ESL-Based Full System Simulation Platform

陳中和

Department of Electrical Engineering
Institute of Computer and Communication Engineering
National Cheng Kung University

Term Project-Preparation

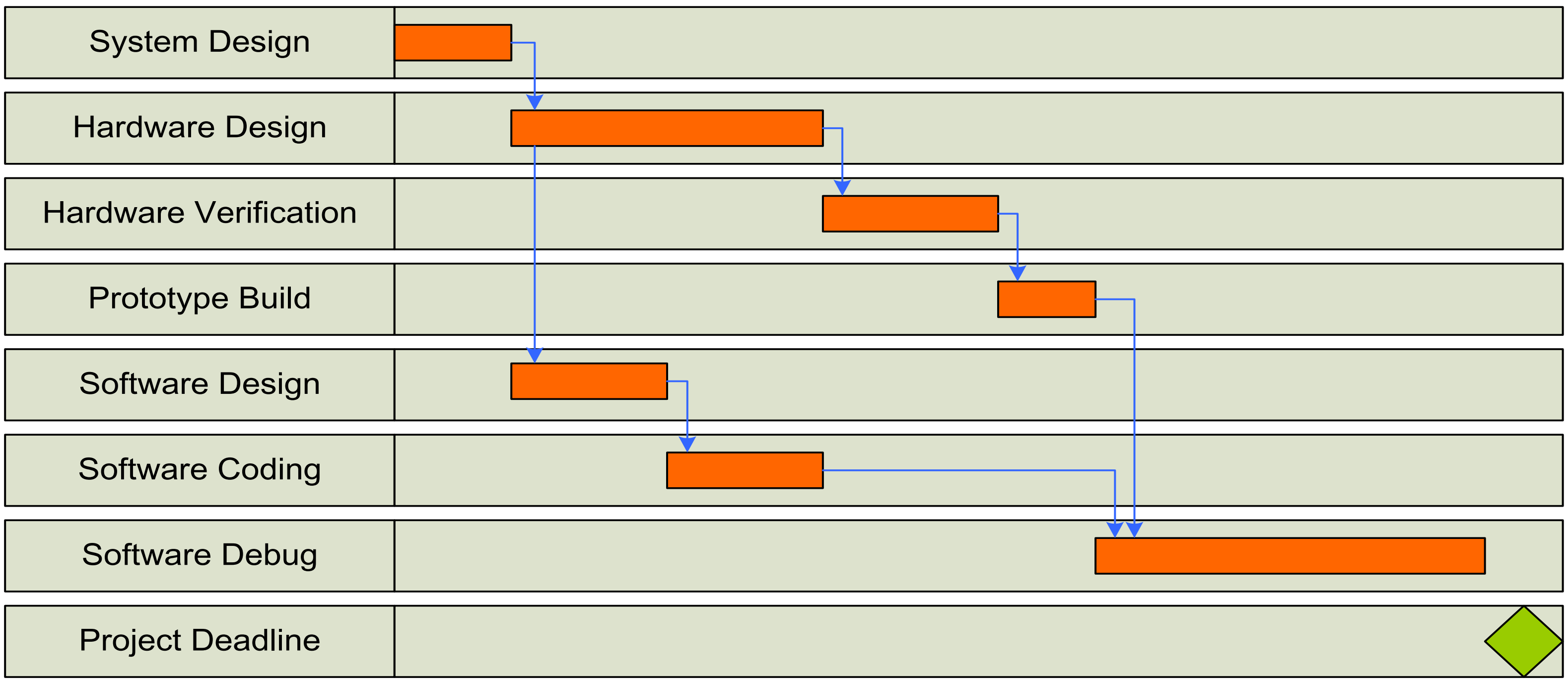
- **Lab1: Building QEMU Experiment al Environment**
- **LAB 2: Building Linux Operating System Environment**
  - Create an environment that boots Linux kernel on ARM Realview EB modeled by QEMU.
- **LAB3: Virtual Machine & Linux Device Driver**
  - Design a virtual hardware running in ARM Realview EB and interacting with Linux device driver and application
- **LAB4: SystemC Module & Full System Simulation using QEMU-SystemC**
- **LAB5: Full System Simulation using QEMU & PlatformArchitect**
  - This lab is not included in this year.

Proposal

- Due in three weeks.
- Proposal report due (11/23)
- Final report and presentation

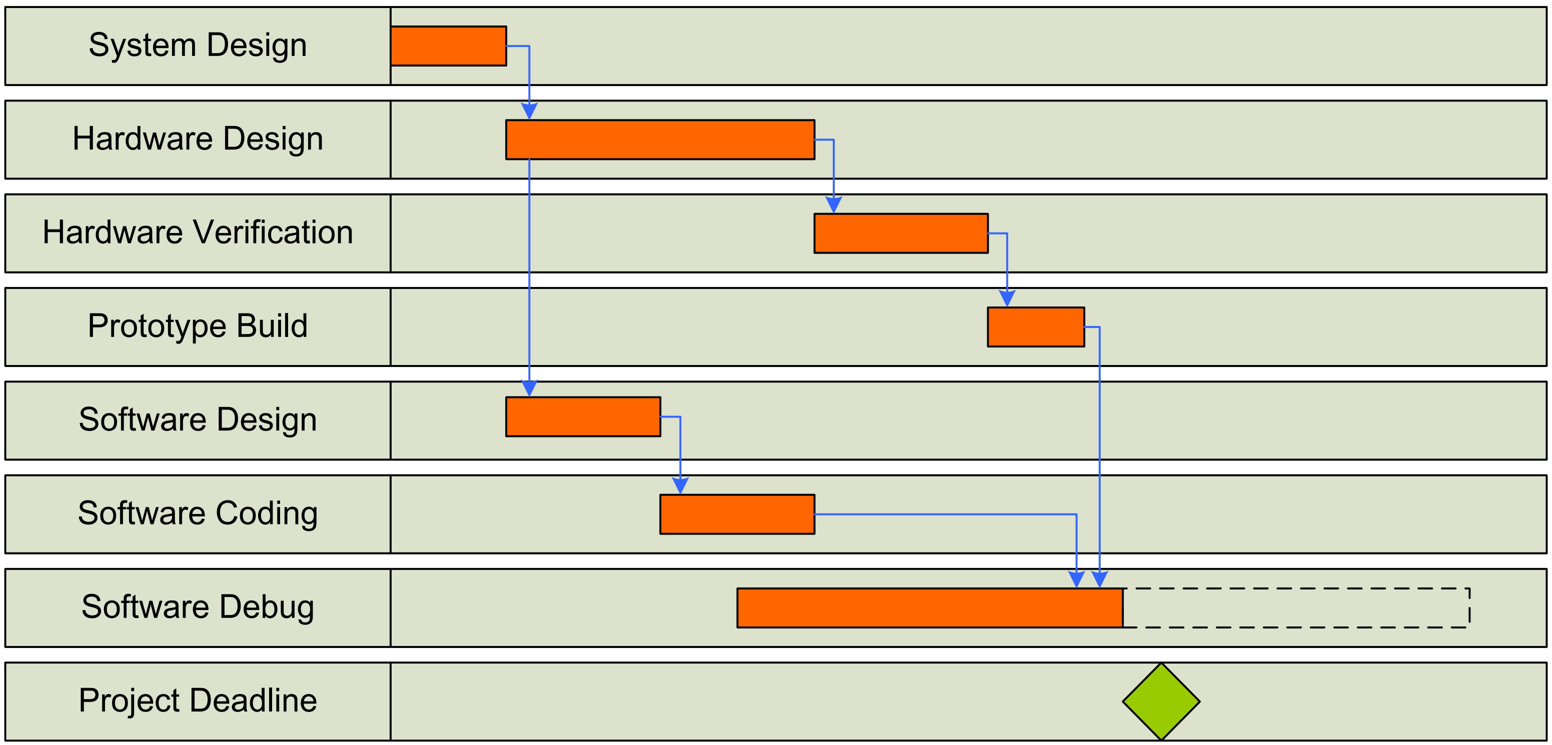
Electronic System Level Design

- Traditional VLSI design flow
  - Software debug begins at late hour.



