Verification of the PULPino SOC platform using UVM

Mahesh R, Associate engineer
Shamanth H K, Associate engineer

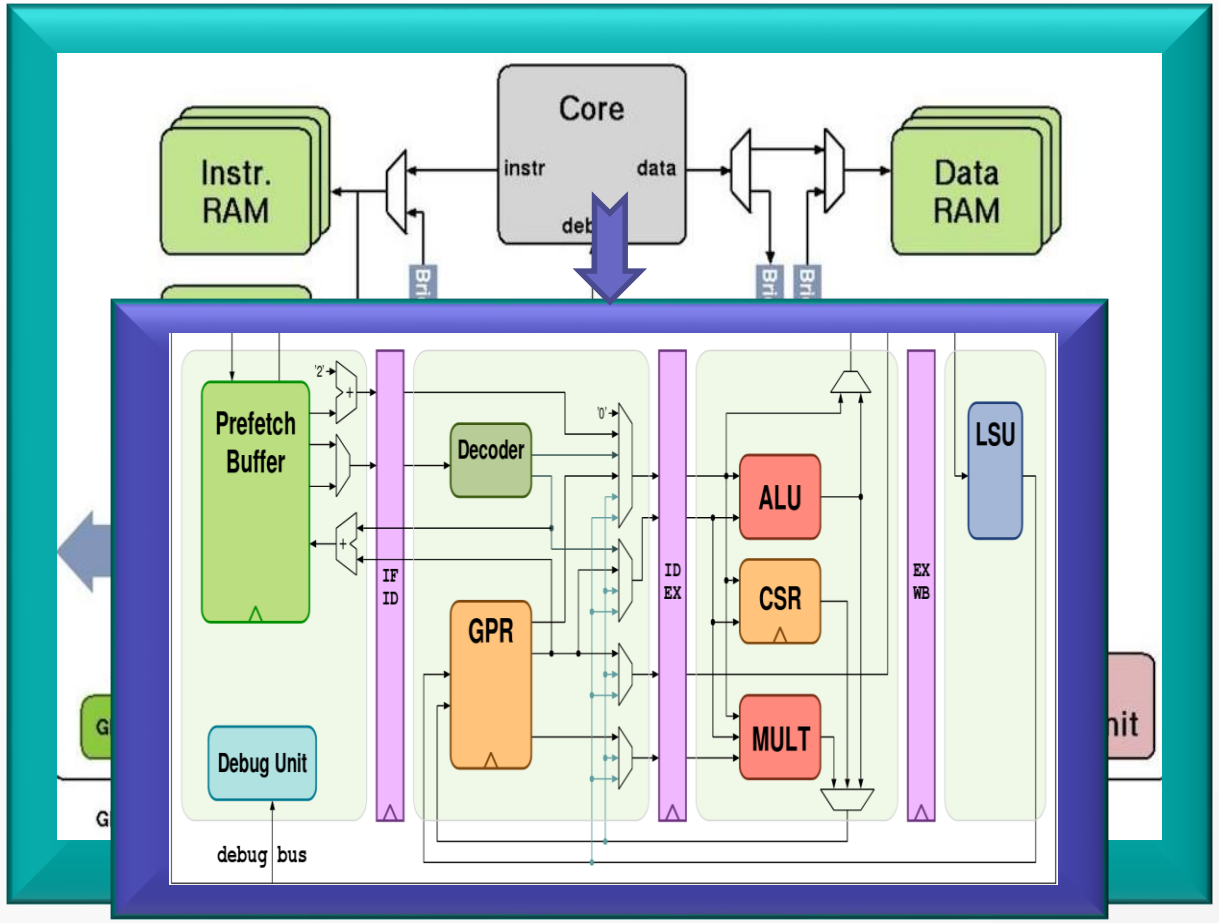
CISMA
(a subsidiary of Verikwest Systems Inc, CA)

RISC-V Workshop
Chennai India (July 2018)

