CDNLIVE℠

cadence®

Cadence User Conference 2019

connect
share
inspire

Elihai Maicas, Tchiya Dayan

Intel, Realsense

Hybrid Verification Approach of Embedded Processor Integration

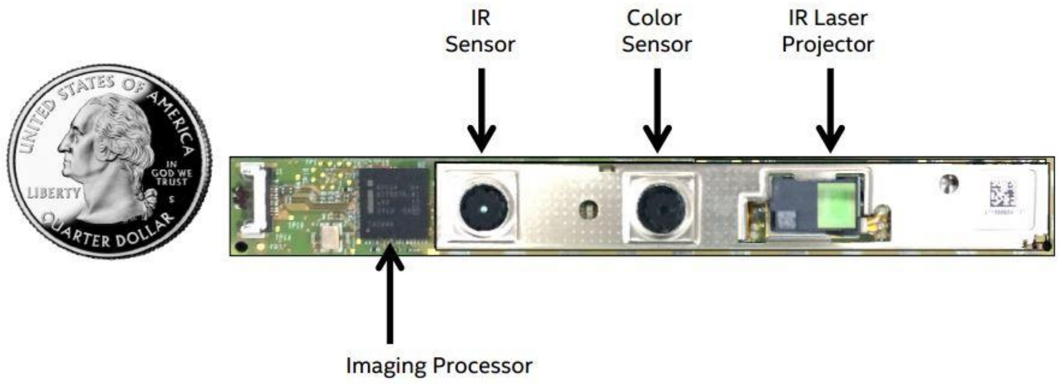
18.9.19

Agenda

- Introduction
- Embedded Processor – tasks and verification perspective
- Problem 1: multiple integration tests needed
  Solution: generate C content with the SV code
- Problem 2: complex system initialization flow
  Solution: hybrid simulation – BFMs and processor
- Summary

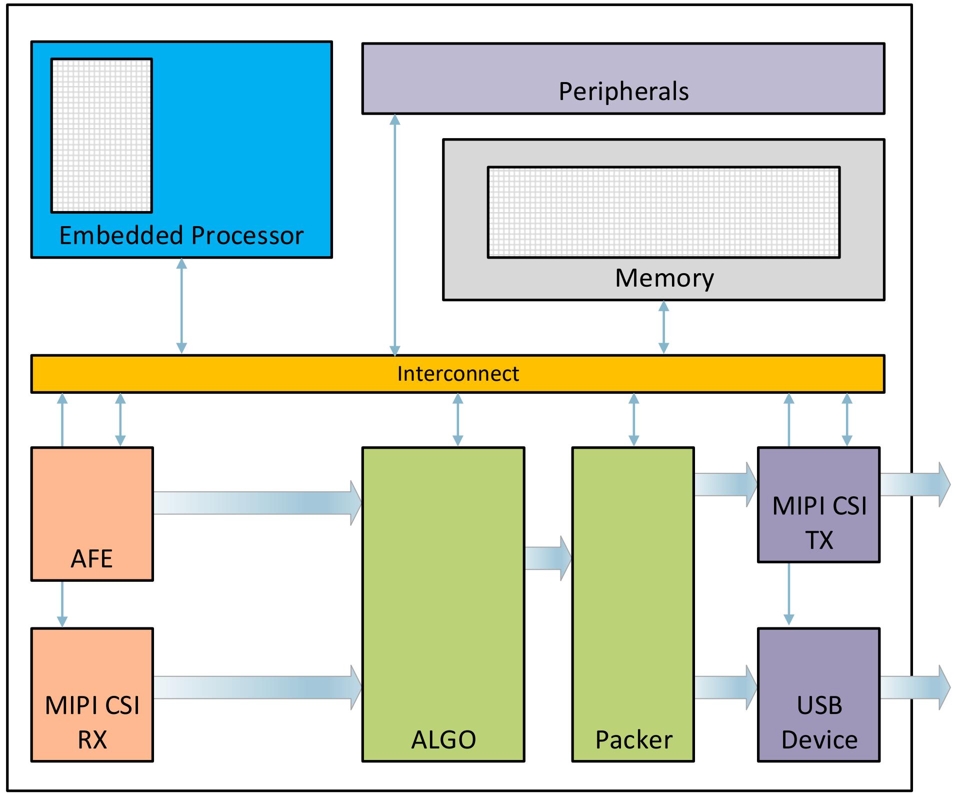
Introduction
Intel Realsense Group

- Intel® RealSense™ camera fits remarkable technology into a small package. There are three cameras that act like one - a 1080p HD camera, an infrared camera, and an infrared laser projector - they "see" like the human eye to sense depth and track human motion.



