Embedded UVM

Puneet Goel <puneet@coverify.com>

COVERIFY

September 15, 2017

accellera SYSTEMS INITIATIVE™

2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION INDIA

Embedded UVM?

- Complete port of UVM-1.2 release including the reg-package
- Natively Compiled into:
  - ELF executables that can run on any processor
  - Shared libraries that can be loaded into any application (including Verilog/VHDL Simulations)
- Multicore Powered:
  - Multiple posix threads running concurrently
  - Allows multiple **`uvm_root`** instances and even mulitple simulators
- Systems Programming Enabled:
  - ABI compatibility with C/C++
  - LLVM based compiler - compiles to LLVM IR
- **Embedded UVM** is NOT **SystemVerilog**

Embedded UVM?

- Complete port of UVM-1.2 release including the reg-package
- Natively Compiled into:
  - ELF executables that can run on any processor
  - Shared libraries that can be loaded into any application (including Verilog/VHDL Simulations)
- Multicore Powered:
  - Multiple posix threads running concurrently
  - Allows multiple **`uvm_root`** instances and even mulitple simulators
- Systems Programming Enabled:
  - ABI compatibility with C/C++
  - LLVM based compiler - compiles to LLVM IR
- **Embedded UVM** is NOT **SystemVerilog**

Embedded UVM?

- Complete port of UVM-1.2 release including the reg-package
- Natively Compiled into:
  - ELF executables that can run on any processor
  - Shared libraries that can be loaded into any application (including Verilog/VHDL Simulations)
- Multicore Powered:
  - Multiple posix threads running concurrently
  - Allows multiple **`uvm_root`** instances and even mulitple simulators
- Systems Programming Enabled:
  - ABI compatibility with C/C++
  - LLVM based compiler - compiles to LLVM IR
- **Embedded UVM** is NOT **SystemVerilog**

Embedded UVM?

- Complete port of UVM-1.2 release including the reg-package
- Natively Compiled into:
  - ELF executables that can run on any processor
  - Shared libraries that can be loaded into any application (including Verilog/VHDL Simulations)
- Multicore Powered:
  - Multiple posix threads running concurrently
  - Allows multiple **`uvm_root`** instances and even mulitple simulators
- Systems Programming Enabled:
  - ABI compatibility with C/C++
  - LLVM based compiler - compiles to LLVM IR
- **Embedded UVM** is NOT **SystemVerilog**

Embedded UVM?

- Complete port of UVM-1.2 release including the reg-package
- Natively Compiled into:
  - ELF executables that can run on any processor
  - Shared libraries that can be loaded into any application (including Verilog/VHDL Simulations)
- Multicore Powered:
  - Multiple posix threads running concurrently
  - Allows multiple **`uvm_root`** instances and even mulitple simulators
- Systems Programming Enabled:
  - ABI compatibility with C/C++
  - LLVM based compiler - compiles to LLVM IR
- **Embedded UVM** is NOT **SystemVerilog**

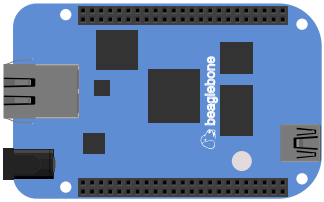
UVM on your RaspberryPi

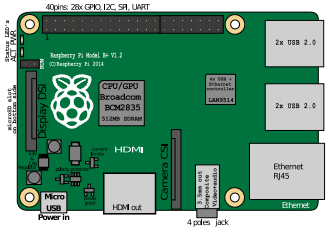
What will you verify?

What will you verify?

Run your UVM testbench on a RaspverryPi or a Beaglebone and Generate

- Constrained Random Ethernet packets that go out on a real ethernet port
- Constrained Random USB packets that emerge from USB connectors of the board
- Test a Beaglebone Cape or a Raspberry Shield that you designed in your Garage
- Stimulate GPIO pins and connect these pins to an Arty FPGA board

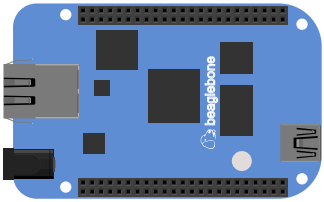


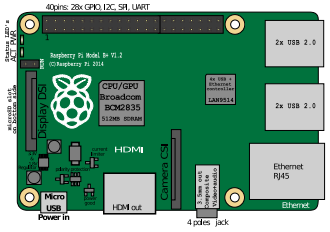


What will you verify?

