Accelerated simulation through design partition and HDL to C++ compilation

Theta Yang, Sga Sun
Typical usage

• Generate cycle-accurate behavioral model from design description in Verilog,
  – Create architecture/performance modeling
• Quick prototype of a design through mixed C++ model and Verilog model
  – Simulate without a HDL simulator
• Behavior verification with *verilated* design in pure C++ based environment
Flow

• Build accelerated design simulation model through design partition and HDL to C++ compilation,
  1. A large chip design is partitioned into smaller blocks
  2. Using Verilator, an open source Verilog compiler and simulator, to converted blocks into its C++
  3. Compile each model into shared object library with generated DPI wrapper or C++ wrapper
  4. Invoke Verilated design model in HDL or C++ based simulation
Verilator Intro

- an open source software which converts Verilog to a model in C++ or SystemC
  - restricted to modeling the synthesizable subset of Verilog
  - generated models are cycle-accurate and 2-state
  - Optimize the design to generate high performance simulation model (flatten, unroll)

Fig 1. Diagram of Verilator generated block model
“Verilated” Design Models

- With ‘Verilator’ as compilation tool
  - invoked with parameters similar to Cadence Verilog-XL/NC-Verilog, or Synopsys's VCS
  - Support C++ or SystemC target

```cpp
// Vconvert.h, C++ format
#include "verilated.h"
class Vconvert :
public VerilatedModule
{
public:
  // PORTS
  unsigned char clk;
  unsigned int data;
  unsigned int out;

public:
  Vconvert(const char* name);
  virtual ~Vconvert();

  void eval();
}
// Vconvert.h, systemc format
#include "systemc.h"
#include "verilated_sc.h"
#include "verilated.h"
SC_MODULE(Vconvert) {
  public:
  // PORTS
  sc_in<bool> clk;
  sc_in<uint32_t> data;
  sc_out<uint32_t> out;

  public:
  SCCTOR(Vconvert);
  virtual ~Vconvert();

  private:
  void eval();
}
```
Design partition

- Based on AMD in house design connectivity tool
  - Design stitching automation
  - Design hierarchy management
  - IP-XACT compatible
Create a Simulation Model

Original Design

- IP1.v
- IP2.v
- IP3.v
- TOP.v

Partitioned Design

- Part1.v: IP1 + IP2
- Part2.v: IP3
- TOP.v

Design partition

verilator

Verilated model for HDL simulation

- Part1_verilated.so
- Part1_dpi.sv
- Part2_verilated.so
- Part2_dpi.sv
- TOP.v

Verilated model for C++ simulation

- Part1_verilated.so
- Part2_verilated.so
- TOP.cpp
Two Simulation Mode

• HDL Simulation Mode
  – Verilog stub & DPI for each Verilated model generated with script
  – Verilated design model is invoked through SV DPI when simulating
  – Verilog and C++ co-simulation through DPI wrapper

• C++ Simulation Mode (HDL simulator is not required)
  – C++ wrapper for each Verilated model generated with script
  – The C++ based top-level model is used to
    • Manage top level ‘wire’, which passes input/output data between ‘Verilated’ models
    • Manage simulation time
    • For TICK, Invoke callback functions for each Verilated model wrapper
module eth_top (...);

initial
dpi__eth_top__init("eth_top");

taxt T Hick:
//sample input
m_wb_dat_r  = m_wb_dat_i;
w_b_dat_r  = wb_dat_i ;

//eval
dpi__eth_top__input(
m_wb_dat_r ,
w_b_adr_r ,
...)

dpi__eth_top__output(
m_wb_adr_w ,
...
);

//drive output
m_wb_adr_o = m_wb_adr_w ;
...
endtask
always @(posedge wb_clk_i) begin
TICK;
end

extern "C" {
  Veth_top *pMod;

  void dpi__eth_top__init (const char* name) {
    pMod = new Veth_top("eth_top");
  }

  void dpi__eth_top__input(
    unsigned int m_wb_dat_i,
    unsigned int wb_dat_i ,
    ...
  ) {
    pMod->m_wb_dat_i  = m_wb_dat_i ;
pMod->wb_dat_i  = wb_dat_i ;
...
pMod->wb_clk_i = 1;
pMod->eval();
  }

  void dpi__eth_top__output(
    unsigned int *m_wb_adr_o,
    ...
  ) {
    pMod->wb_clk_i = 0;
pMod->eval();
    *m_wb_adr_o  = pMod->m_wb_adr_o ;
...
  }
}
C++ Wrapper Example

Verilog top level

```verilog
wire clk;
wire [7:0] x;
wire [7:0] y;
wire [15:0] mult;
wire [23:0] sum;

mult mult_u0(clk, x, y, mult);
add add_u0(clk, mult, sum);
```

C++ Wrapper

```c
class VerilogWrapper {
public:
    VerilogWrapper(const char *path, const unsigned int cb,
                   const unsigned char clk, const unsigned short m,
                   unsigned short &sum);

    void INIT() {
        // INIT code
    }

    void TICK() {
        // TICK code
    }
};
```

C++ Top level

```c
void adder_wrapper(const char *path,
                    const unsigned int cb,
                    const unsigned char clk,
                    const unsigned short m,
                    unsigned short &sum) {
    switch(cb) {
    INIT: dpi__Vadder__init(path); break:
    INPUT: dpi__Vadder__input(path, clk, m); break:
    OUTPUT: dpi__Vadder__output(path, &sum); break:
    }
}
```
C++ Top Level Scheduler

Initialization

ModelCreate → Clock/Reset setup

Callback Stage in a Simulation Cycle

ClockEvaluation

TestPutSignal

InputMonitor

ModelEvaluation

OutputDrive

TestGetSignal

Complete?

Check

Cleanup

Unified Cycled Based Simulation Model

Verilated Design

Behavioral Models

Testbench Components

C++ Top Level
Simulation time comparison on MIPS core design, run set of tests with verilated design and original Verilog: ~4 times speedup
Performance Profiling (Complex Design)

- 12M cell instances
- UVM Testbench support PIPE mode and serial mode
- PHY model is not synthesizable, so cannot be verilated.
Progress on Complex Design

• Make DUT compliable by Verilator
  – Replace Foundry Memory Model with Synthesizable Memory Model
  – Changed instantiated library cell (clock gater, sync cell) to Synthesizable behavioral model
  – PHY is excluded

• Whole Design has been partitioned into 13 blocks, and verilated

• Simulation Result (Still working in progress)
  – CPS of original simulation CPS in UVM (27 cycles/second)
  – CPS of verilated C++ model in bare run (42 cycles/second)

3/1/2022
Theta Yang, Sga Sun – AMD Inc.
Further improvement

• Further analyze and optimize the performance of Verilated design block
  – Model computing intensive part of design with C++ directly
  – Optimize design to make Verilated optimize it better
  – Optimize Verilator to support multibit synthesis

• Construct system level behavior modeling
  – Integrate external CPU models and peripheral models
  – Simulate system boot and OS

• Use Verilated models for pure C++ based verification
  – Leverage UVM-SystemC
Questions?