ESL

- Early interaction with software

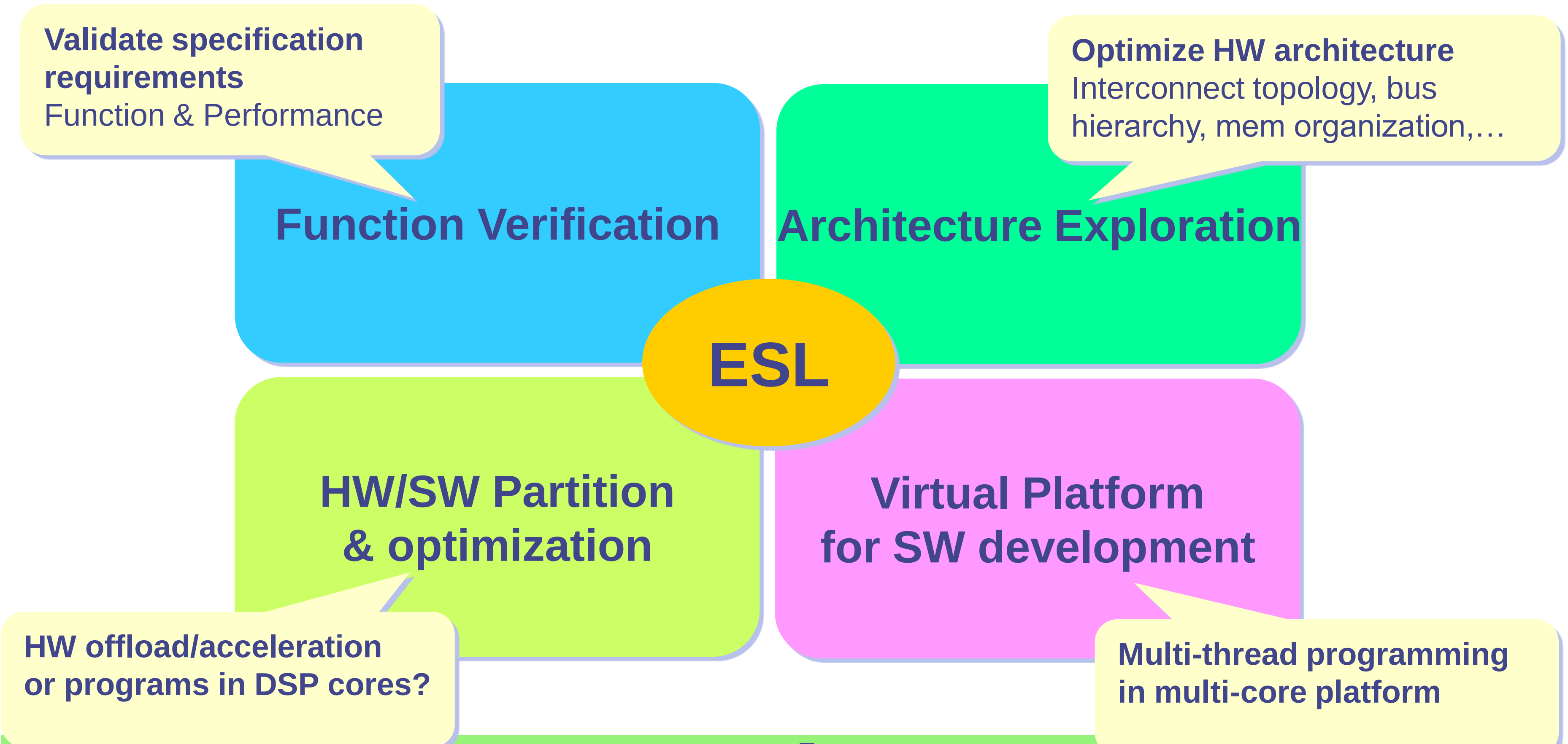


What is Full System Simulation

- Full system simulation platform
  - Hardware : processor cores, memories, interconnection buses, and peripheral devices, ASICs, co-processor, etc.
  - Software : operating system, device drivers, and applications

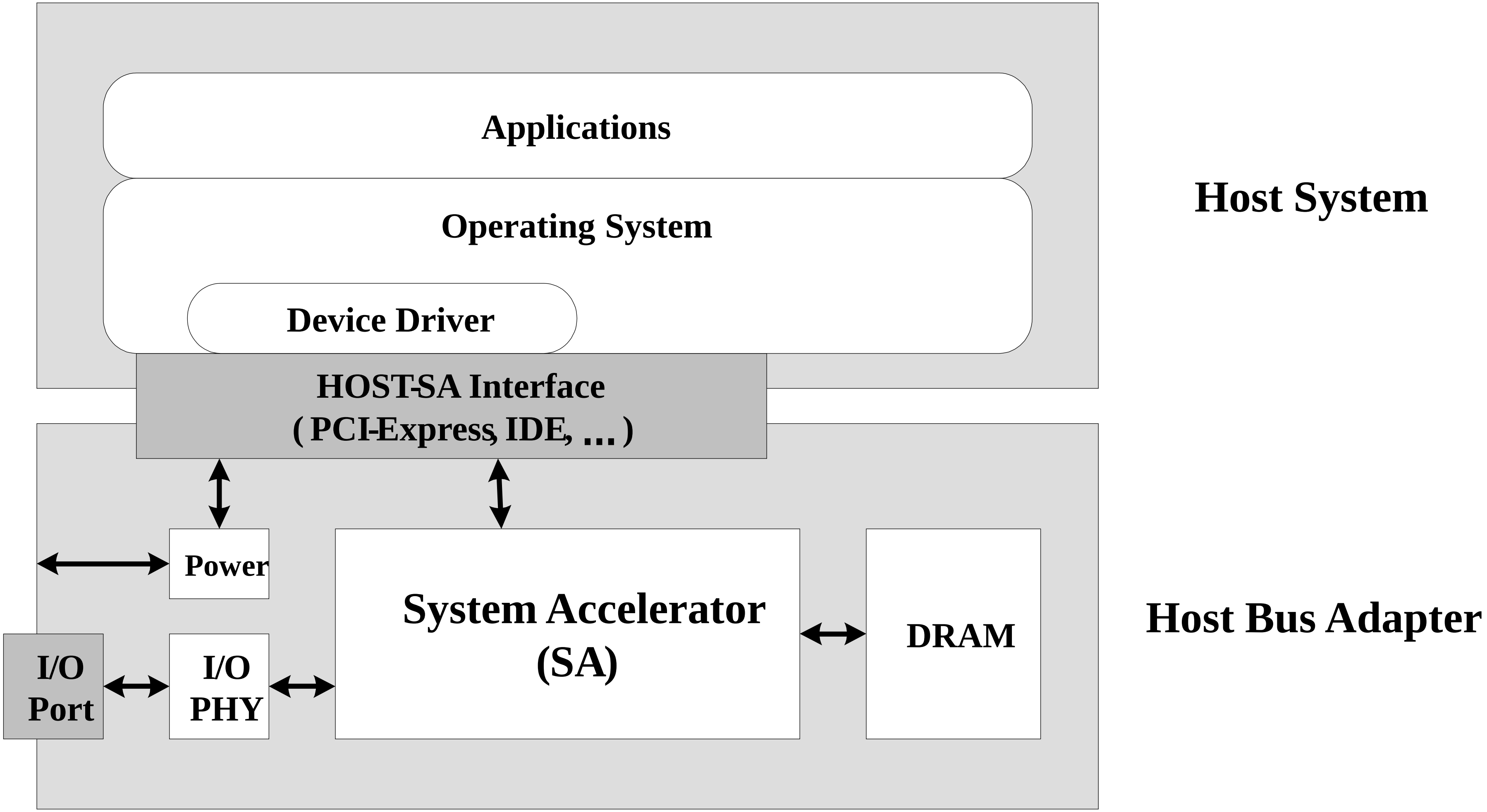
Why full system simulation?

- Higher abstraction level, higher productivity.
- Make verification and optimization of complex systems possible.