Run your UVM testbench on a RaspverryPi or a Beaglebone and Generate

- Constrained Random Ethernet packets that go out on a real ethernet port
- Constrained Random USB packets that emerge from USB connectors of the board
- Test a Beaglebone Cape or a Raspberry Shield that you designed in your Garage
- Stimulate GPIO pins and connect these pins to an Arty FPGA board

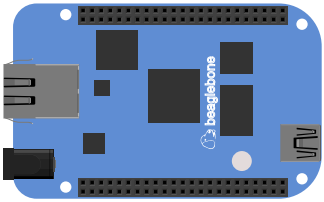


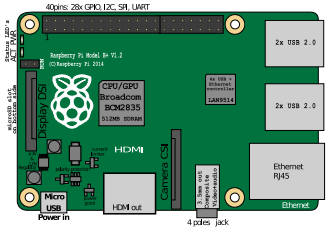


What will you verify?

Run your UVM testbench on a RaspverryPi or a Beaglebone and Generate

- Constrained Random Ethernet packets that go out on a real ethernet port
- Constrained Random USB packets that emerge from USB connectors of the board
- Test a Beaglebone Cape or a Raspberry Shield that you designed in your Garage
- Stimulate GPIO pins and connect these pins to an Arty FPGA board

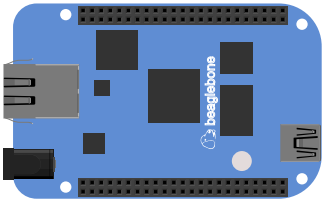




What will you verify?

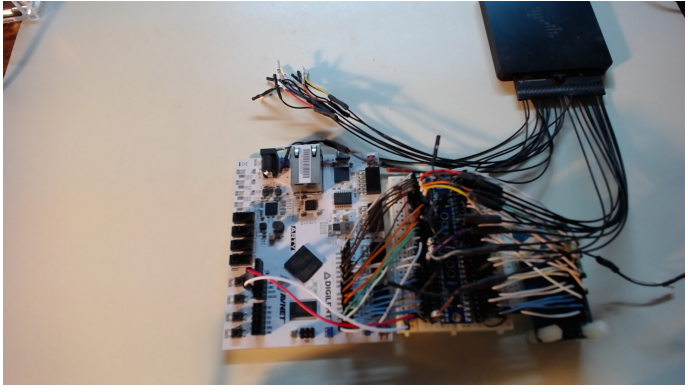
Run your UVM testbench on a RaspverryPi or a Beaglebone and Generate

- Constrained Random Ethernet packets that go out on a real ethernet port
- Constrained Random USB packets that emerge from USB connectors of the board
- Test a Beaglebone Cape or a Raspberry Shield that you designed in your Garage
- Stimulate GPIO pins and connect these pins to an Arty FPGA board



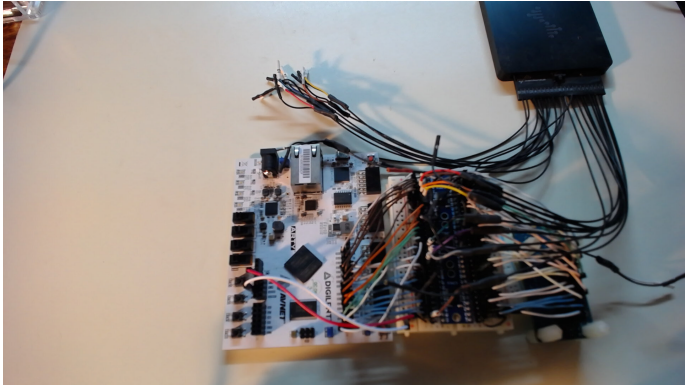
Embedded UVM Powered Emulation

- MRAA library provides user friendly interface to embedded processor's GPIO pins
- But MRAA allows you to handle only one GPIO at a time – Efficiency?