Outline

- PULPino SOC features
- Goals of SOC Verification
- Testbench architecture
- Test flow
- UVM based methodology for external traffic
- High-level C APIs ease SOC test creation
- Interrupt test methodology
- C-UVM coordination: Ending a test
- Summary

PULPino SOC features (www.pulp-platform.org)



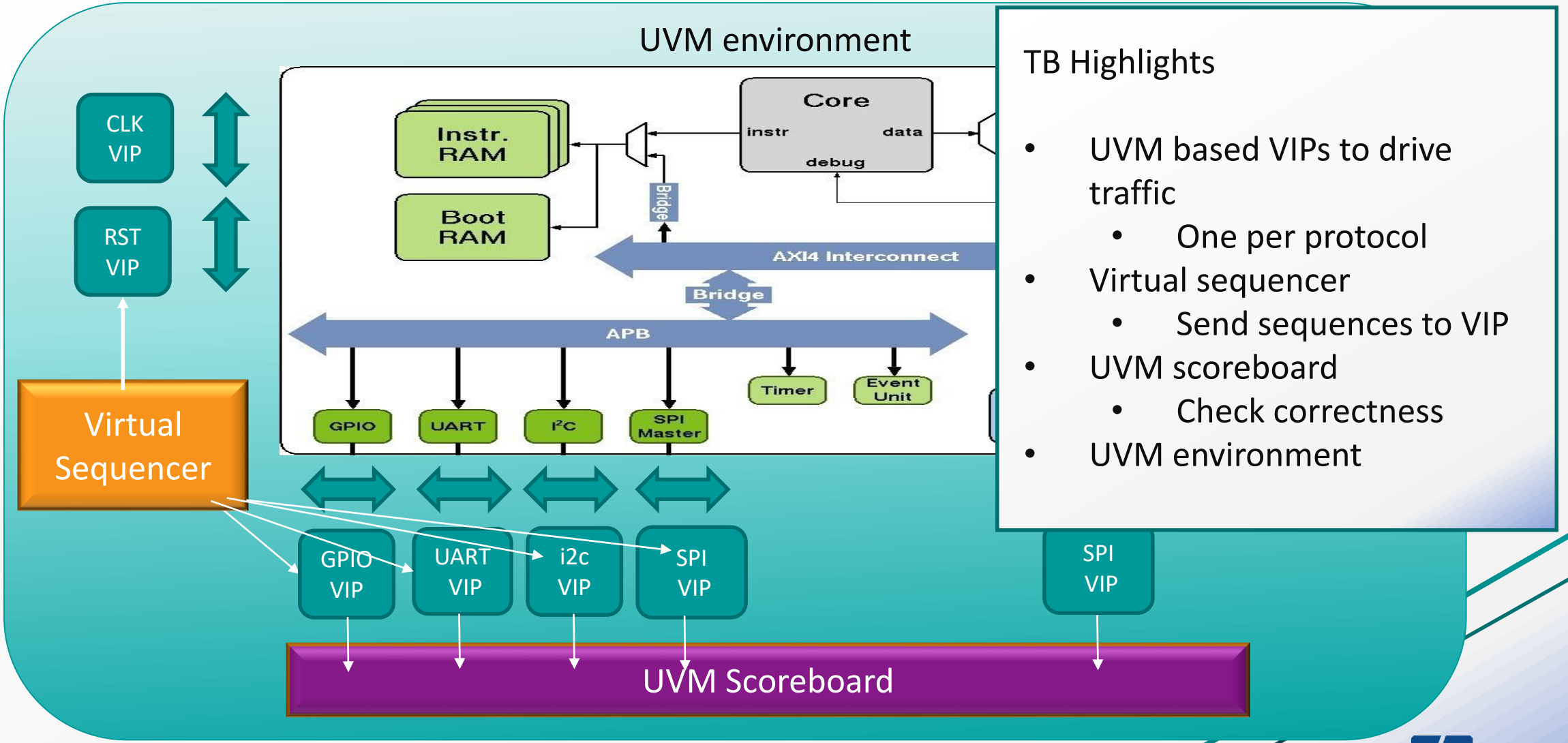
- 32-bit single RISC-V core
  - 4-stage pipeline
  - Extended ISA
    - hardware loops, per load-stores
- Separate Instruction/ Data Memories
  - Single cycle access
- Simple architecture
  - No caches, no DMA
- AXI central interconnect
- APB for peripherals
- Several peripherals
  - I2C, SPI, GPIO and UART