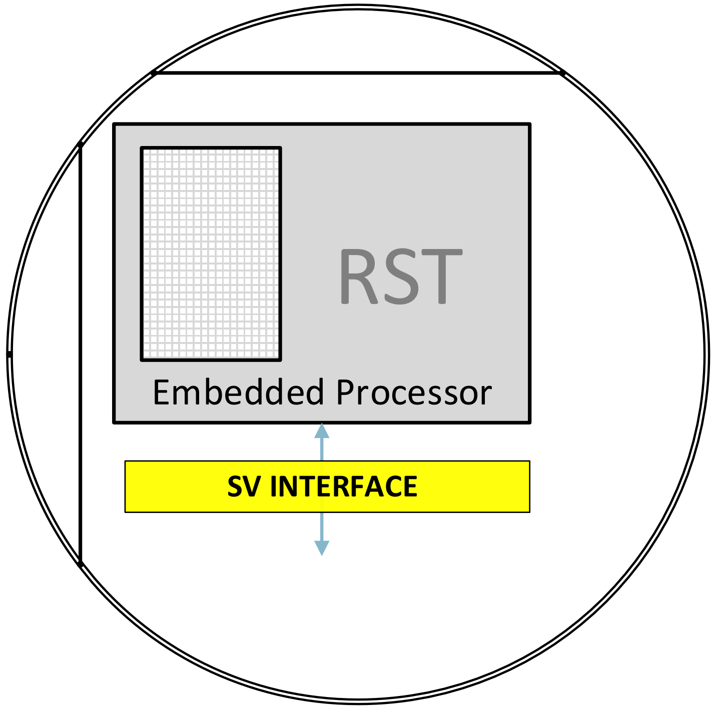
Introduction
Embedded Processor

- Embedded processor tasks
  - From simple registers configuration
  - To complex algorithms calculations (DSP)



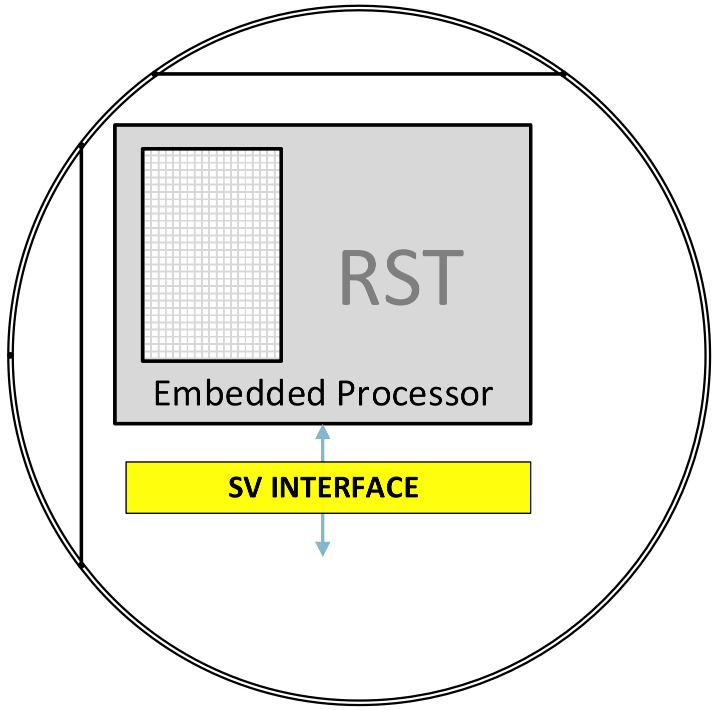
Embedded Processor Verification Perspective

- Verification Perspective
  - No need for full verification – 3rd party IP.
  - Bound BFM interface to the relevant busses and perform transaction via UVC.
  - Processor IP resides in reset.
  - The main issue is that the real processor behavior is not simulated.