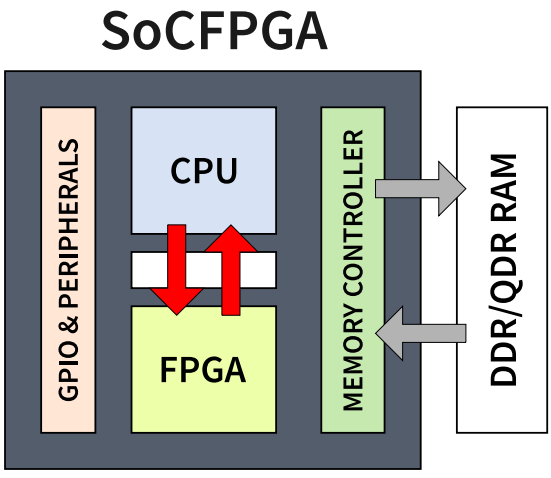


Embedded UVM Powered Emulation

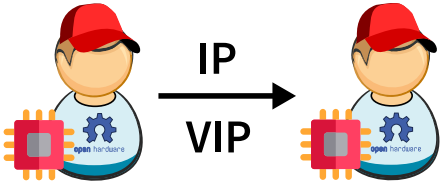
- MRAA library provides user friendly interface to embedded processor's GPIO pins
- But MRAA allows you to handle only one GPIO at a time – Efficiency?



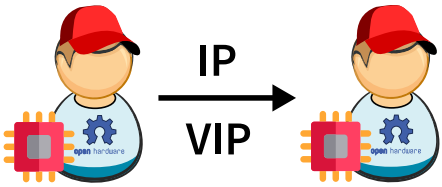
UVM on SoCFPGA – Hardware Accelerators

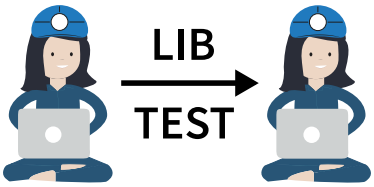


Hardware Accelerators Perspective

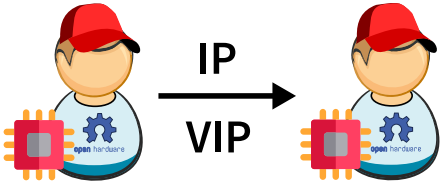


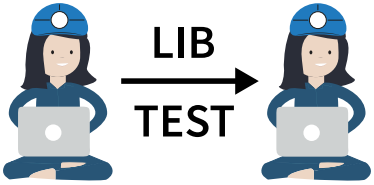
Hardware Accelerators Perspective

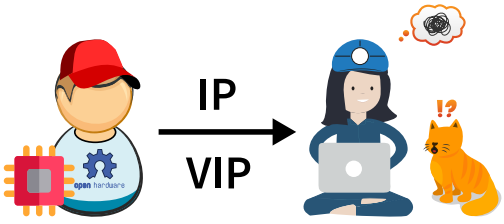




Hardware Accelerators Perspective

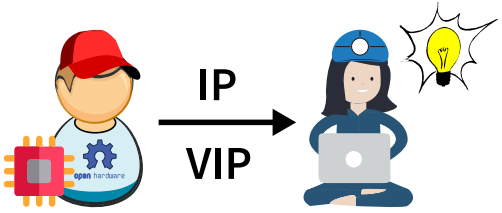






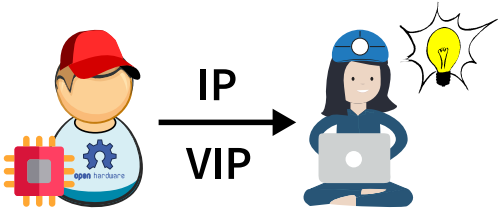
Hardware Accelerators Perspective – Embdded UVM

- Embedded UVM tests the point of deployment
- Allows System Level calls from withing a Testcase
  - For example DUT config part of testcase can be replaced by loading of device driver!!
- Failing tests can be taken back to RTL for debug
- Software testing is often Opensource



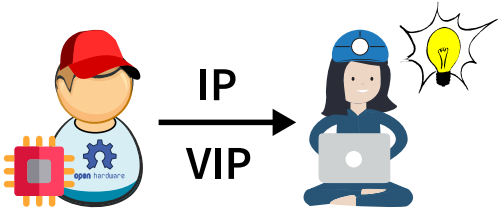
Hardware Accelerators Perspective – Embdded UVM

