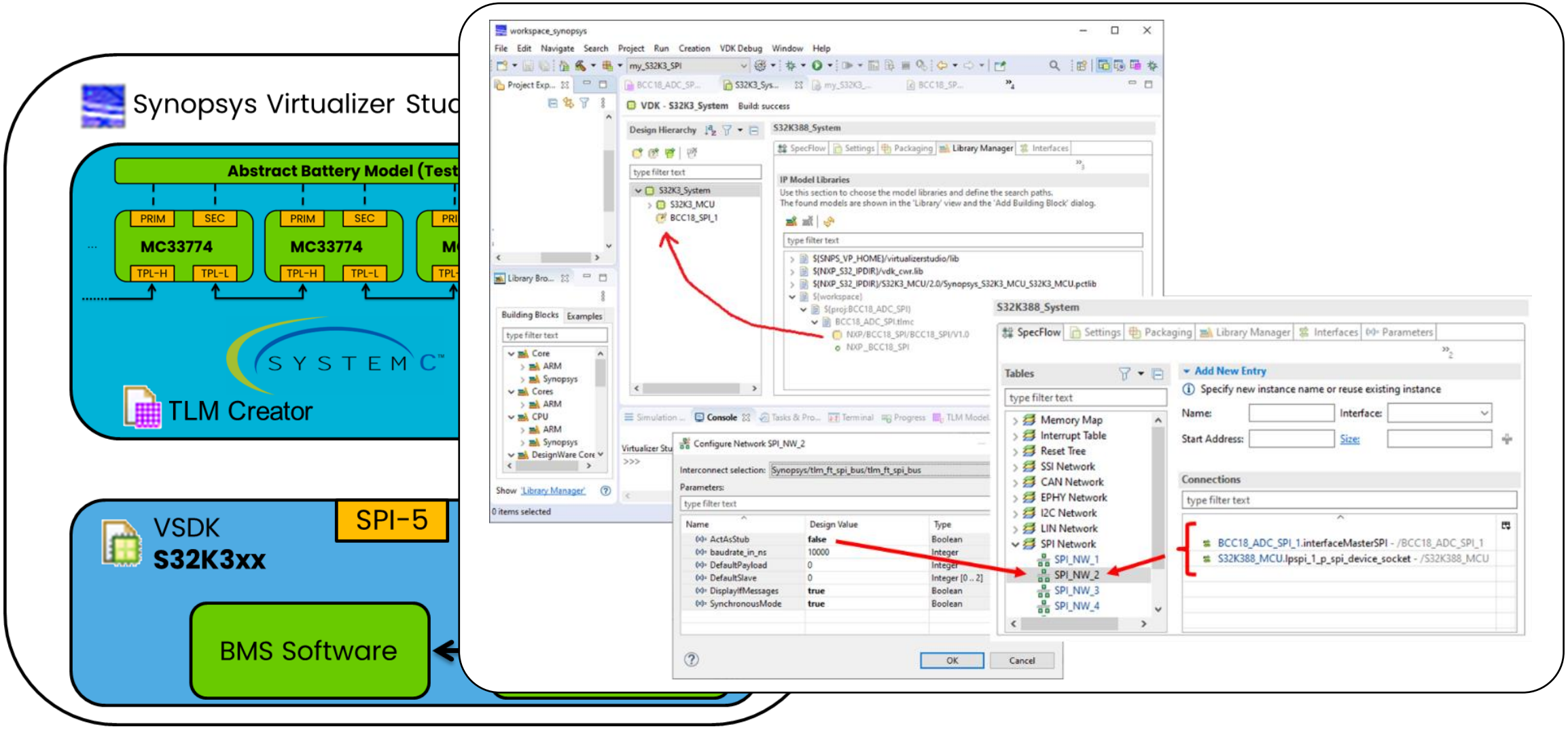
Hardware



- Synopsys VSDK of NXP's MCU
  - complete hardware perspective with registers and peripherals (GPT, UART, CAN, SPI)
- BCC functional SystemC model
  - SystemC architecture generated from SysML
  - functional behavior added manually
  - importing SystemC into TLM creator using python script (300 path- and filenames)
- Gateway
  - helpful Synopsys tutorial on SPI controller
  - translates TLM-SPI to TLM-TPL
- Drag and drop into VSDK
  - components in Virtualizer Studio Library