Embedded Processor Verification Perspective

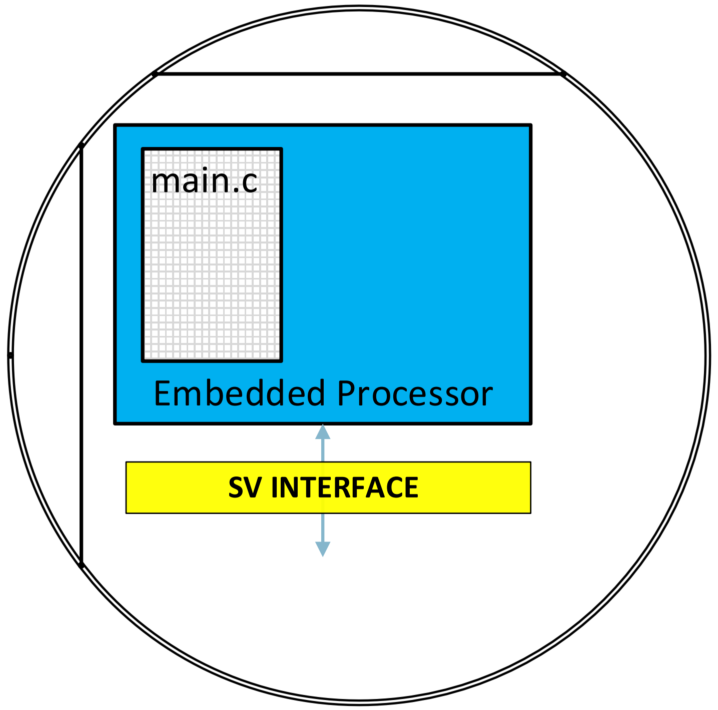
- Verification Perspective cont.
  - For the integration verification and system flows, individual C/ASM tests are composed.
  - This may become a drawback, if number of tests needed increases.



Introduction
Embedded Processor

- (Our) flow for running C test

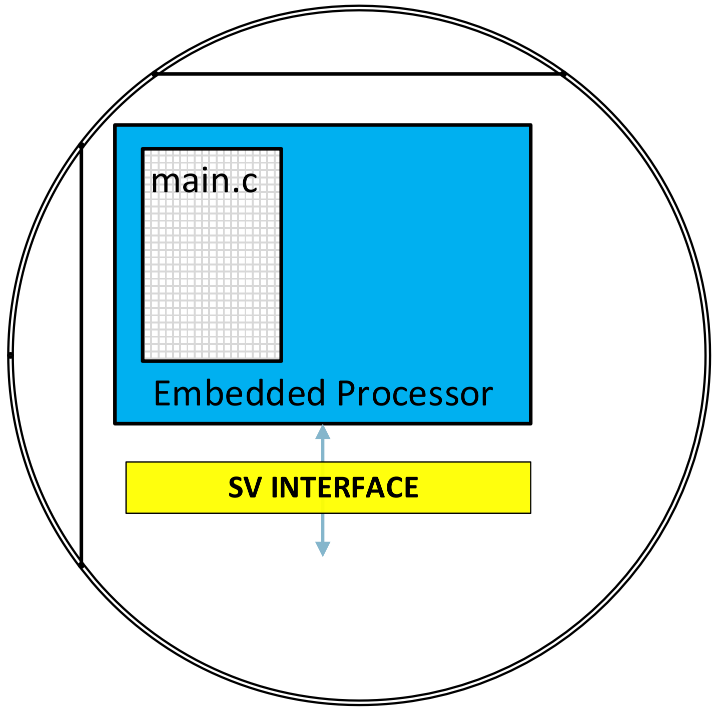
1. Compose `main.c` with your favorite editor.
2. Compile the code and generate hex file.
3. During the reset phase of the simulation, load the hex file to iRAM.
4. Once the processor in out of reset, it will start executing the loaded code.