- Embedded UVM tests the point of deployment
- Allows System Level calls from withing a Testcase
  - For example DUT config part of testcase can be replaced by loading of device driver!!
- Failing tests can be taken back to RTL for debug
- Software testing is often Opensource



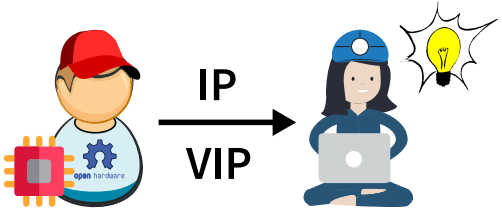
Hardware Accelerators Perspective – Embdded UVM

- Embedded UVM tests the point of deployment
- Allows System Level calls from withing a Testcase
  - For example DUT config part of testcase can be replaced by loading of device driver!!
- Failing tests can be taken back to RTL for debug
- Software testing is often Opensource



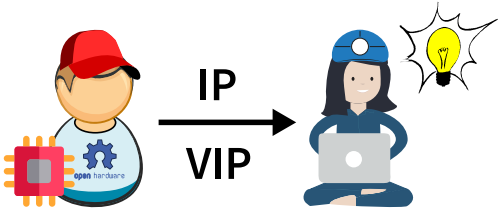
Hardware Accelerators Perspective – Embdded UVM

- Embedded UVM tests the point of deployment
- Allows System Level calls from withing a Testcase
  - For example DUT config part of testcase can be replaced by loading of device driver!!
- Failing tests can be taken back to RTL for debug
- Software testing is often Opensource



Hardware Accelerators Perspective – Embdded UVM

- Embedded UVM tests the point of deployment
- Allows System Level calls from withing a Testcase
  - For example DUT config part of testcase can be replaced by loading of device driver!!
- Failing tests can be taken back to RTL for debug
- Software testing is often Opensource

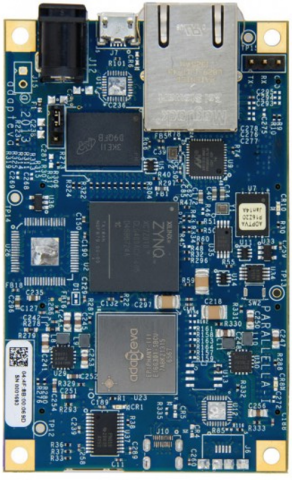


Emulation – Look Ma no SCE-MI

- Low cost, Easy to setup emulation for Small Designs
- Xilinx Zynq - PARALLELLA BOARD (USD 99)
- Intel Cyclone-V - DE10 NANO BOARD (USD 130)

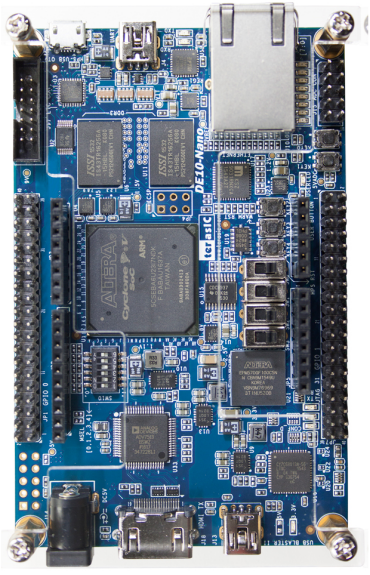
Emulation – Look Ma no SCE-MI

- Low cost, Easy to setup emulation for Small Designs
- Xilinx Zynq - PARALLELLA BOARD (USD 99)
- Intel Cyclone-V - DE10 NANO BOARD (USD 130)



Emulation – Look Ma no SCE-MI

- Low cost, Easy to setup emulation for Small Designs
- Xilinx Zynq - PARALLELLA BOARD (USD 99)
- Intel Cyclone-V - DE10 NANO BOARD (USD 130)

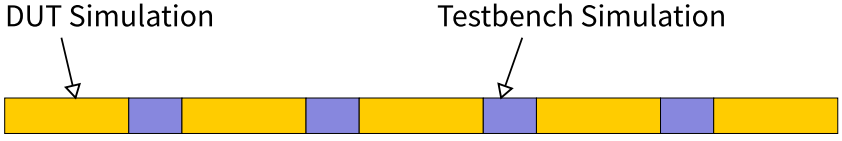


Debug – Back to RTL

- UVM that embeds into…
  - Verilog simulators via DPI and PLI
  - VHDL simulators via FLI and VHPI
  - SystemC simulator
- Provides same stimulus for all the DUT simulators
- And runs parallel to the DUT simulation