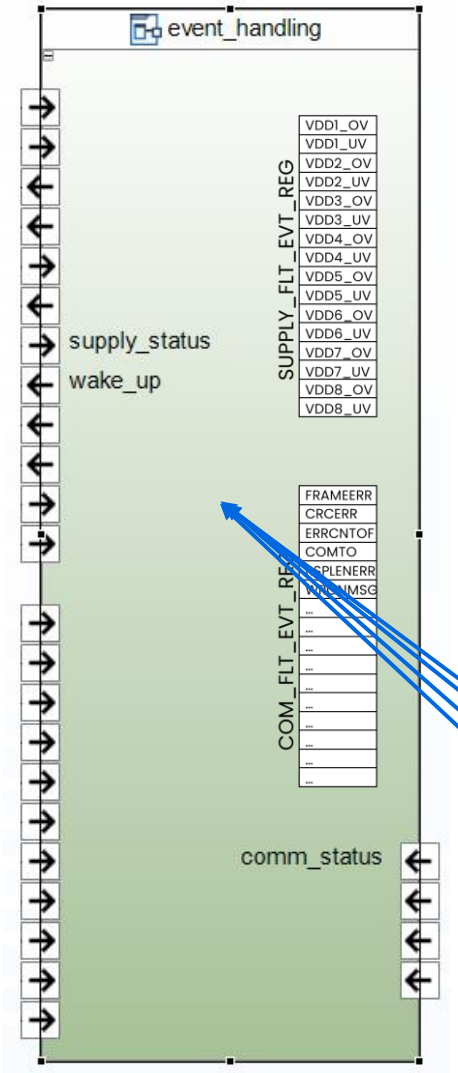
Full System Simulation Setup

Hardware

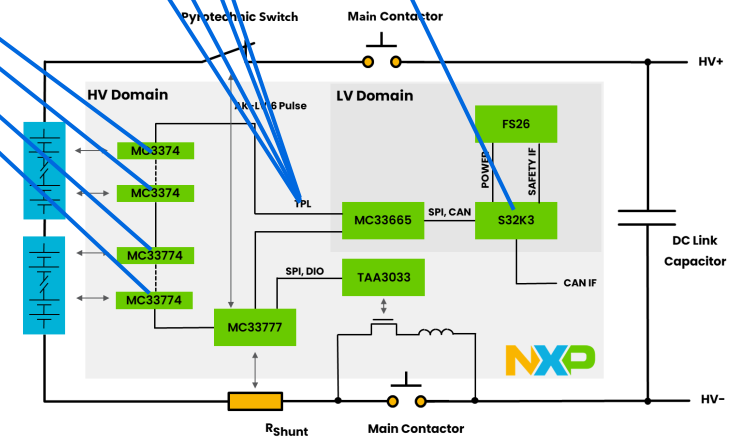


Fault Injection

Coseda Fault Injection

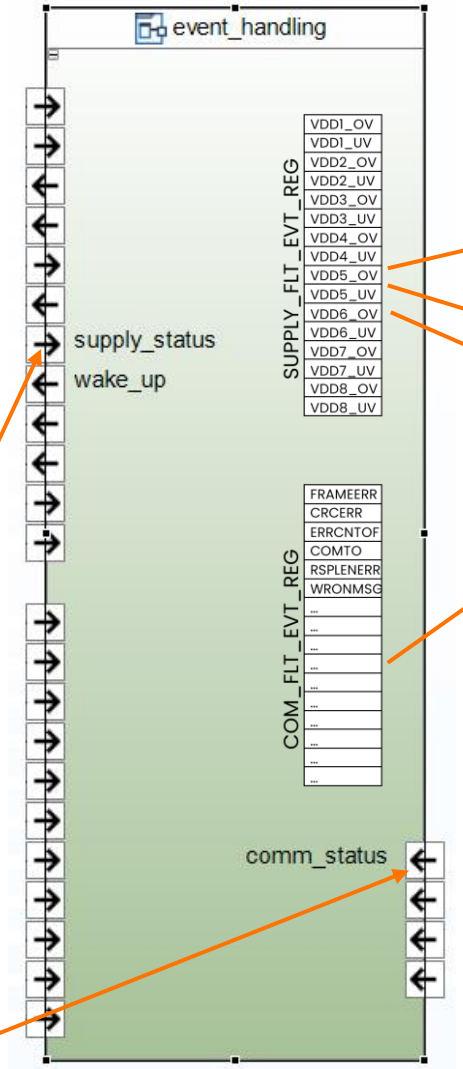



```
print("PY: Ensure the proper initialisation of the simulation model.")
tpl.mcui.scWaitUS(1000)

print("PY: Reading 'SUPPLY_FLT_EVT_REG' register. Should be 0x0.")
reset_reason = tpl.getRegisterVal(bcc.SUPPLY_FLT_EVT_REG, 1)
assert (reset_reason == 0x0), f"Expected '0x0', got: {reset_reason}"

print("PY: Reading 'COM_FLT_EVT_REG' register. Should be 0x08.")
com_rst_cfg = tpl.getRegisterVal(bcc.COM_FLT_EVT_REG, 1)
assert (com_rst_cfg == 0x08), f"Expected '0x08', got: {com_rst_cfg}"

...

print("PY: Idling for 10 msec, to start injecting faults.")
tpl.mcui.scWaitUS(10000)

# Under Voltage
print("PY: Reading 'SUPPLY_FLT_EVT_REG' register. Should be 0x08.")
reset_reason = tpl.getRegisterVal(bcc.SUPPLY_FLT_EVT_REG, 1)
assert (reset_reason == 0x08), f"Expected '0x08', got: {reset_reason}"

print("PY: Reading 'SUPPLY_FLT_EVT_REG' register. Should be 0x0 after prv read.")