One Example

- TCP/IP offloads

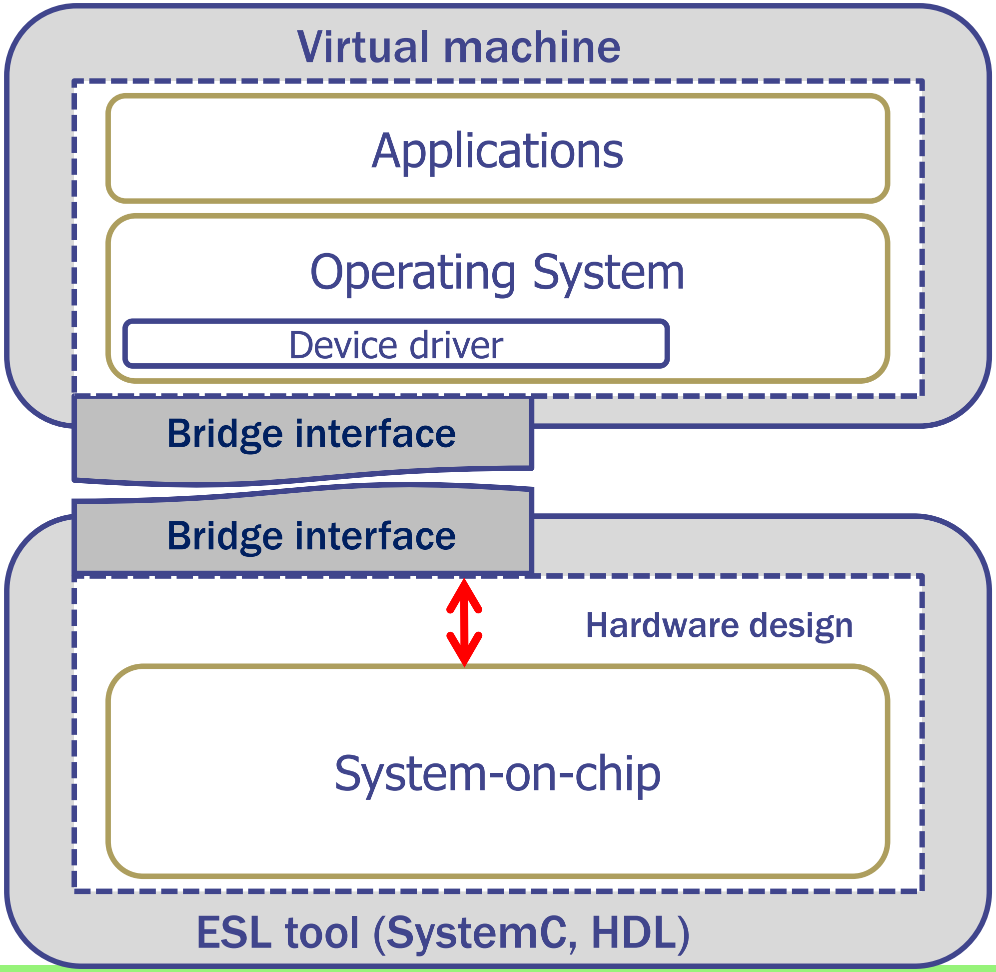


Limitation of Current ESL Simulation Tool

- ESL SystemC simulation tool
  - CoWare Platform Architect
- Advantages
  - Ready to use processor/bus models
  - Multiple level of abstractions
    - Transaction level
    - Register transfer level
  - Profiling tool
    - Bus utilization, reads/writes, etc.
- However,
  - Unacceptable OS booting time (half an hour)

Acceleration of OS Booting

- Take apart OS and CPU from ESL tool (CoWare)
- Use other tool to simulate CPU and to boot OS



What is a Virtual Machine

- Broad definition includes all emulation methods that provide a standard software interface, such as the Java VM

- "System Virtual Machines" provide a complete system level environment at binary ISA

- VM is an AP of the host OS

- Underlying HW platform is called the host, and its resources are shared among the guest VMs

Virtual Machine

- Virtual machine
  - VM-Ware
  - Virtual-PC
  - Parallel Desktop for Mac
  - QEMU (Quick Emulator)
- QEMU (http://bellard.org/qemu) (C/C++)
  - Open source code
  - Different ISAs support (x86,ARM,MIPS…etc)
  - Fast simulation speed (Functional level)
- QEMU-SystemC (Extension of QEMU)
  - Enable QEMU and SystemC modelling through AMBA interface in ARM versatile baseboard

QEMU Architecture

- QEMU is made of several subsystems
  - CPU emulator (e.g. x86, ARM, MIPS)
  - Emulator devices (e.g. VGA, IDE HD)
  - Generic devices (e.g. network devices)
    - Connecting QEMU emulated devices to the corresponding host devices.
  - Machine descriptions
    - Instantiating the emulated device.
  - Debugger
  - User interface

Add New Virtual Hardware

- QEMU allows us to write a virtual hardware and emulate it
- Steps
  - Design your virtual machine in C code
    - including initialization of the hardware , low level read/write (commands to hardware) functions for the hardware
  - Design device driver for that hardware

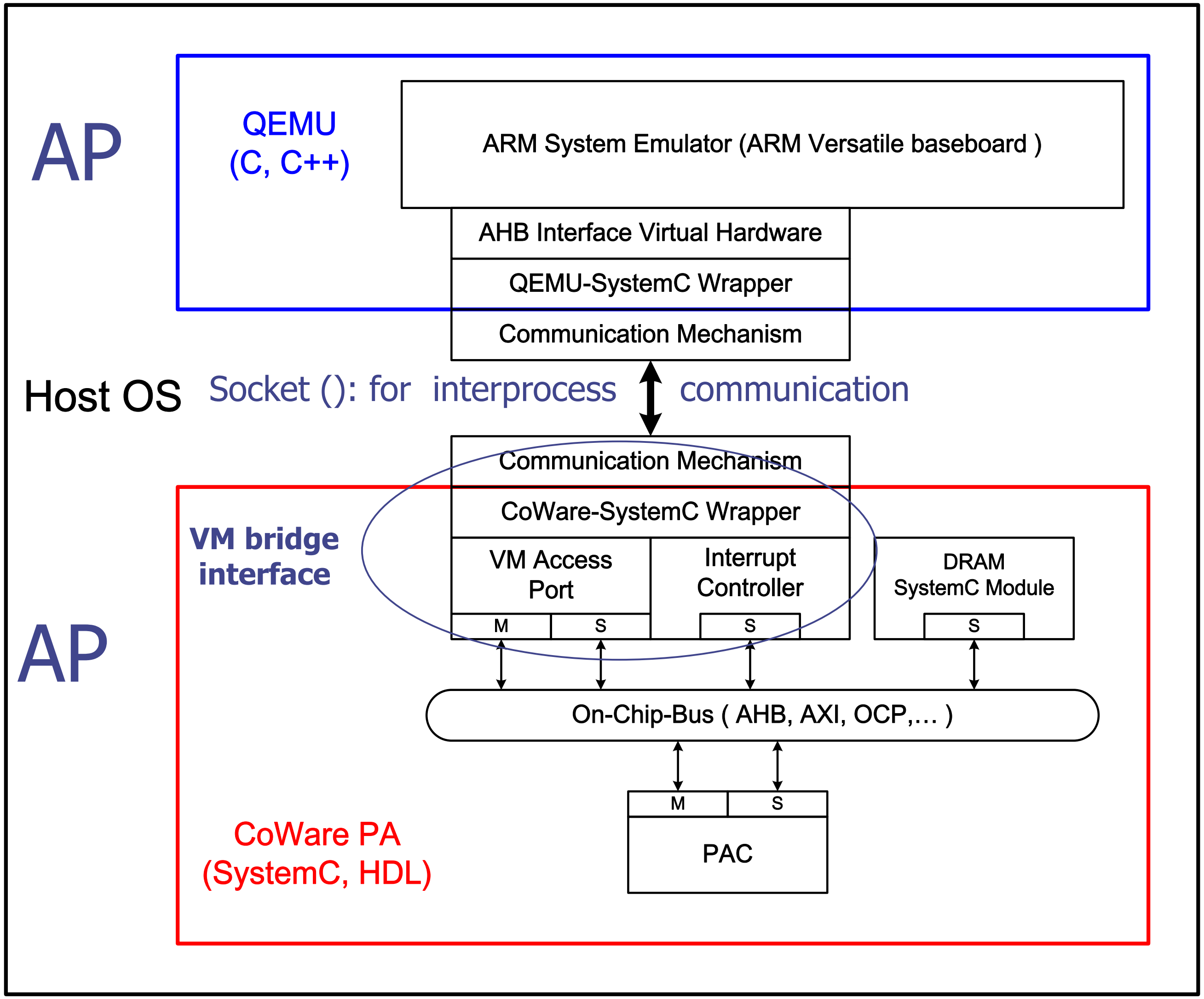
A Fast Hybrid Full System Simulation Platform

- QEMU
  - Boot and run OS with much less time (less 1 min)
  - Only functional simulation
- CoWare
  - SystemC based simulator & design environment in addition to C/C++, HDL
  - Detailed profiling
  - Booting Linux OS – long booting time
- Integration (QEMU & CoWare)
  - QEMU runs OS, upon which users develop AP
  - CoWare simulates hardware design
    - Accurate level (RTL)
    - Higher level

What is needed?

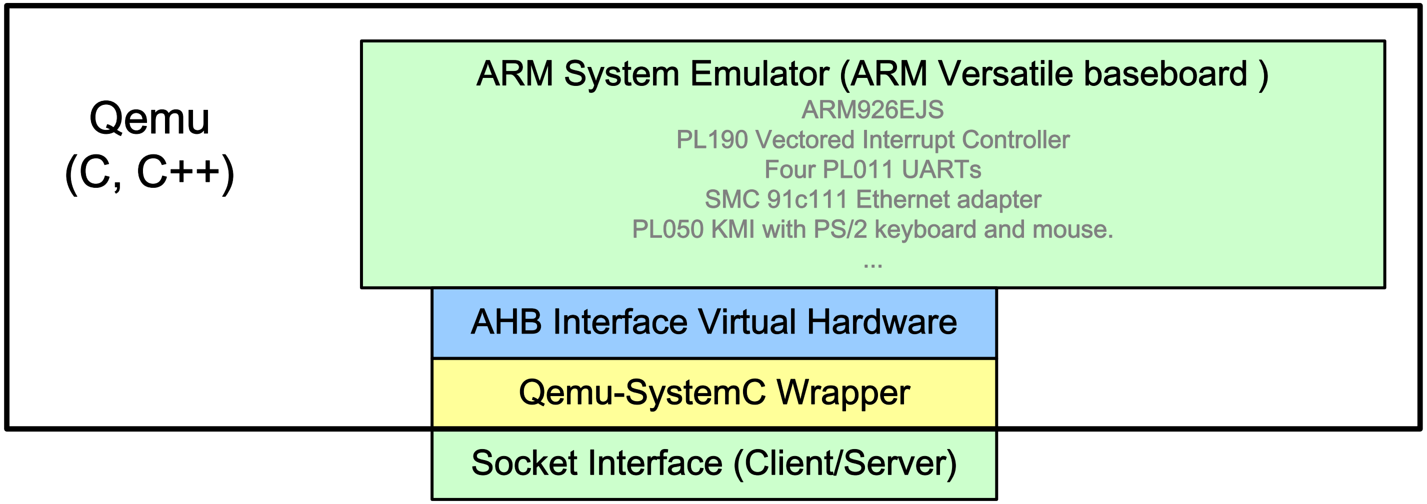
- Host Computer
  - Personal computer with Linux OS
- CoWare
  - Platform Architect v2007.1.2
- QEMU
  - QEMU-SystemC v0.91