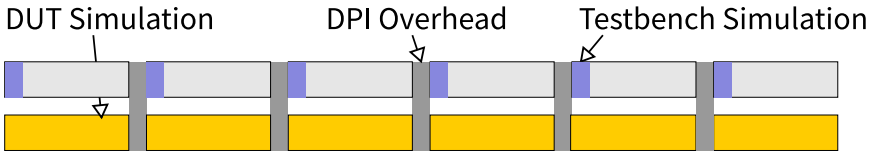
Debug – Back to RTL

- UVM that embeds into…
  - Verilog simulators via DPI and PLI
  - VHDL simulators via FLI and VHPI
  - SystemC simulator
- Provides same stimulus for all the DUT simulators
- And runs parallel to the DUT simulation



Debug – Back to RTL

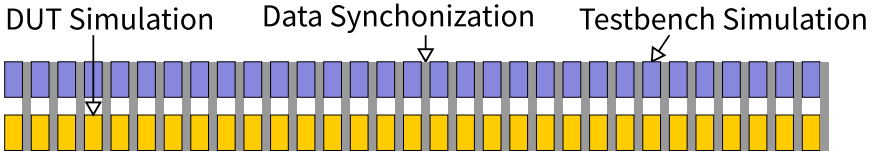
- UVM that embeds into…
  - Verilog simulators via DPI and PLI
  - VHDL simulators via FLI and VHPI
  - SystemC simulator
- Provides same stimulus for all the DUT simulators
- And runs parallel to the DUT simulation



Embedded UVM RTL Cosimulation

Debug – Back to RTL

- UVM that embeds into…
  - Verilog simulators via DPI and PLI
  - VHDL simulators via FLI and VHPI
  - SystemC simulator
- Provides same stimulus for all the DUT simulators
- And runs parallel to the DUT simulation

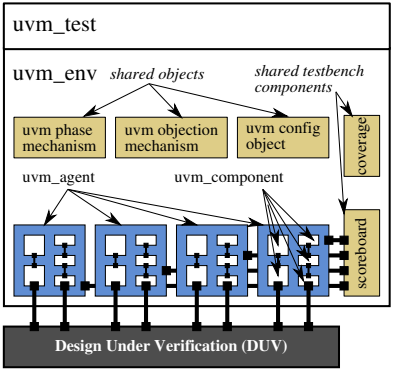


Cosimulation with Virtual/Emulation Platforms

Multicore Simulations

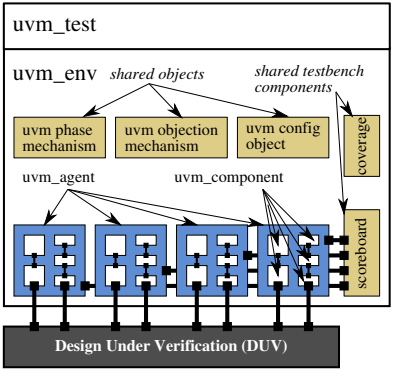
- *The Free Lunch is over*
- The numbers of cores in server processors are projected to grow to hundreds in next 5 years
- Possible to partition RTL structurally because all variables are statically allocated
- No such automatic partioning possible for the dynamically allocated testbench
- Contemporary Verification Languages do not support parallel programming for behavioral code

Embedded UVM is Thread Safe



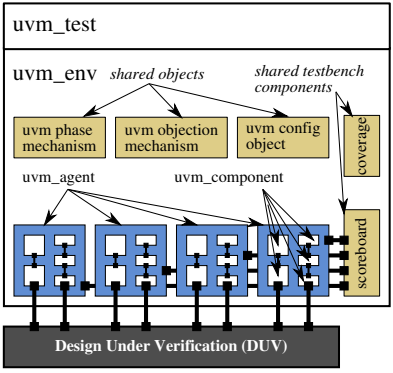
- Embedded UVM takes advantage of the fact that there is minimal interaction between different `uvm_agents`
- UVM constructs (like `uvm_objection`), that are shared between the components, are *synchronized* in the UVM base library implementation
- Abstraction `ParContext` manages parallelization
  - In Embedded UVM, a `uvm_component` implements `ParContext`