reset_reason = tpl.getRegisterVal(bcc.SUPPLY_FLT_EVT_REG, 1)
assert (reset_reason == 0x0), f"Expected '0x0', got: {reset_reason}"

print("PY: Idling for 10 msec, for next injected fault.")
tpl.mcui.scWaitUS(10000)

...
```



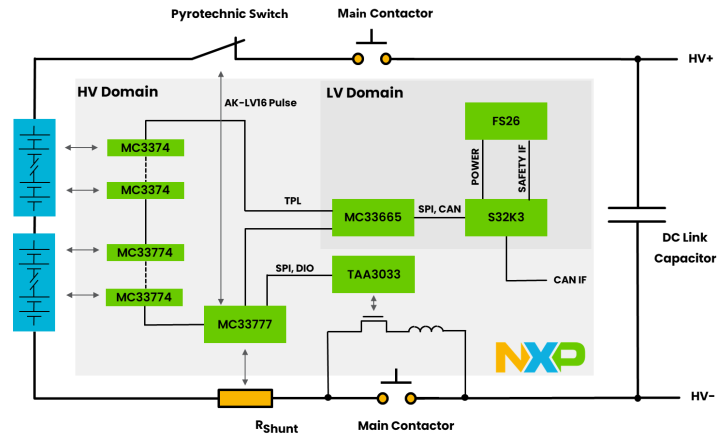
Coseda Fault Injection

```
int sc_main(int argc,char** argv) {
  ...
  std::cout << "Init model." << std::endl;
  bcctb_fi bcctb("bcctb");
  ...
  /*********************************************************************
   * Fault Injection controlled by file validating event_handling module
   *********************************************************************/
  if (scheduled_fi){
    ...
    cos_register_fault_type<cos_stuck_at<bms::supply_status_t>>
              ("cos_stuck_at_supply");
    cos_fault_injector injector;
    injector.read_faults("fault_list_POR.fi");
    injector.inject_faults();
    sc_core::sc_start(sc_core::sc_time(simDurationTime));
    std::cout << "Simulation end at " << sc_time_stamp() << std::endl;
  }
  ...
}
```