Platform Overview

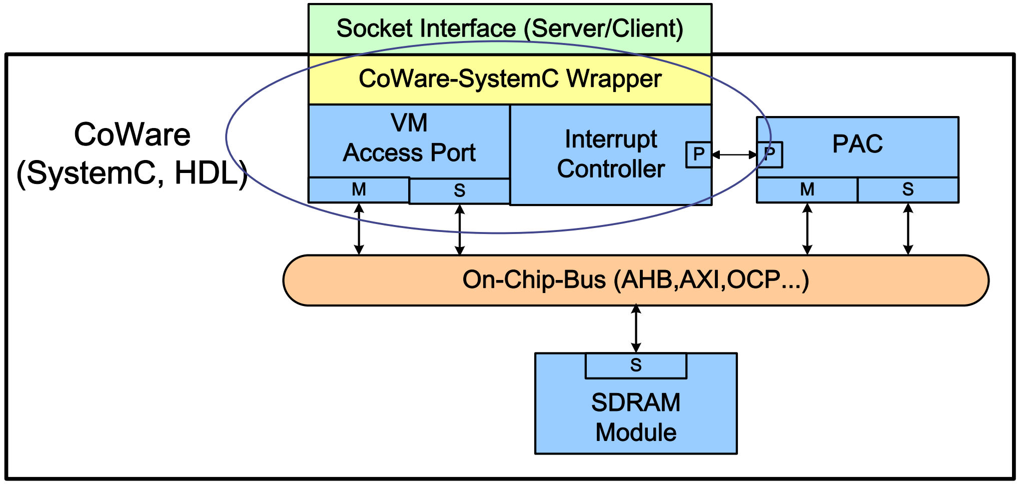


QEMU Side Details

- Simulated machine
  - ARM Versatile baseboard
  - Debian Linux 2.6.18
- Integration schemes for QEMU and CoWare
  - AHB interface virtual hardware
  - Character device driver (API) for design in CoWare
  - Interrupt service routine



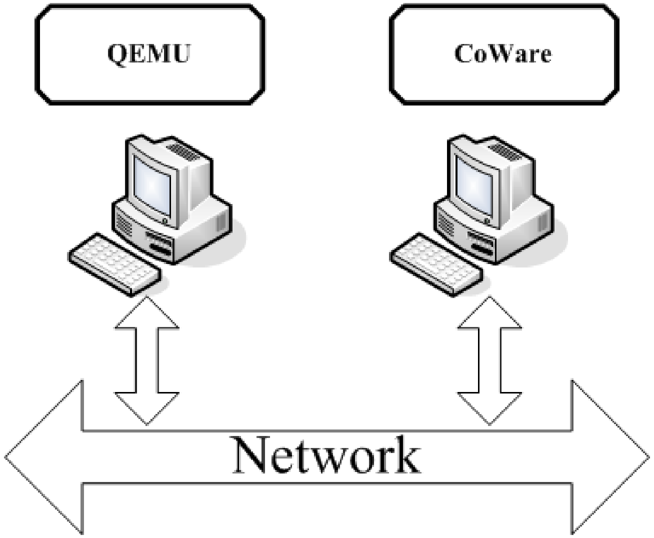
CoWare Side Details



- Hardware
  - AHB Bus
  - DSP/ASICs
  - Other devices
  - VM interface bridge
- VM interface bridge
  - VM access port
    - Read/write data from QEMU AP to slave modules in CoWare
  - Interrupt controller
    - Bypass interrupt signal to QEMU OS

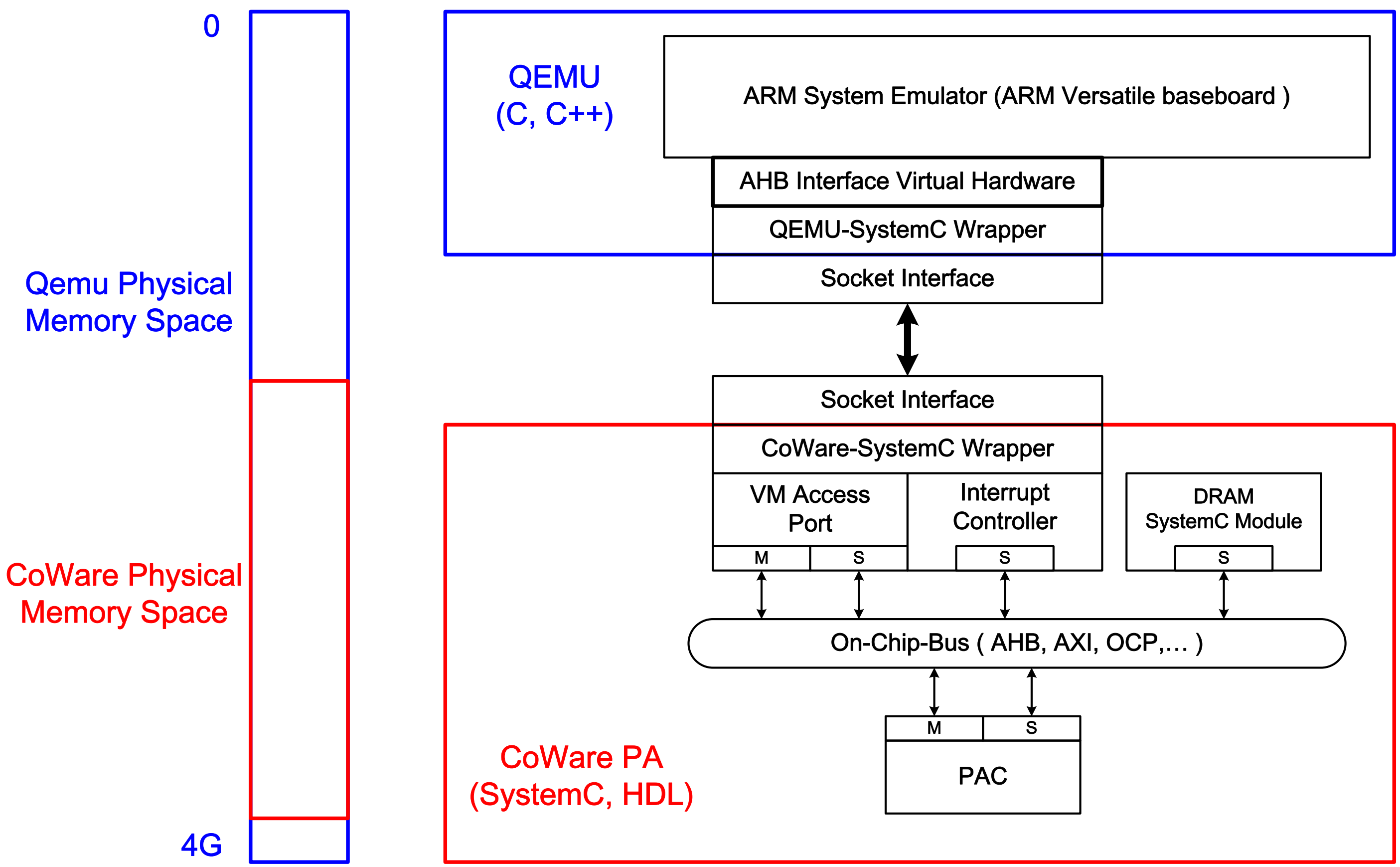
Communication Mechanism

- Socket call
  - Easy to use
  - Flexible
    - Other ESL simulation tool
  - Multiple computer support



System Memory Allocation

- Allocate physical memory space of CoWare hardware into memory space of QEMU virtual platform (simulated platform)



Examples of Application

- Heterogeneous Multi-Core
  - ARM + PAC (DSP)
- GPU (OpenGL/ES) + Multi-view generation
- Network SCTP/IP offload design