Embedded UVM is Thread Safe



- Embedded UVM takes advantage of the fact that there is minimal interaction between different `uvm_agents`
- UVM constructs (like `uvm_objection`), that are shared between the components, are *synchronized* in the UVM base library implementation
- Abstraction `ParContext` manages parallelization
  - In Embedded UVM, a `uvm_component` implements `ParContext`

Embedded UVM is Thread Safe



- Embedded UVM takes advantage of the fact that there is minimal interaction between different `uvm_agents`
- UVM constructs (like `uvm_objection`), that are shared between the components, are *synchronized* in the UVM base library implementation
- Abstraction `ParContext` manages parallelization
  - In Embedded UVM, a `uvm_component` implements `ParContext`

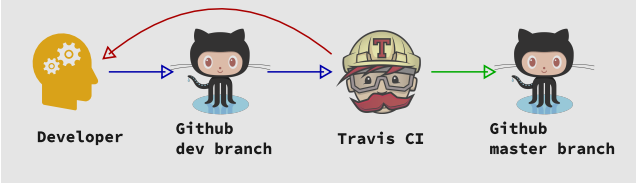
And More…

- SystemVerilog like Verification Features…
  - Garbage Collaction
  - Constrained Randomization
  - Functional Coverage
  - Complete port of UVM including Reg package
- Natively compiled (into .so/ELF executable)
- **Multicore Enabled**
- Multi-simulator and multi `uvm_root`
- Completely Free and Opensource

```
class Foo: uvm_object {
  mixin uvm_object_utils;
  @rand!8 byte[] foo;
  @rand Logic!12 baz;
}
class Bar: Foo {
  mixin uvm_object_utils;
  @rand ubyte[8] bar;
  Constraint! q{
    foo.length > 2; // array
    baz[0..8] == 16;
  } cstFooLength;
  Constraint! q{
    foreach(i, f; bar) f <= i;
    foreach(i, f; foo) {
      if(i > 4) /*condition*/
        f + i < 32 && f > 16;
    }
  } cstFoo;
}
void main() {
```

The Opensource Perspective

- Opensource hardware is fast becoming a platform of choice for many IoT developers
- Typical opensource development flow requires test/verification on the cloud
- Embedded UVM is 100% Opensource



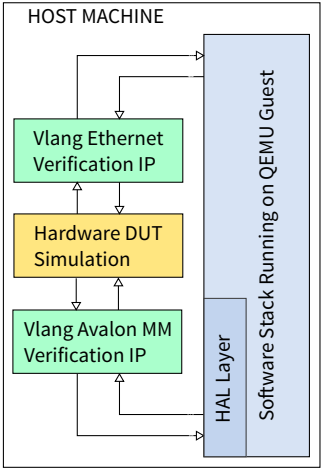
Usecase 1 – SoCFPGA Emulation/Verification

- SHA3 hardware accelerator verification on SoCFPGA and with Icarus..
- Hardware/Software Coverification
- Do it Yourself – Step-by-step instructions on `http://embeddeduvm.com`
- Buy a parallella from `https://www.parallella.org/buy/`
- Or a de10-nano from `http://de10-nano.terasic.com`
- Fork me on Github

```
// start with reset
volatileStore(regs + 3, 0);
auto data = req.phrase;
for (size_t i=0; i!=data.length/4; ++i) {
  uint word = 0;
  for (size_t j=0; j!=4; ++j) {
    word += (cast(uint) data[4*i+j])
      << ((3-j) * 8);
  }
  volatileStore(regs, word);
}
// last byte_num
uint m = cast(uint) (data.length % 4);
volatileStore(regs + 1, m);
// last
uint word = 0;
for (size_t j=0; j!=m; ++j) {
  word += (cast(uint) data[(data.length/4)*4 + j]
    << ((3-j) * 8);
}
volatileStore(regs, word);
```

Usecase 2 – Software Driven Verification

- QEMU is fast becoming the platform of choice for embedded software development and test
- A convenient way to exchange data with QEMU is via shared file descriptors
- Embdded UVM directly taps a file descriptor and feeds the transaction to simulation
- Data coming out of DUT is reverse fed into QEMU



Questions?