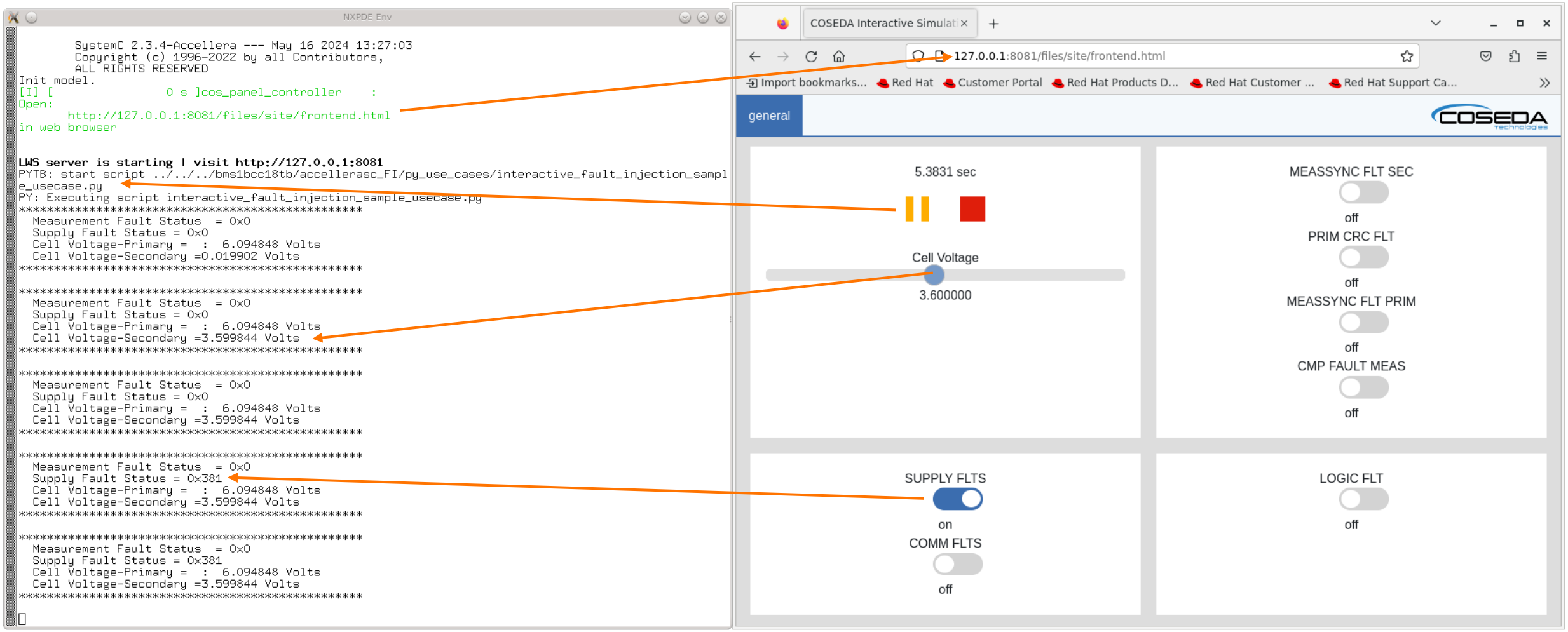



```
print("PY: Ensure the proper initialisation of the simulation model.")
tpl.mcui.scWaitUS(1000)

print("PY: Reading 'SUPPLY_FLT_EVT_REG' register. Should be 0x0.")
reset_reason = tpl.getRegisterVal(bcc.SUPPLY_FLT_EVT_REG, 1)
assert (reset_reason == 0x0), f"Expected '0x0', got: {reset_reason}"

print("PY: Reading 'COM_FLT_EVT_REG' register. Should be 0x08.")
com_rst_cfg = tpl.getRegisterVal(bcc.COM_FLT_EVT_REG, 1)
assert (com_rst_cfg == 0x08), f"Expected '0x08', got: {com_rst_cfg}"

...

print("PY: Idling for 10 msec, to start injecting faults.")
tpl.mcui.scWaitUS(10000)

# Under Voltage
print("PY: Reading 'SUPPLY_FLT_EVT_REG' register. Should be 0x08.")
reset_reason = tpl.getRegisterVal(bcc.SUPPLY_FLT_EVT_REG, 1)
assert (reset_reason == 0x08), f"Expected '0x08', got: {reset_reason}"

print("PY: Reading 'SUPPLY_FLT_EVT_REG' register. Should be 0x0 after prv read.")
reset_reason = tpl.getRegisterVal(bcc.SUPPLY_FLT_EVT_REG, 1)
assert (reset_reason == 0x0), f"Expected '0x0', got: {reset_reason}"

print("PY: Idling for 10 msec, for next injected fault.")
tpl.mcui.scWaitUS(10000)

...
```

```
// fault_list_POR.fi.
...
cos_stuck_at_supply stuck_power_supply ( xxxxx.yyyyy.event_handling.supply_status_pi ) value:00000000000
cos_stuck_at_comm stuck_comm_status ( xxxxx.yyyyy.event_handling.comm_status_pi ) value:0000000000000000000
#Sequence
wait 10e-3
// VDD1 Under Voltage
stuck_power_supply active value:000010000000
wait 10e-3
// VDD1 Over Voltage
stuck_power_supply active value:000100000000
wait 10e-3
...
```