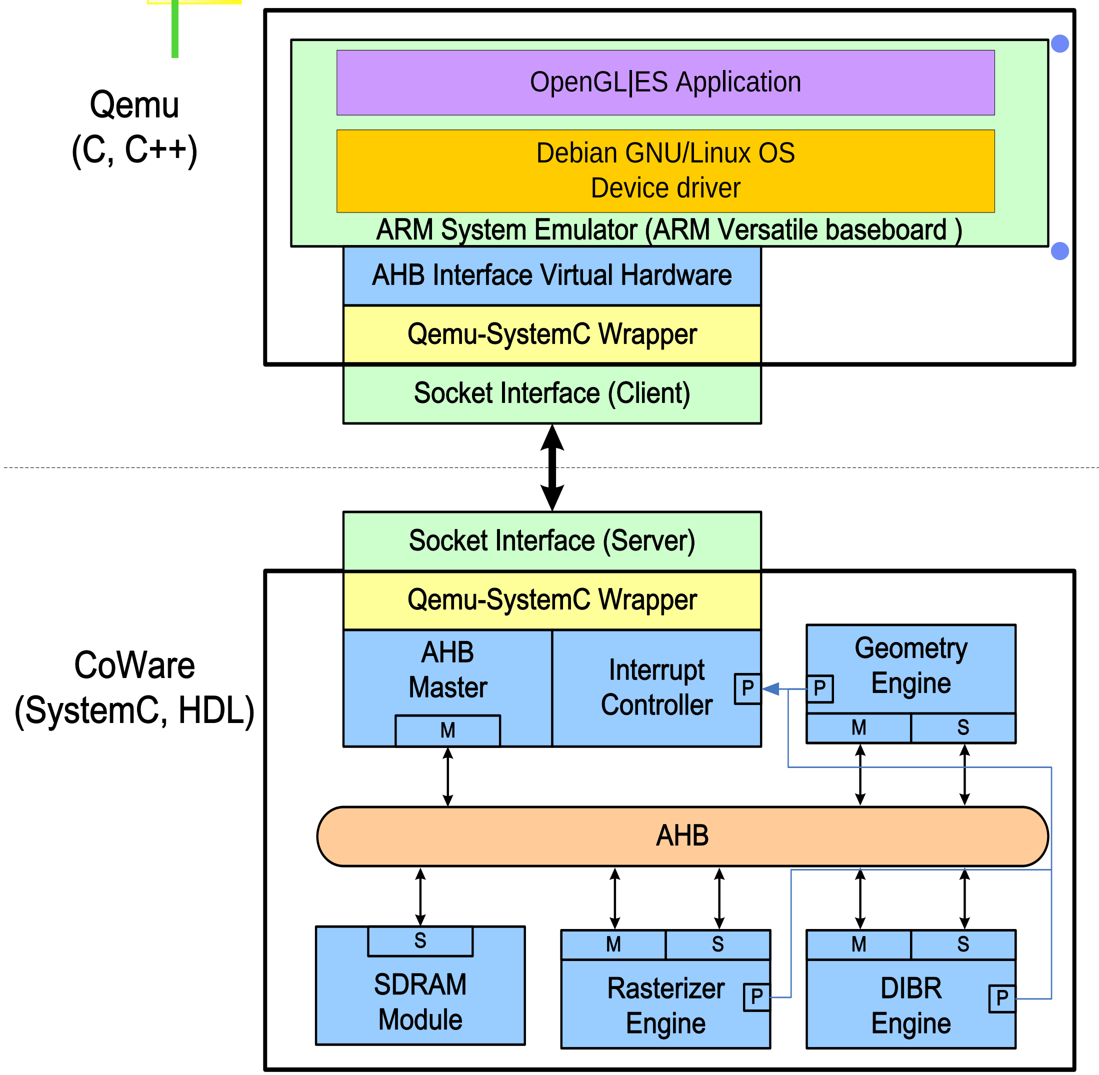
DSP Runs FFT Program

- Develop applications using driver API
- Use FFT program for example
  - Functions for designer
    - We should open the device first and close the device after using it.
      - IO_init() /*standard I/O initialization operation*/
      - IO_exit()
    - After opening the device , the FFT main program can use these functions to call APIs to read/write data from/to hardware in CoWare.
      - IO_read_byte , IO_read_half , IO_read_word
      - IO_write_byte, IO_write_half, IO_write_word

Heterogeneous Multi-Core

- FFT main program runs in QEMU OS
  - First open device using IO_init()
  - Send PAC binary and data(fft.img) to CoWare
    - IO_write_word(0xa0000000, send_data)
  - Call function fft()
    - use IO_write_word to set PAC to run fft
    - use IO_read_word to read data calculated by PAC
  - Close the device, use IO_exit()
  - Check FFT results

FULL SYSTEM VERIFICATION PLATFORM FOR MULTI-VIEW GPU

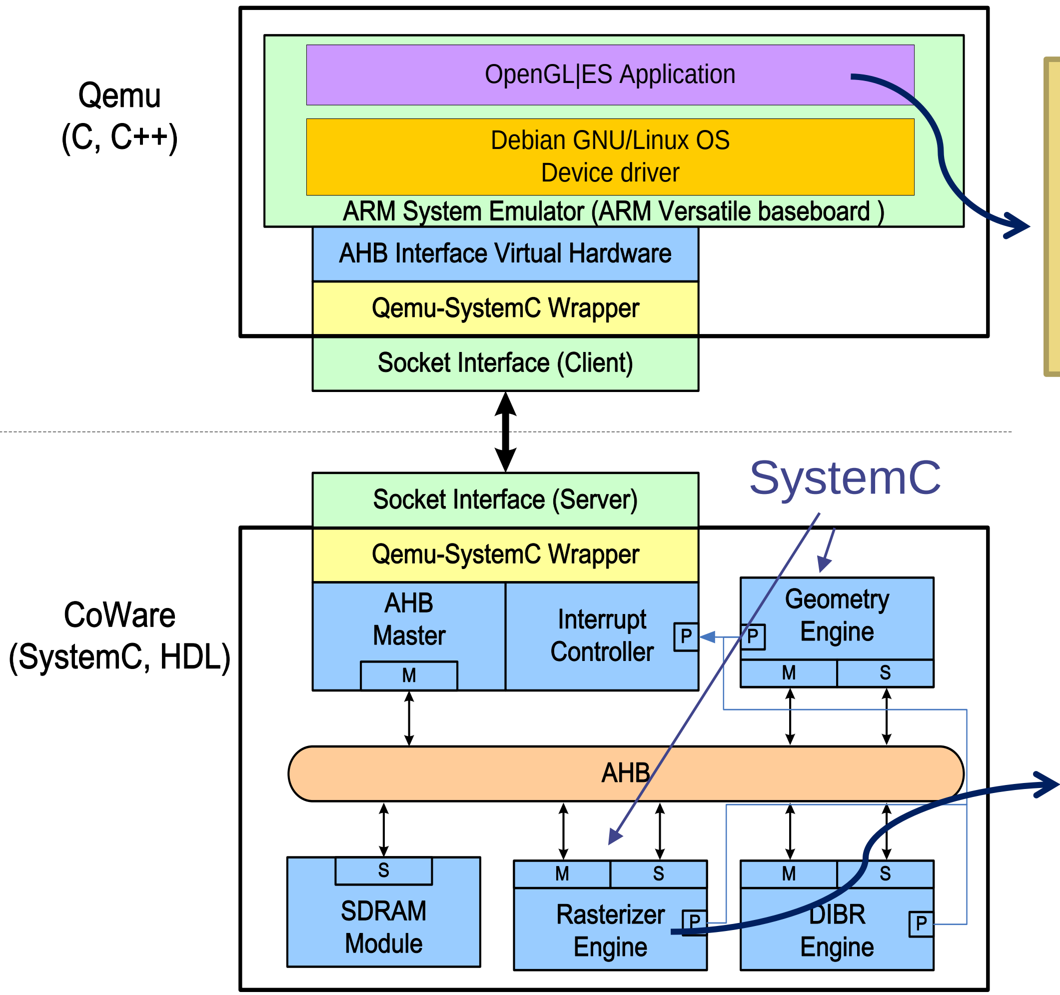


QEMU
- OpenGL ES Application
- Customized device driver

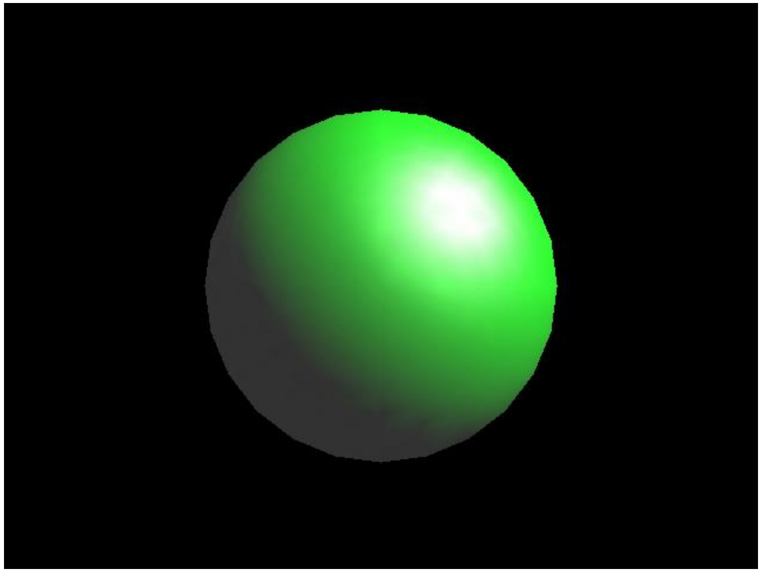
SystemC/RTL Co-Simulation
- GPU core
  - Geometry module
  - Rasterization module
- Multi-View generation
  - **D**epth-**I**mage **B**ased **R**endering