Embedded Processor Integration Problem: multiple tests needed

- Problem 1:

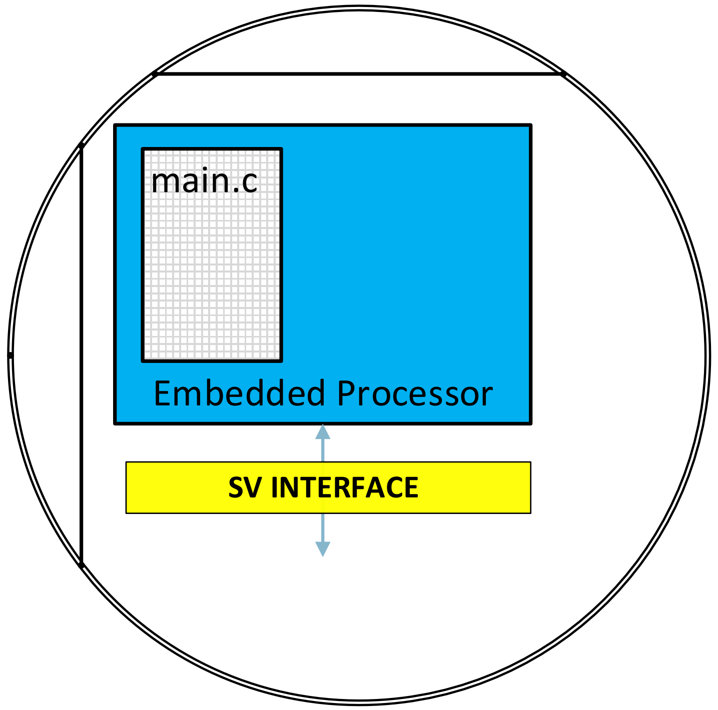
  In order to thoroughly verify some system flows that involves the processor, multiple C/ASM test must be individually composed.



Embedded Processor Integration Problem: multiple tests needed

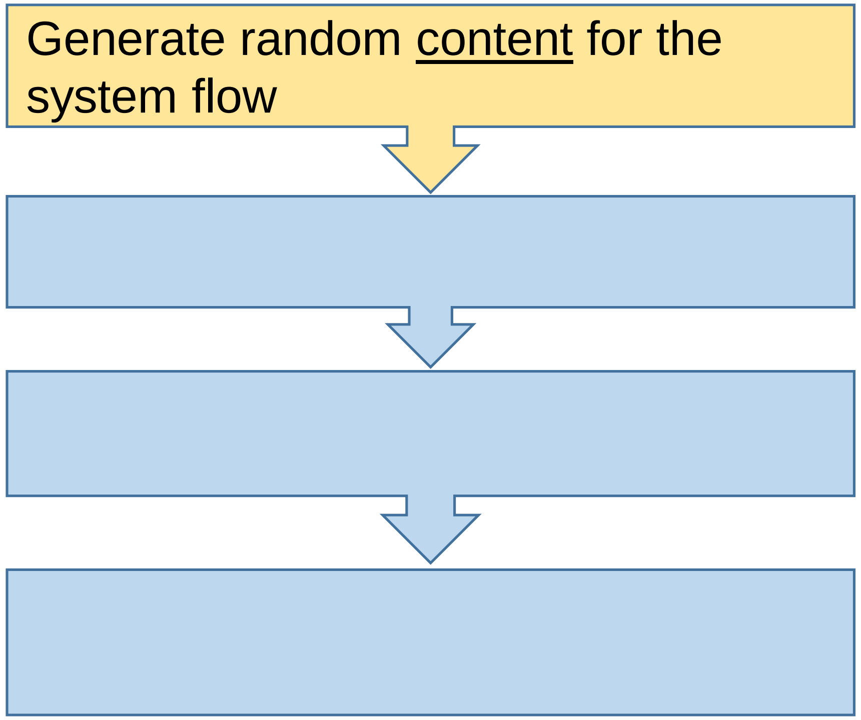
- Proposed solution:

  Use our state-of-the-art constraint-random verification environment, to generate random content for the C test, and run it with the processor.