Goals of SOC Verification

- Interoperability of the CPUs, Memories and Peripherals with latencies
- Communication paths (reads/writes) between CPU and each peripheral
- Communication paths between peripheral blocks
- Interrupt handling in the SOC

Assumptions: Individual cores have been verified

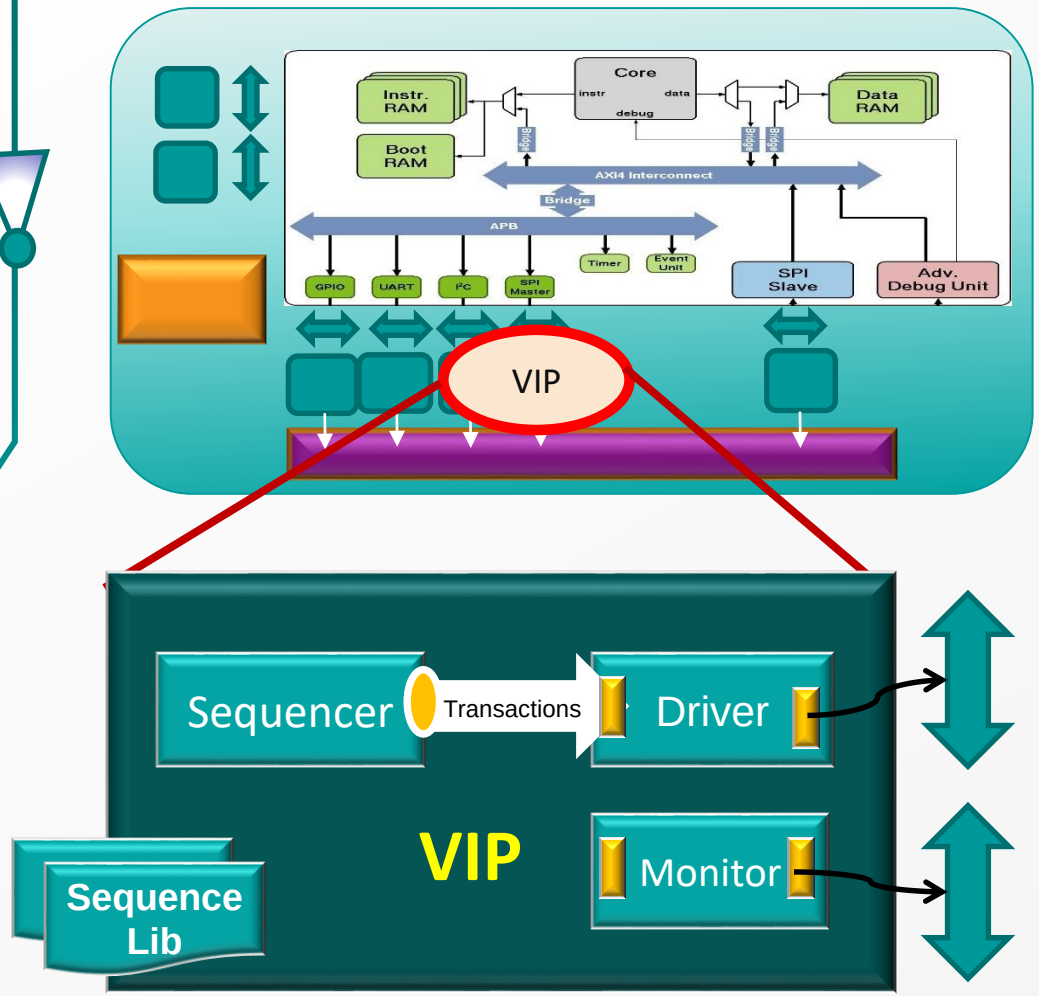
Testbench Architecture



TB Highlights

- UVM based VIPs to drive traffic
  - One per protocol
- Virtual sequencer
  - Send sequences to VIP
- UVM scoreboard
  - Check correctness
- UVM environment