GPU in System C

- GPU with SystemC encapsulation

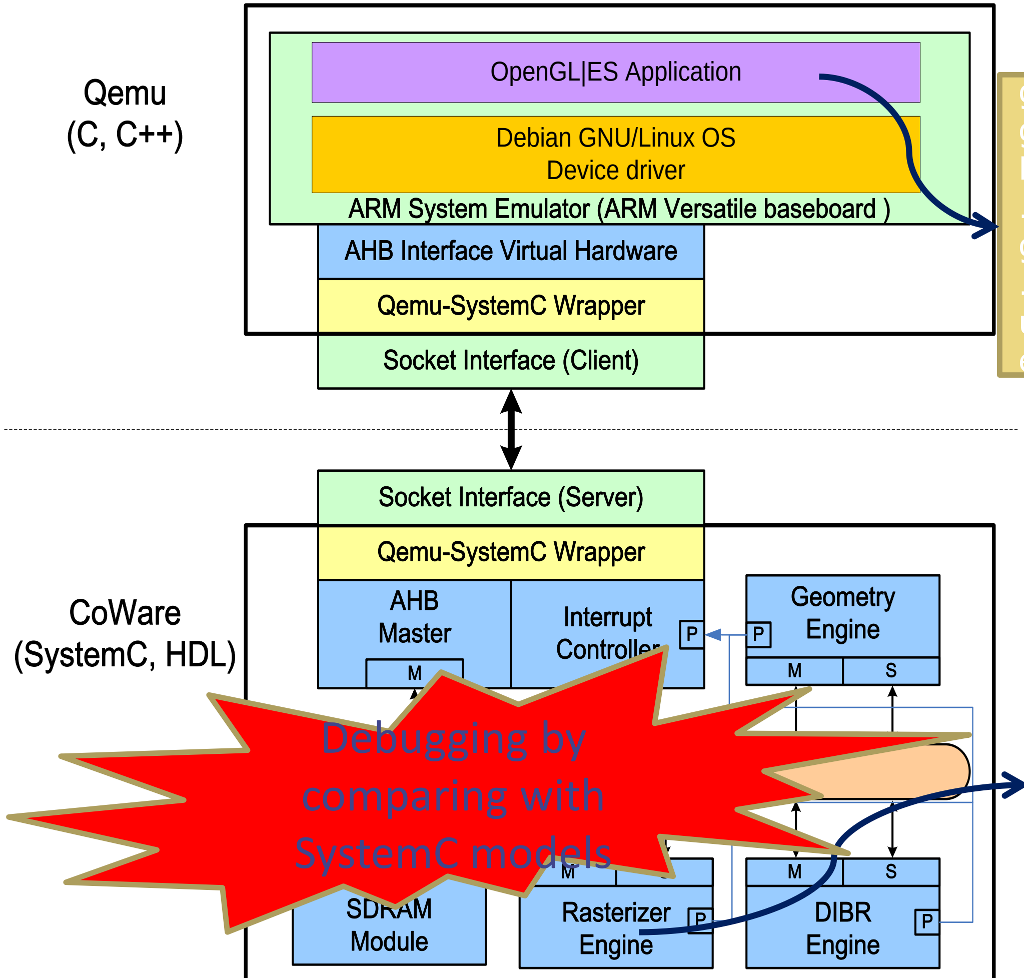



```
glFrustumf(-1.0, 1.0, -1.0, 1.0, 1.0, 20.0);
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
…
glTranslatef(0.5, 0.0, -2.0);
…
ugSolidSpheref(1.0f, 24, 24);
eglSwapBuffers(eglDisplay,eglSurface);
```

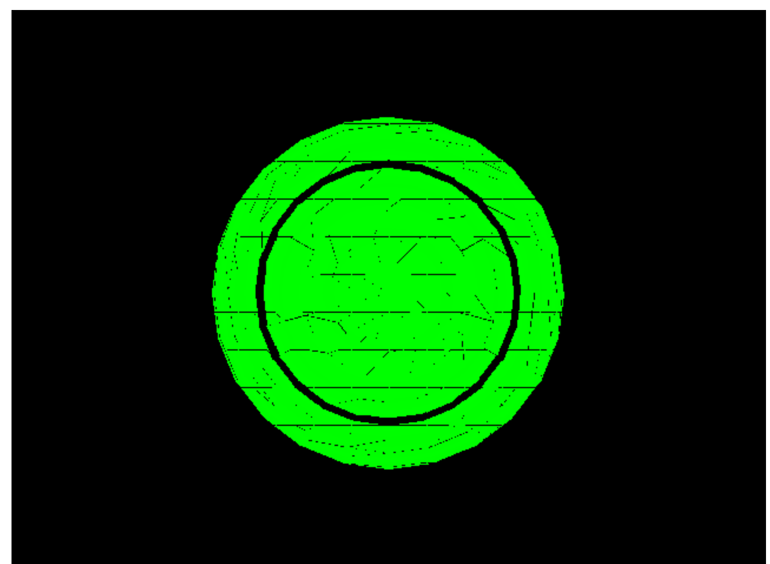


GPU in fresh RTL modules

- GPU with RTL encapsulation

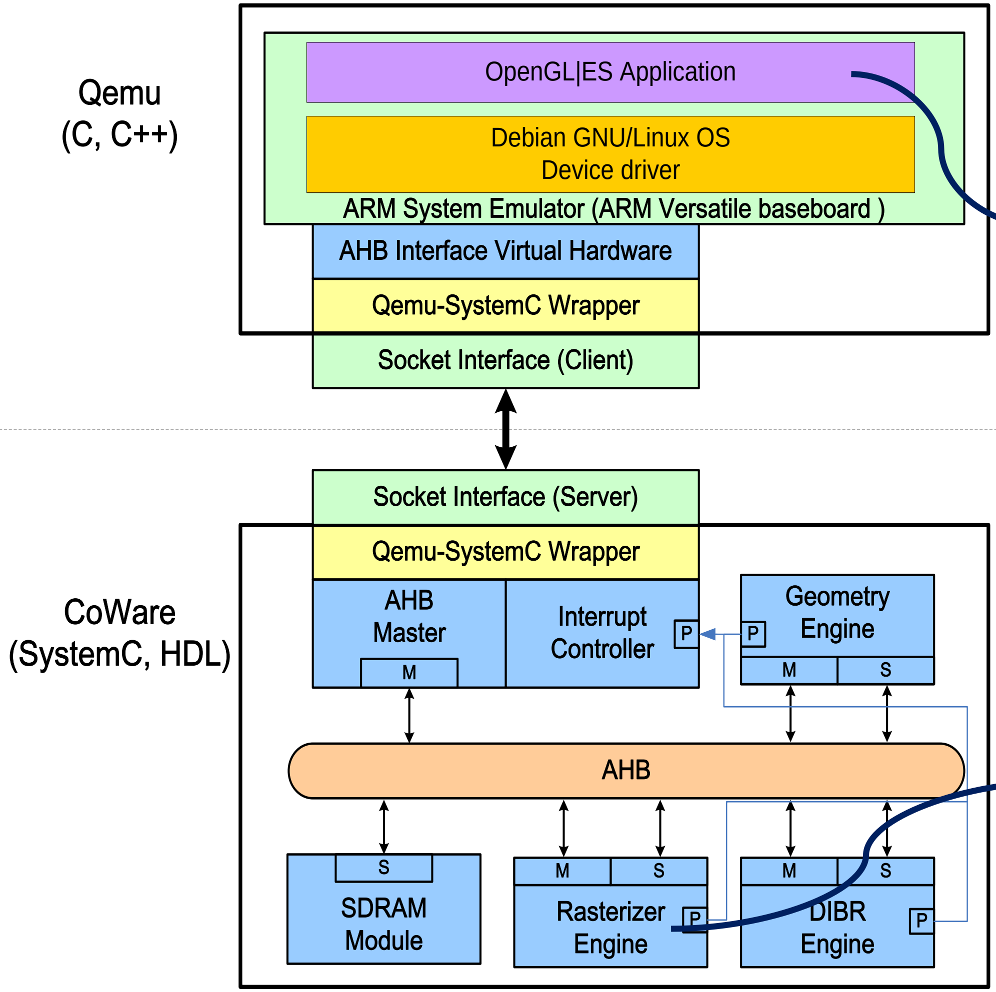



```
glFrustumf(-1.0, 1.0, -1.0, 1.0, 1.0, 20.0);
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
...
glTranslatef(0.5, 0.0, -2.0);
...
ugSolidSpheref(1.0f, 24, 24);
eglSwapBuffers(eglDisplay,eglSurface);
```