Embedded Processor Integration Problem: multiple tests needed

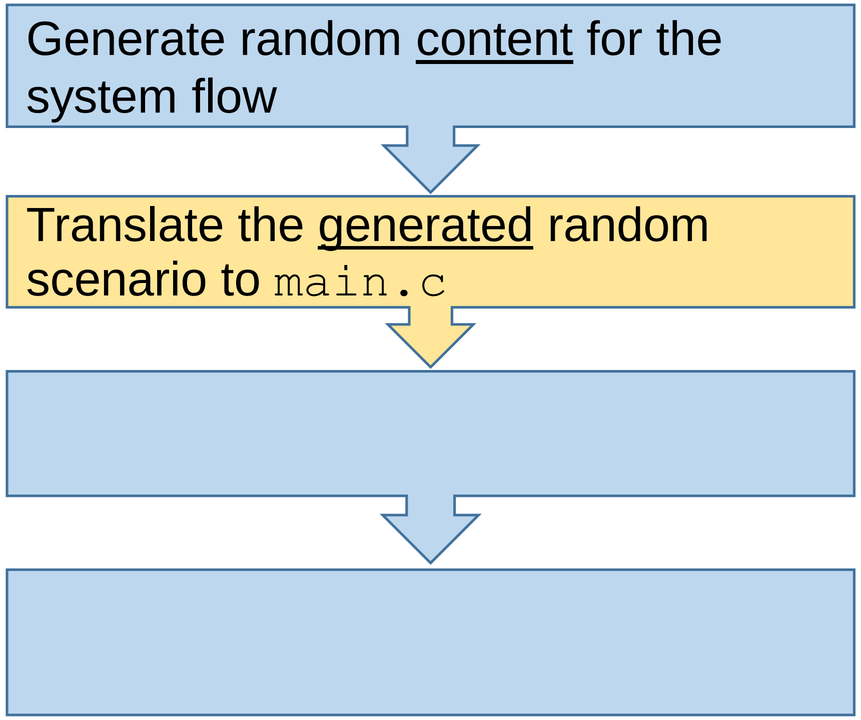
- Proposed solution:



```
class soc_ic_scenario_vseq extends uvm_sequence;
  `uvm_declare_p_sequencer(asr_txrx_virtual_sequencer)
  `uvm_object_utils(soc_ic_scenario_vseq)

  rand bit[1:0]                          m_prior_arr[AHB_MASTER_T_SIZE];

  rand bit[1:0]                          m_inter_delay[AHB_MASTER_T_SIZE][$];

constraint soc_ic_scenario_vseq::simple_c {

  unique {m_prior_arr};

  foreach(m_prior_arr[ii]){
    m_prior_arr[ii] < 3;
  }

  foreach(m_inter_delay[ii]) {
    foreach (m_inter_delay[ii][jj]) {
      m_inter_delay[ii][jj] dist{0 := 18, 1 := 2, 2 := 2};
    }
  }
  //...
}
```

Embedded Processor Integration Problem: multiple tests needed

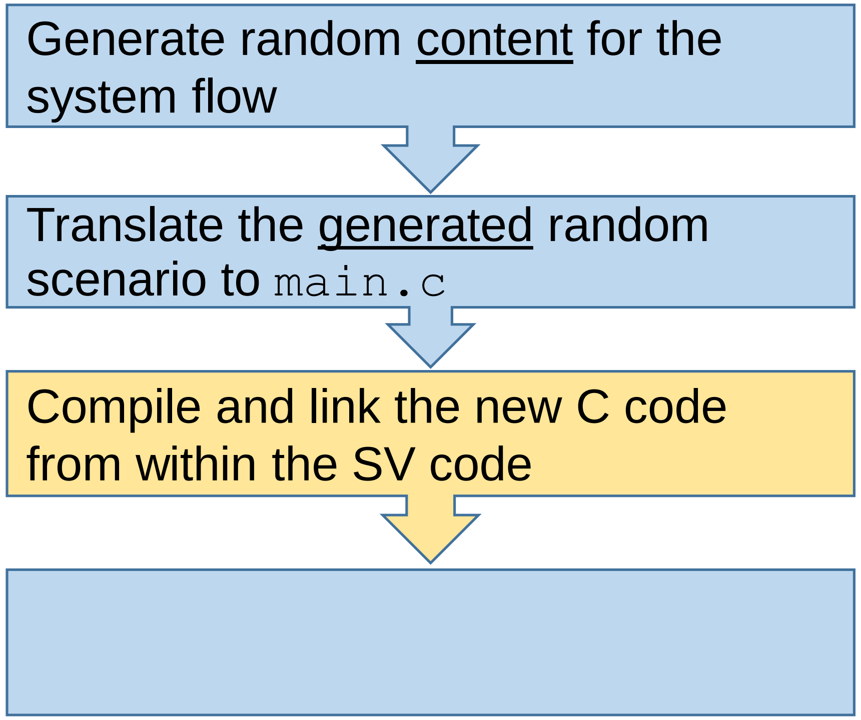
- Proposed solution:


Generate random content for the system flow
Translate the generated random scenario to main.c

```
function  void soc_ic_scenario_vseq::generate_lcp_code();

 int file_h;

 file_h = $fopen("main_lcp.c");

 $fwrite(file_h,"uint32_t read_value;\n");

 for(int ii=0; ii<m_addr_q[AHB_LCP].size;ii++) begin
   if(m_access_q[AHB_LCP][ii] == ic_env_pkg::READ)
     $fwrite(file_h,"read_value = (*((volatile uint32_t *) (0x%0h))); \n",
       m_addr_q[AHB_LCP][ii]);

   //...
 end
```

Embedded Processor Integration Problem: multiple tests needed

- Proposed solution:


Generate random content for the system flow

Translate the generated random scenario to `main.c`

Compile and link the new C code from within the SV code

```
//compile C - build and install script
$system("compile_and_gen_hex.sh");
```

Embedded Processor Integration Problem: multiple tests needed

- Proposed solution:

Generate random content for the system flow

↓

Translate the generated random scenario to `main.c`

↓

Compile and link the new C code from within the SV code

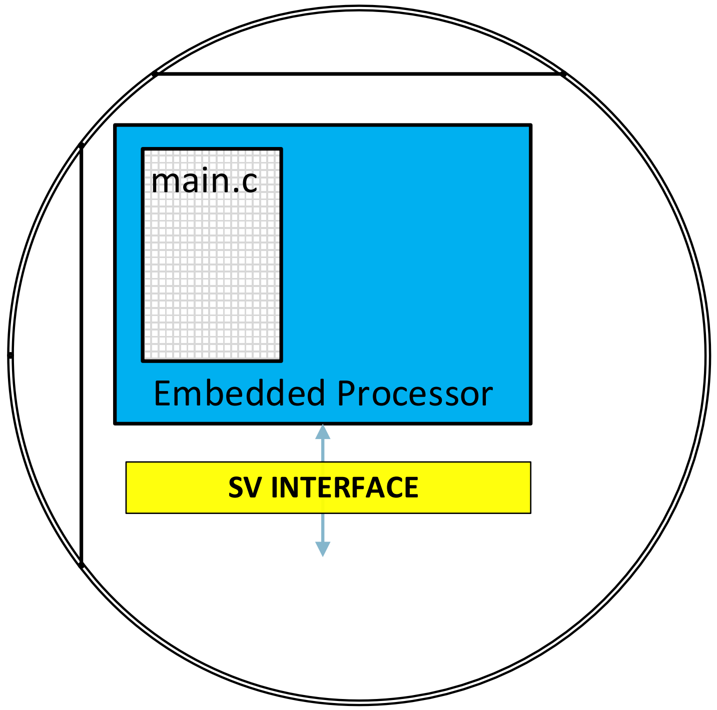
↓

Load the hex file to the iRAM. Done!

```
task load_lcp_mem();

  $readmemh(iram0_filename   , `EMB_PROC.u_lcp_xtmem.iram0

  $readmemh(iram0_1_filename, `EMB_PROC.u_lcp_xtmem.iram0_1
```

Embedded Processor Integration Problem: multiple tests needed

- Proposed solution – what have we achieved
  - Generate random content for C/ASM tests at before the simulation starts, and load it to the processor.
  - Same method may be implied in order to generate random content on-the-fly