UVM Based Methodology for external traffic

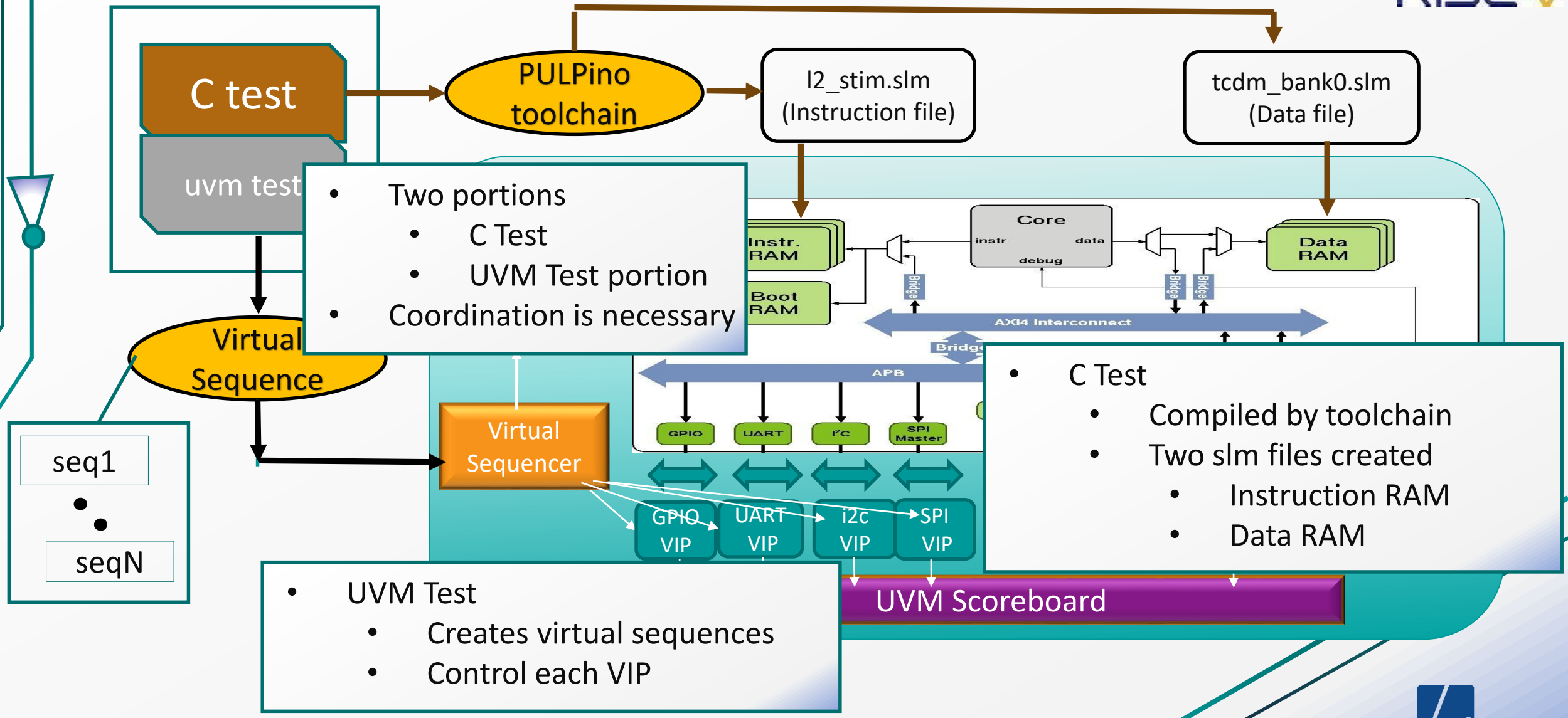


UVM VIPs

- QSPI
- I2C
- UART
- GPIO
- CLK
- RST

- IEEE standard methodology for block level verification
- VIPs generate traffic sequences to the SOC
  - based on the protocol
- Each VIP contains
  - uvm_driver for driving transactions through SV interfaces
  - uvm_monitor for monitoring activities on the bus
  - uvm_sequencer for scheduling the sequences based on test intent

Test Flow



- Two portions
  - C Test
  - UVM Test portion
- Coordination is necessary

- C Test
  - Compiled by toolchain
  - Two slm files created
    - Instruction RAM
    - Data RAM

- UVM Test
  - Creates virtual sequences
  - Control each VIP

High-level C APIs ease SOC test creation

DataBuffer Access API

```
allocate_buffer(int size);
check_resource_table(dataBuffer_t buffer);
store_databyteArray_in_buffer(dataBuffer_t write_buff
store_databyte_in_buffer(dataBuffer_t write_buffer, ch
store_datawordArray_in_buffer(dataBuffer_t write_buf
store_dataword_in_buffer(dataBuffer_t writeBuffer, int
get_buffer_length(dataBuffer_t dataBuffer);