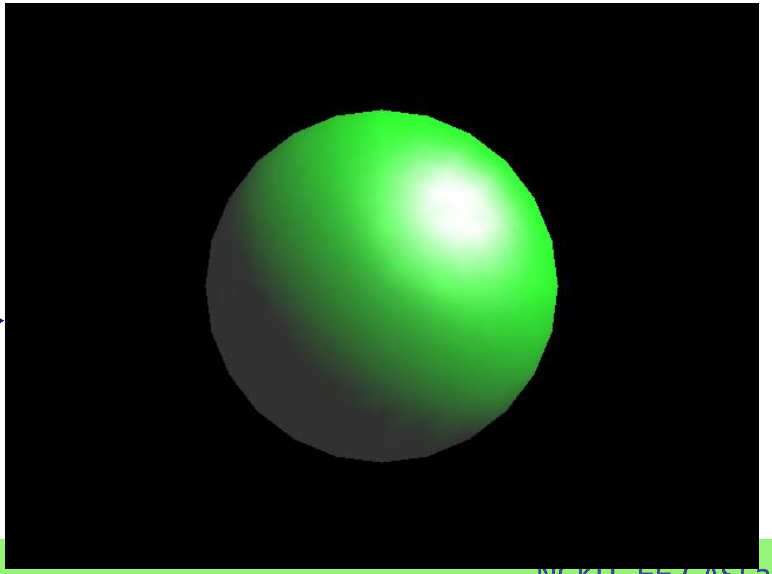


100 % FULL SYSTEM VERIFICATION

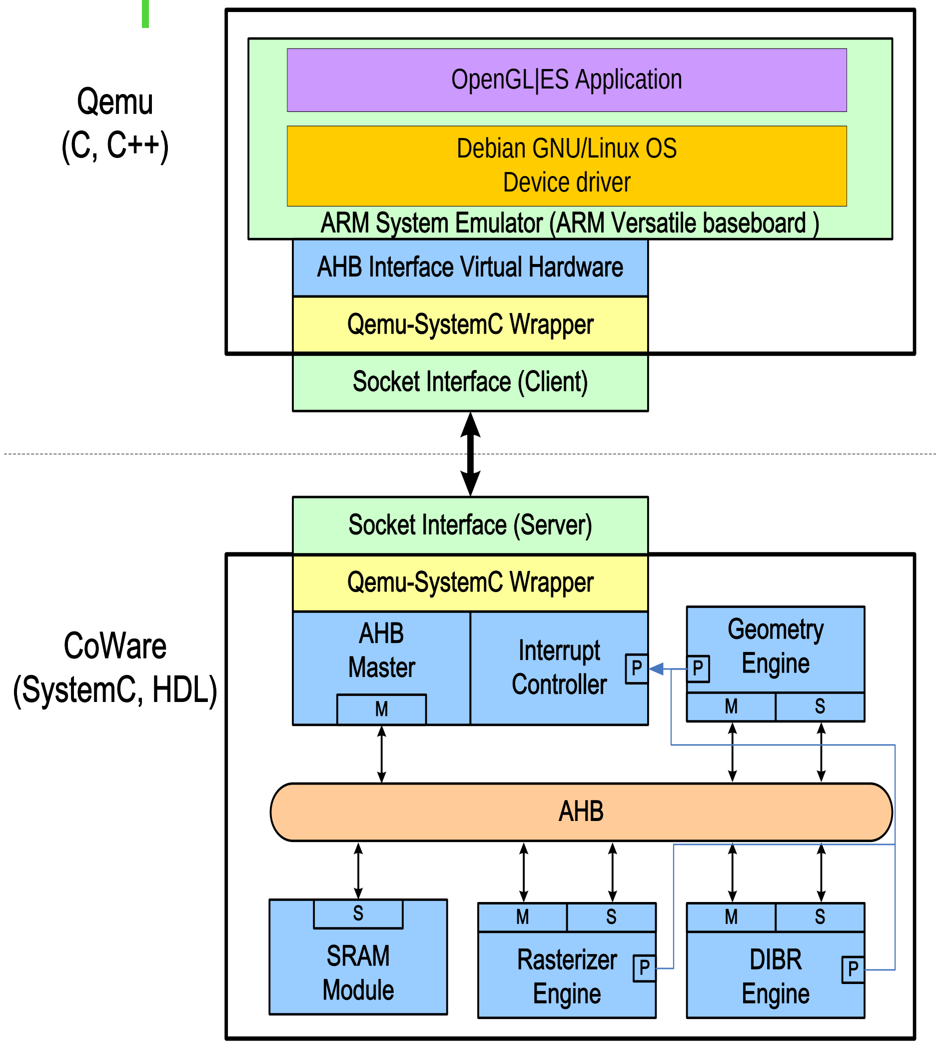
- GPU with RTL encapsulation
- RTL verification confirmed




```
glFrustumf(-1.0, 1.0, -1.0, 1.0, 1.0, 20.0);
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_
BIT);
…
glTranslatef(0.5, 0.0, -2.0);
…
ugSolidSpheref(1.0f, 24, 24);
eglSwapBuffers(eglDisplay,eglSurface);
```



Flexibility

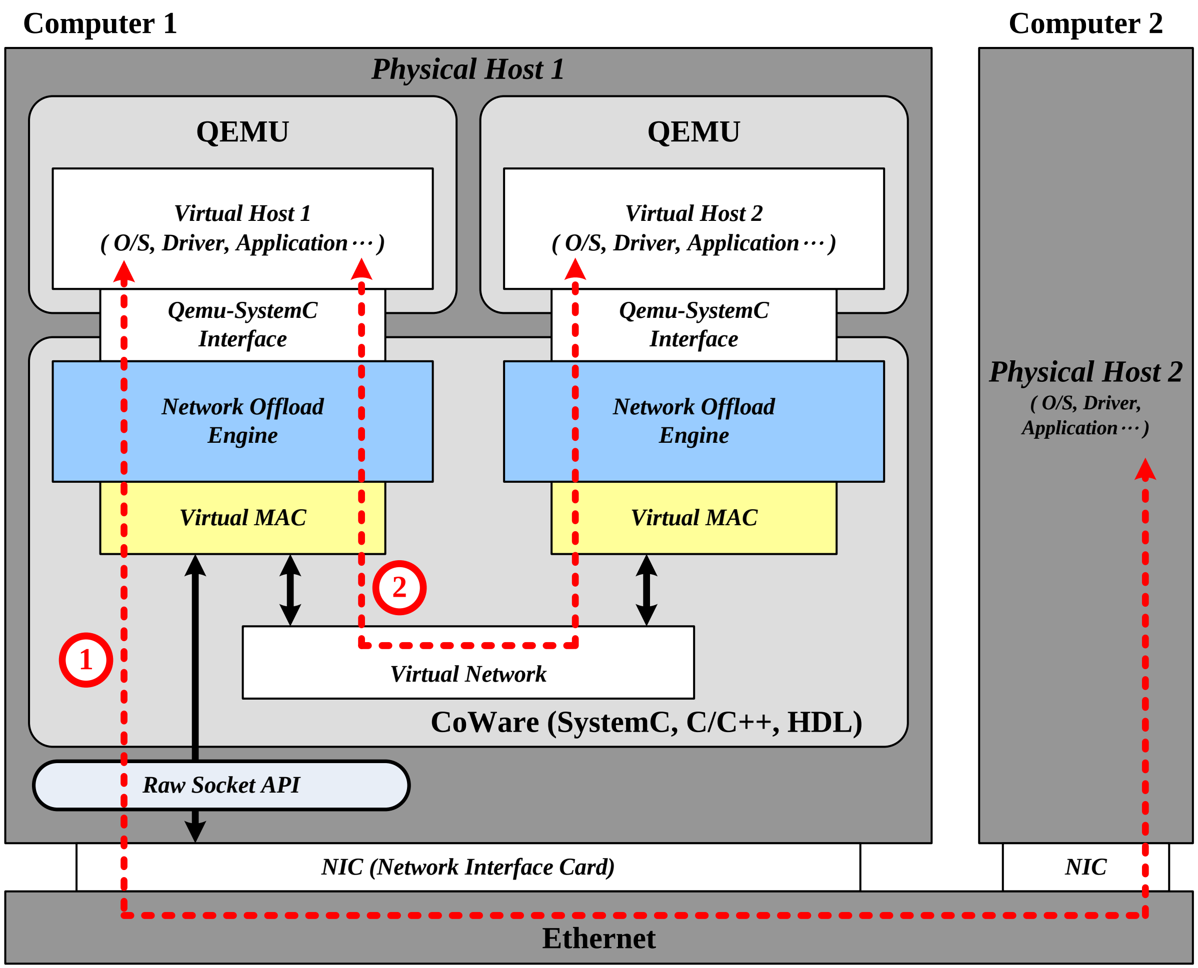


- QEMU (fast emulator)
  - OpenGL ES benchmark suite
  - Customized device driver
    - For GPU + DIBR
- Co-simulation

