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

- 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

C test
  • Two portions
    • C Test
    • UVM Test portion
  • Coordination is necessary

uvm test

Virtual Sequence
  • seq1
  • seqN

PULPino toolchain
  • I2_stim.slm (Instruction file)
  • tcdm_bank0.slm (Data file)

Virtual Sequencer

UVM Scoreboard
  • 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_buffer, char *byteArray, int byteArrayLength);
store_datawordArray_in_buffer(dataBuffer_t write_buffer, int *wordArray, int wordArrayLength);
get_buffer_length(dataBuffer_t buffer);
get_buffer_size(dataBuffer_t buffer);
get_buffer_address(dataBuffer_t buffer);
get_buffer_address_pointer(dataBuffer_t buffer);
get_buffer_address_offset_pointer(dataBuffer_t buffer, int offset);
reset_buffer(dataBuffer_t buffer);
free_buffer(dataBuffer_t buffer);
copy_buffer(dataBuffer_t from_buffer, dataBuffer_t to_buffer);
read_word_from_buffer(dataBuffer_t buffer);
read_byte_from_buffer(dataBuffer_t buffer);
read_from_buffer_complete(dataBuffer_t buffer);
update_bytes_stored_in_buffer(dataBuffer_t buffer, int numBytes);
dump_buffer(dataBuffer_t buffer);
data_mismatch_in_buffers(dataBuffer_t buffer1, dataBuffer_t buffer2);

i2C Access API

i2c_start_read_transfer();
i2c_transmit_data(writeByteBuffer);
i2c_end_transfer();
i2c_start_read_transfer();
i2c_transmit_data(readByteBuffer);
i2c_end_transfer();

Interrupt test API

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

SPI Access API

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

int main()
{
    
    
    i2c_start_write_transfer();
i2c_transmit_data(writeByteBuffer);
i2c_end_transfer();
i2c_start_read_transfer();
i2c_transmit_data(readByteBuffer);
i2c_end_transfer();
    
    return 0;
}
Interrupt test methodology

1. **GPIO VIP activates interrupt**
   
   ```cpp
   gpio_master_seq::body()
   {
       ....
       start_item(req);
       req.m_driving_val = 32'h40;
       ....
       finish_item();
   }
   ```

2. **I2c monitor::run_phase()**
   
   ```cpp
   I2c_monitor::run_phase()
   {
       forever
       collect_data();
   }
   ```

3. **ISR starts**
   
   ```cpp
   gpio_interrupt_service_routine()
   {
       address = 0x00000050
       if (pin5val == 1) {
           qspi_receive_data_array(address)
       }
   }
   ```

4. **I2c transfer in progress**
   
   ```cpp
   enable_all_interrupts();
   repeat(100)
   i2c_transmit_data();
   ```

5. **Drives the data**
   
   ```cpp
   req.set_slave_address(0x00000050);
   req.set_m_qspi_mem_data(500);
   ```
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

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

```verilog
set_gpio_pin()
```

```verilog
phase_drop_objection()
```
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