get_buffer_size(dataBuffer_t dataBuffer);
get_buffer_address(dataBuffer_t dataBuffer);
get_buffer_address_pointer(dataBuffer_t dataBuffer);
get_buffer_address_offset_pointer(dataBuffer_t dataBu
reset_buffer(dataBuffer_t buffer);
free_buffer(dataBuffer_t buffer);
copy_buffer(dataBuffer_t from_buffer, dataBuffer_t to_
read_word_from_buffer(dataBuffer_t read_buffer);
read_byte_from_buffer(dataBuffer_t read_buffer);
read_from_buffer_complete(dataBuffer_t read_buffer)
update_bytes_stored_in_buffer(dataBuffer_t writeBuff
dump_buffer(dataBuffer_t buffer);
data_mismatch_in_buffers(dataBuffer_t data_buffer1,
```

Interrupt test API

```
disable_all_interrupts();
enable_all_interrupts();
enable_i2c_interrupt();
enable_qspi_interrupt();
enable_uart_interrupt();
enable_gpio_interrupt();
```

```
int main()
{
        .
        .
        .
        i2c_start_write_transfe r();
        i2c_transmit_data  (writeByteBuffer);
        i2c_end_transfer ();

        i2c_start_read_transfer();
        i2c_receive_data  (readByteBuffer);
        i2c_end_transfer ();

        return 0;
 }
```