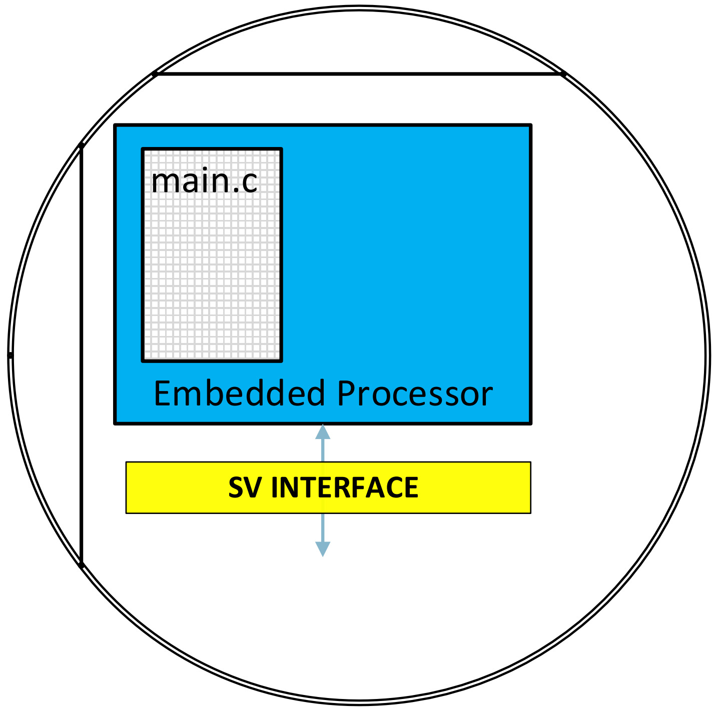


Embedded Processor Integration Problem: complex initialization flow

- Problem 2:

  The initialization flow (boot/ other) might include some complex elements, such as: interrupt handling, PLL setup, memory loads etc.

  Their C code may be complex, and is not needed throughout the ASIC development cycle. (Supplied by FW team close/after TO)

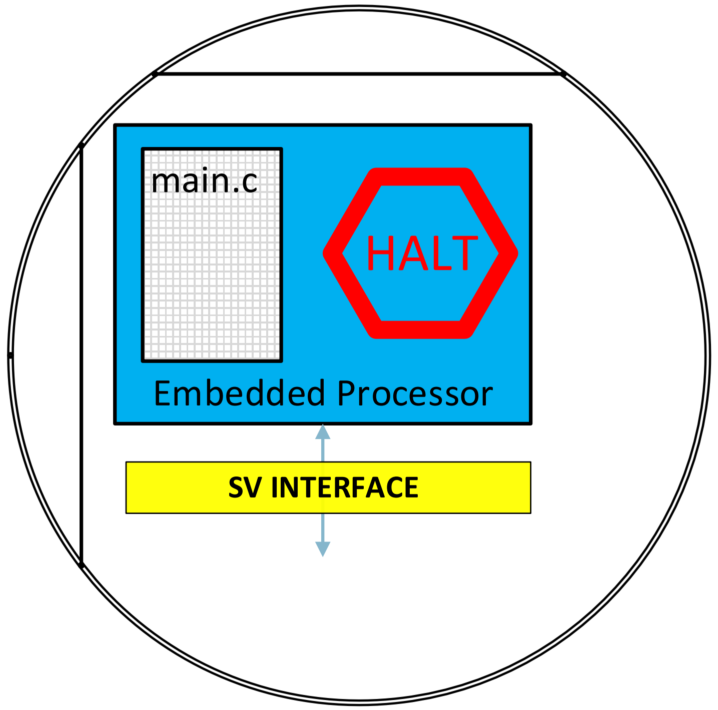


Embedded Processor Integration Problem: complex initialization flow

- Proposed solution:

  HYBRID processor integration flow

  - The first part of the simulation (boot/ init flow) is executed by the environment BFMs.
    During this phase, the processor is halted (PC is not advancing).



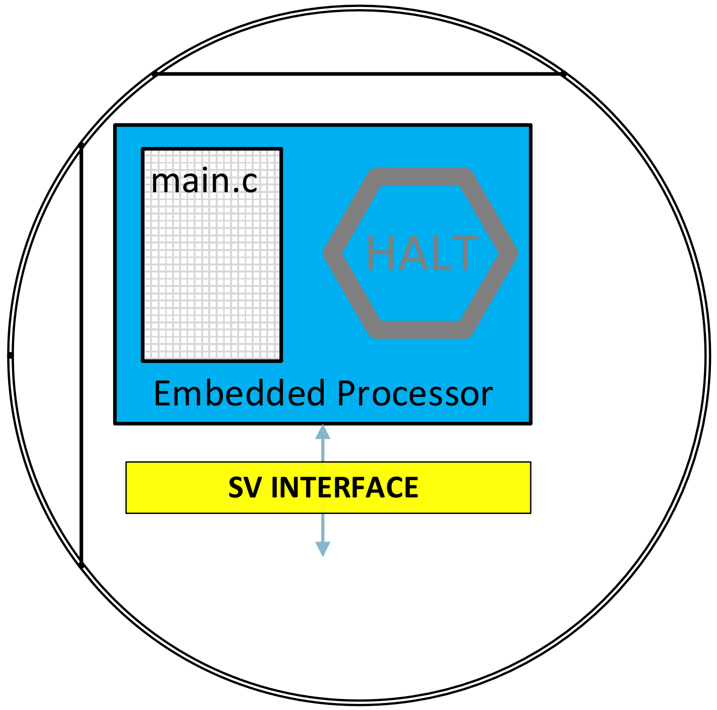
Embedded Processor Integration Problem: complex initialization flow