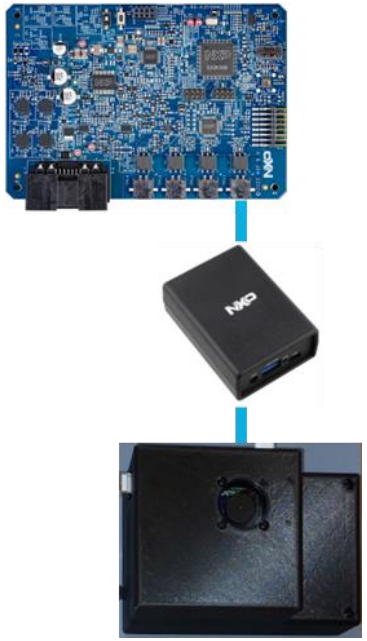
Coseda Fault Injection



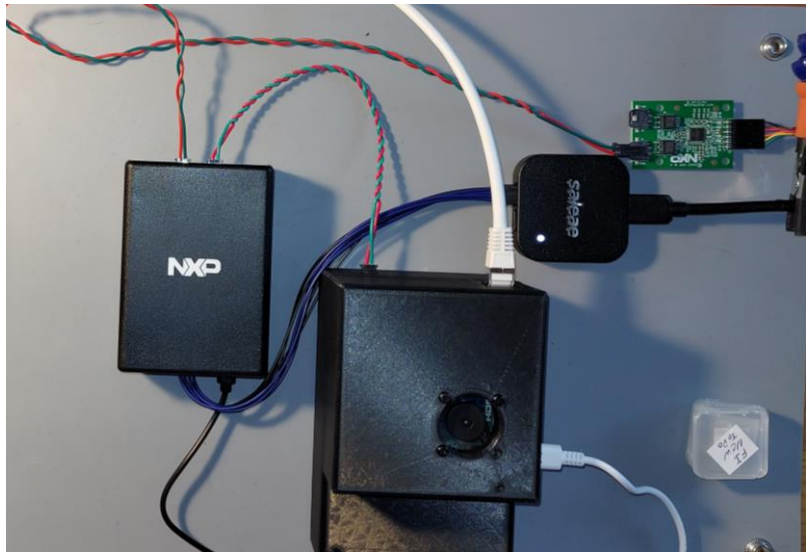
Conclusion and Future Work

Conclusion and Future Work

- Experience and results are very convincing
- Investment of resources is justified
- Involved RTD/CDD developers:
  - “when to make use of this in our projects?”
  - “would help to clean out laboratory space!”



- Product variants to be modelled
  - reuse potential expected between models
  - Using SysML MBSE (Model Based System Engineering)
- Model In the Loop with Fault Injection
  - first experiments with Raspberry Pis 4 and 5
  - real, physical MCU and simulated BCC



Frank Poppen
frank.poppen@nxp.com

nxp.com