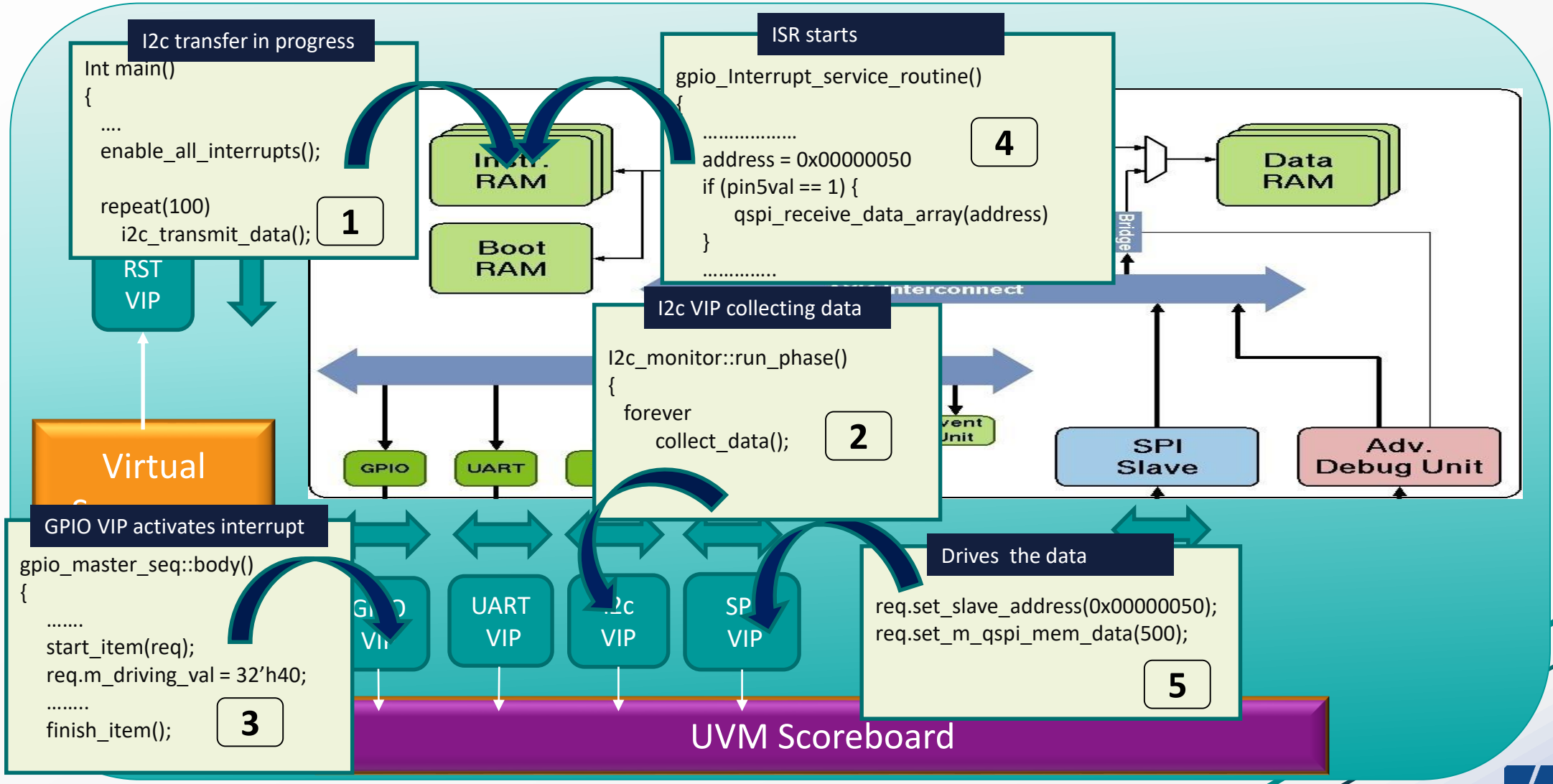
i2C Access API

```
e_buffer);
buffer);
```

```
te_buffer,int offset);
te_buffer);
lBuffer, int offset);
lBuffer);
```