Module name	Design level
AMBA AHB	Timed TLM
AMBA bridge	Timed TLM
SRAM	Untimed TLM
Geometry Engine	RTL
Rasterizer Engine	RTL
DIBR Engine	RTL

SCTP/IP Offload System

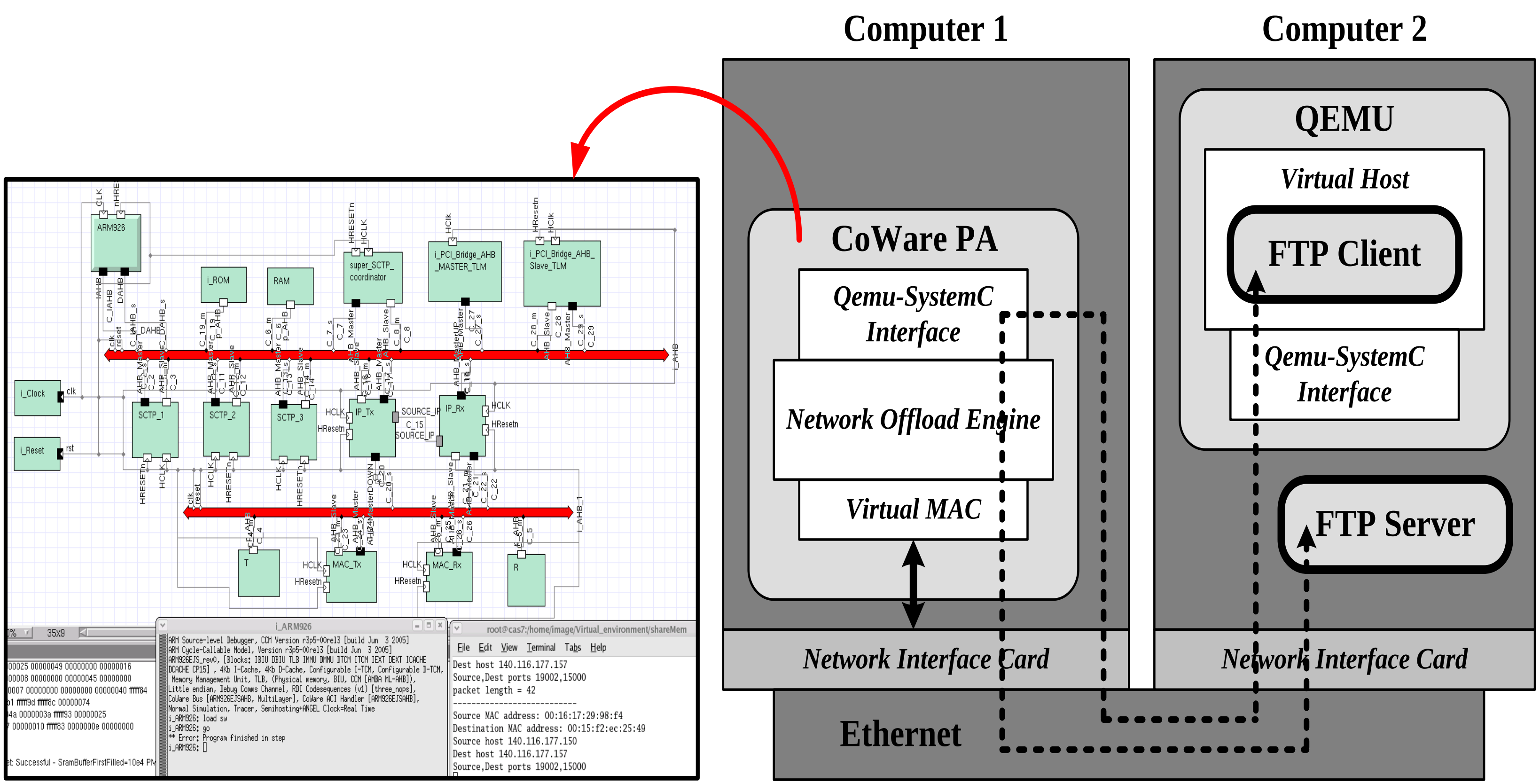
SCTP: Stream Control Transmission Protocol



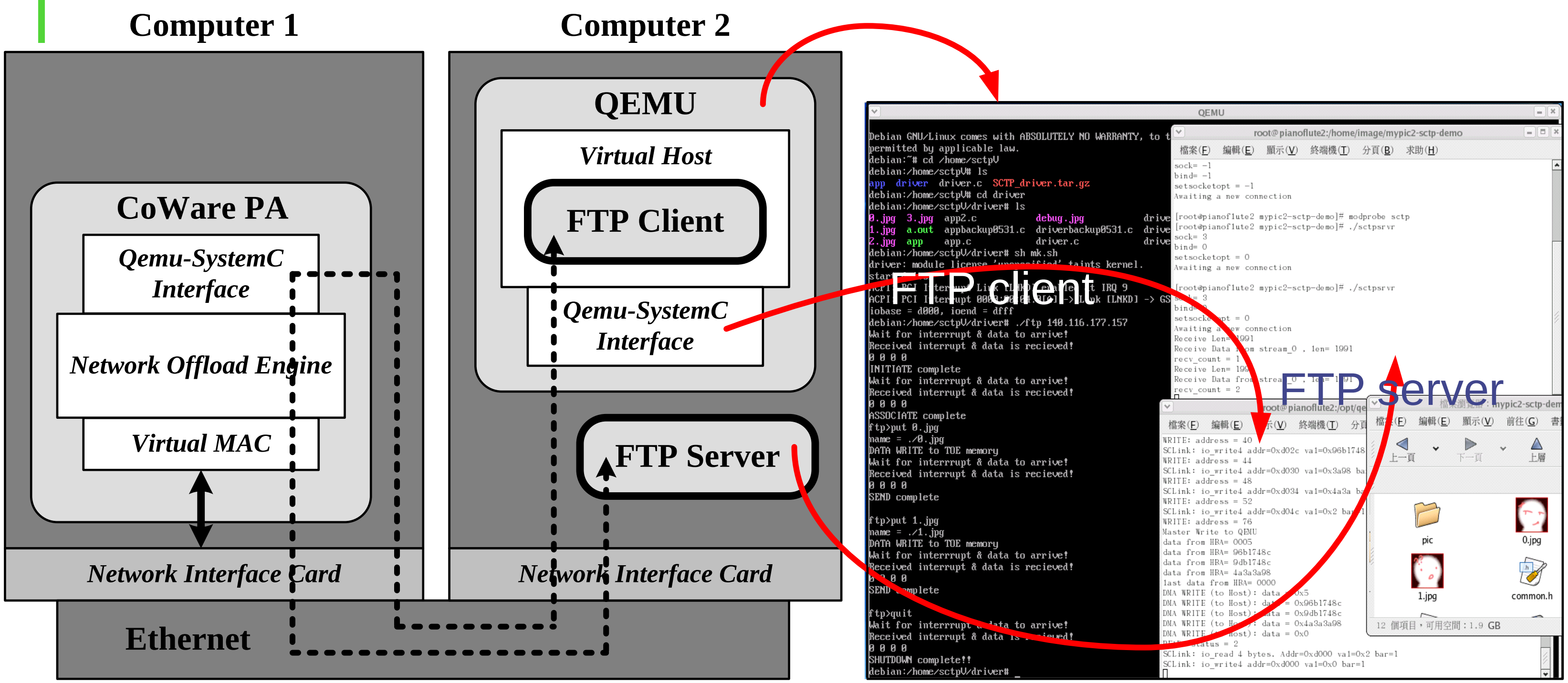
1. Functional verification
2. Connection with real world (path1)
3. Performance evaluation for 10 Gb (path 2)

SCTP/IP Offload System

- CoWare on PC 1, Host QEMU on PC 2
  - Network Offload Engine (SCTP, IP, MAC)
  - FTP client (run on your design) talks to FTP server (real world)
  - Virtual MAC (model bit rates)



Network Offload System



- The FTP client in the virtual platform was uploading files to the server.
- The FTP server in the real world computer was receiving data from the client.
- Finally, the files had been received completely at the server.

Portability

- The same memory allocation and OS
  - No need to change device driver and application
- Different OS
  - Only need to change device driver
    - Header files, different system calls
  - No need to change application
- Different memory allocation
  - Need to change device driver and application but only address dependent statements

Performance Issue

- Simulation overhead
  - Use socket call for communication between QEMU and CoWare
  - Hardware implementation (FPGA) uses no socket call
- Performance improvement
  - Reduce communication
    - Rbyte+Rbyte+Rbyte+Rbyte => Rword
    - Reconstruct Data flow

And in conclusion……

- A full system simulation platform that enables Application, Linux operating system, Host processor, and RTL/SystemC design simulation.
- A convenient and easy-to-use integrated platform for software/hardware debugging and verification.
  - Applications, drivers, RTLs.
- An ESL tool that can tackle with designs of high complexity.
- Instruction profiling in QEMU
  - Instruction count (PID-based), type, user/kernel mode
- Power estimation