- Proposed solution:

  HYBRID processor integration flow

  - The first part of the simulation (boot/ init flow) is executed by the environment BFMs.
    During this phase, the processor is halted (PC is not advancing).

  - The second part, is performed by the processor, after being loaded on-the-fly with generated random <u>content</u>, and released from halt.

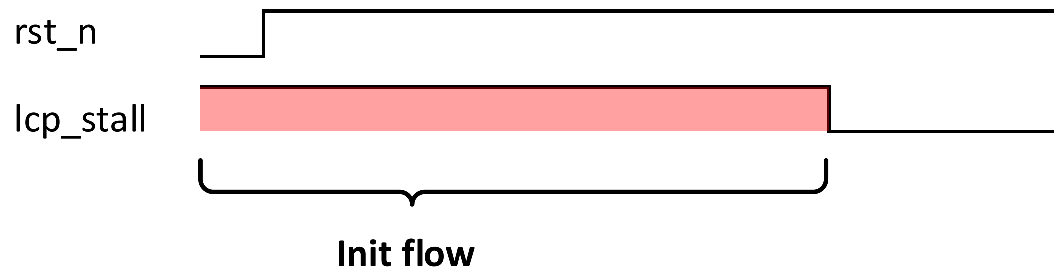


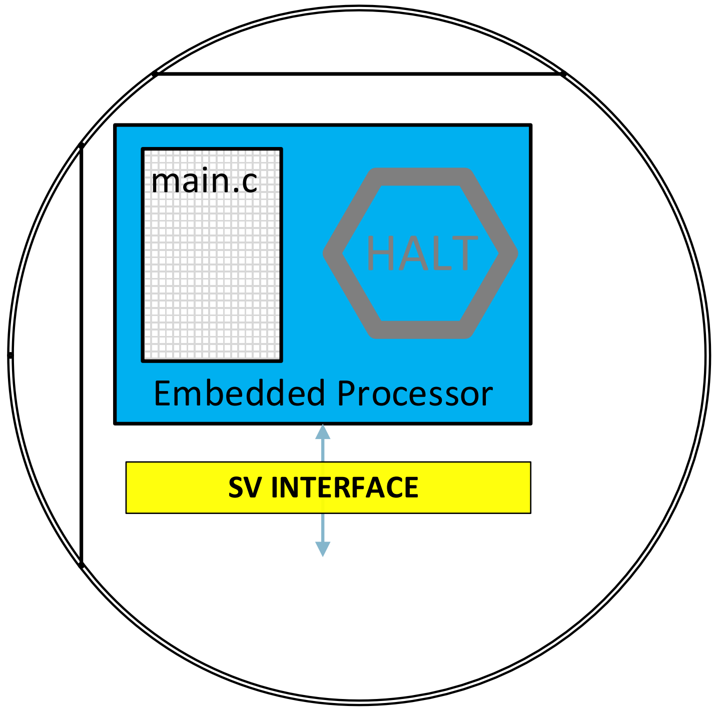
Embedded Processor Integration Problem: complex initialization flow

- Proposed solution:

  HYBRID processor integration flow

```
assign (supply1,supply0) `PATH_LCP_PROC.HADDR = lcp_stall ? ahb_lcp_if.master_if[0].haddr  : 'hz;
```





Summary

- In order to eliminate the need of writing multiple integration C tests:
  - Use the environment’s infrastructure and constraints to generate random <u>content</u> – and create a C file from it.

- In order to avoid coding the complex SOC initialization flow in C:
  - Use the proposed HYBRID flow, and run the first part of the test via the BFMs, and the second part via the processor.

CDNLIVE℠

Cadence User Conference 2019

cadence®