SPI Access API

Interrupt test methodology

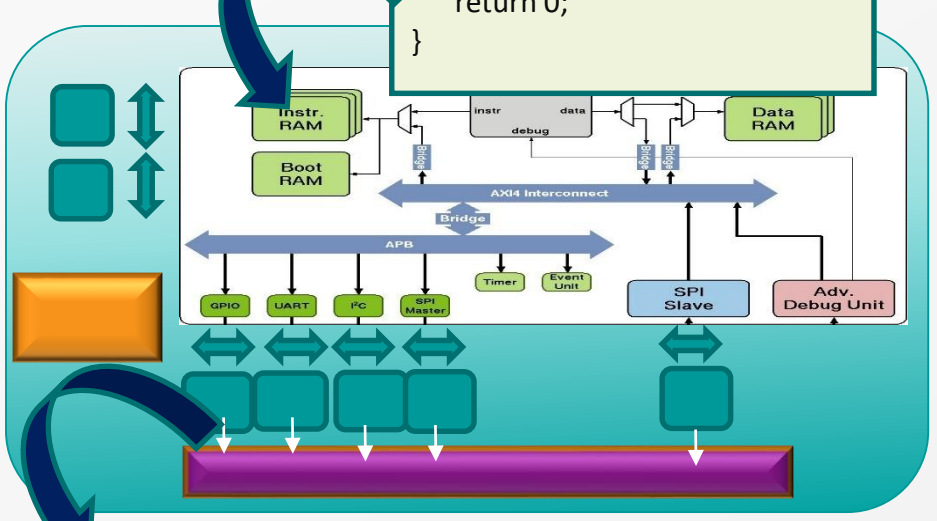


C-UVM coordination: Ending a test

- Toggling a unique GPIO pin
  - C test completes data transfer and toggles a GPIO pin
  - GPIO VIP drops objection (in UVM) to end the test
- Sending unique “halt” string to the UART peripheral
  - C test completes data transfer and sends a unique “halt” string to UART
  - UART VIP drops objection (in UVM) to end the test

set_gpio_pin()

```
Int main()
{
    // perform test logic
    set_gpio_pin();
    return 0;
}
```



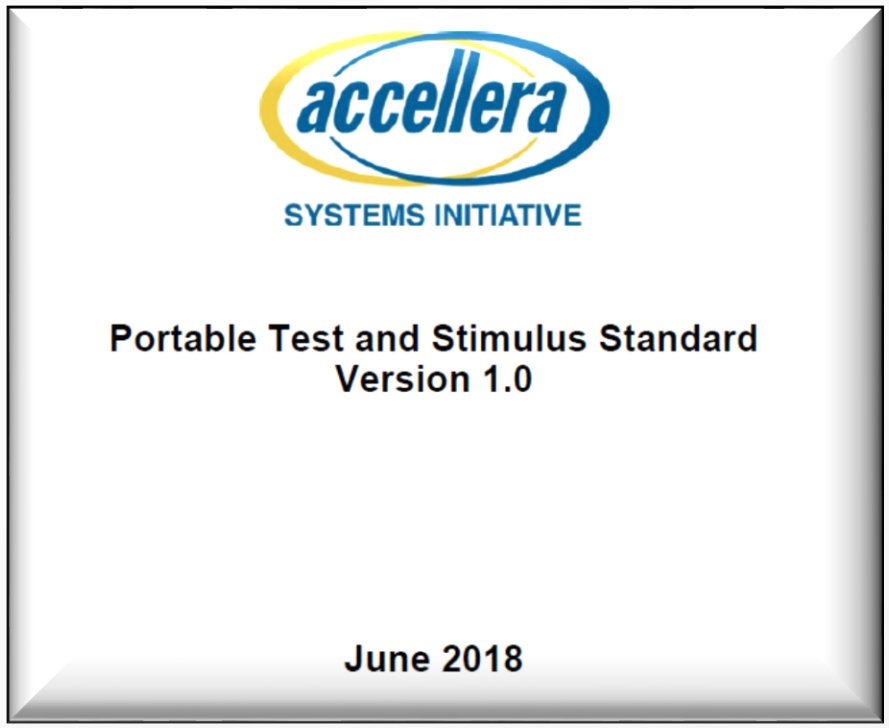
phase_drop_objection()

```
task run_phase(); begin
    phase.raise_objection(this);
    .
    .
    if( interface.gpio_pin == 1'b1)
        phase.drop_objection();
end
```

Summary

- Created 35+ high level APIs for C test
- Created all required UVM VIPs for testing
- 30+ test cases
  - Read and write operation on interfaces
    - bytes, words
  - Interrupt tests
- UVM – C coordination
  - Ending tests

Going forward..



- Use Portable Stimulus Technology to capture test intent
- Generate more complex test scenarios using inferencing in PSS
- Use coverage metrics for pruning unnecessary test cases