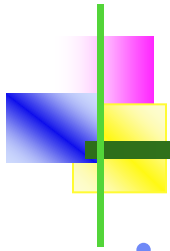# ESL-Based Full System Simulation Platform

陳中和

Department of Electrical Engineering

Institute of Computer and Communication Engineering

National Cheng Kung University

NCKU-CASLab

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**

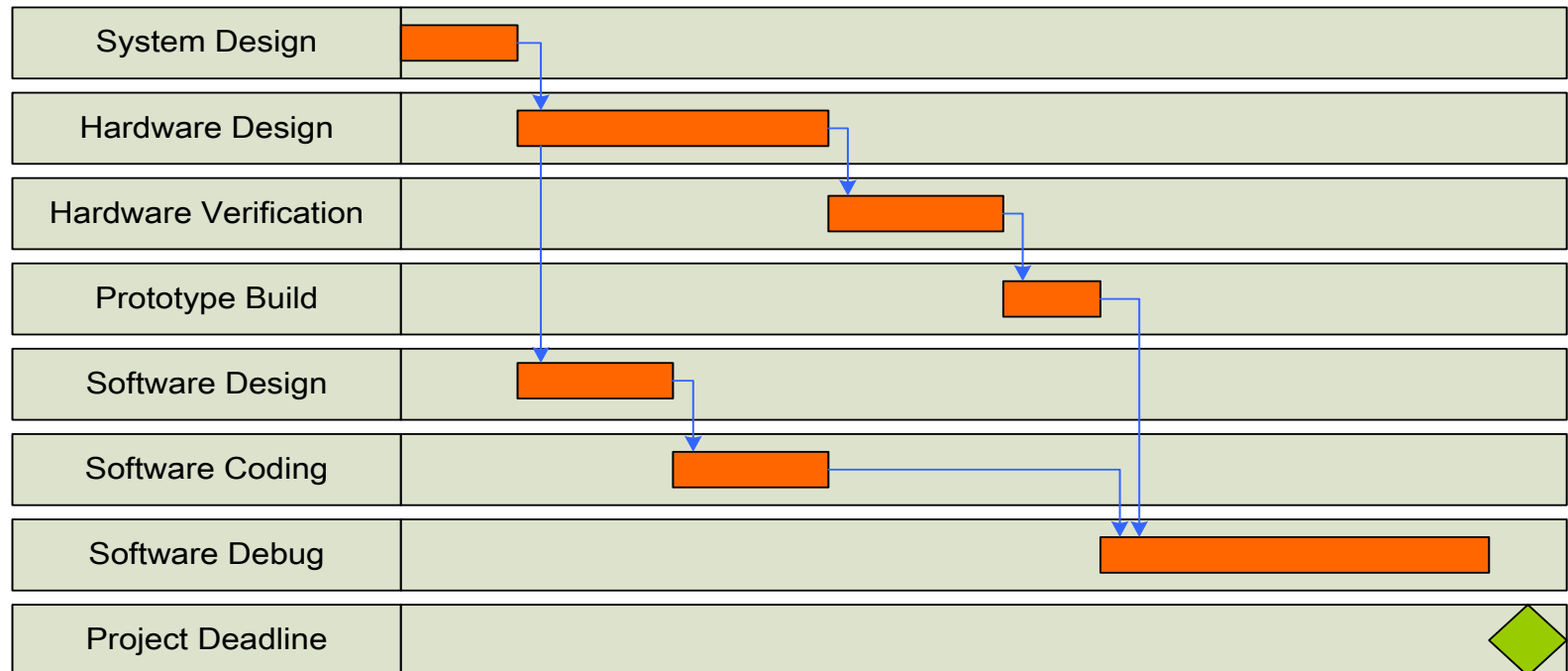- **LAB5: Full System Simulation using QEMU & PlatformArchitect**
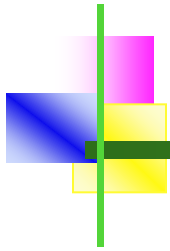
Proposal

- Due in three weeks.
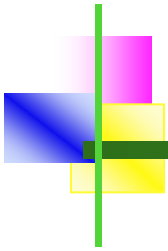- Proposal report due (11/28)
  - and presentation on 11/29 (5-mins)
- Final report and presentation

Electronic System Level Design

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

System Design	
Hardware Design	
Hardware Verification	
Prototype Build	
Software Design	
Software Coding	
Software Debug	
Project Deadline	

ESL

- Early interaction with software

System Design	
Hardware Design	
Hardware Verification	
Prototype Build	
Software Design	
Software Coding	
Software Debug	
Project Deadline	
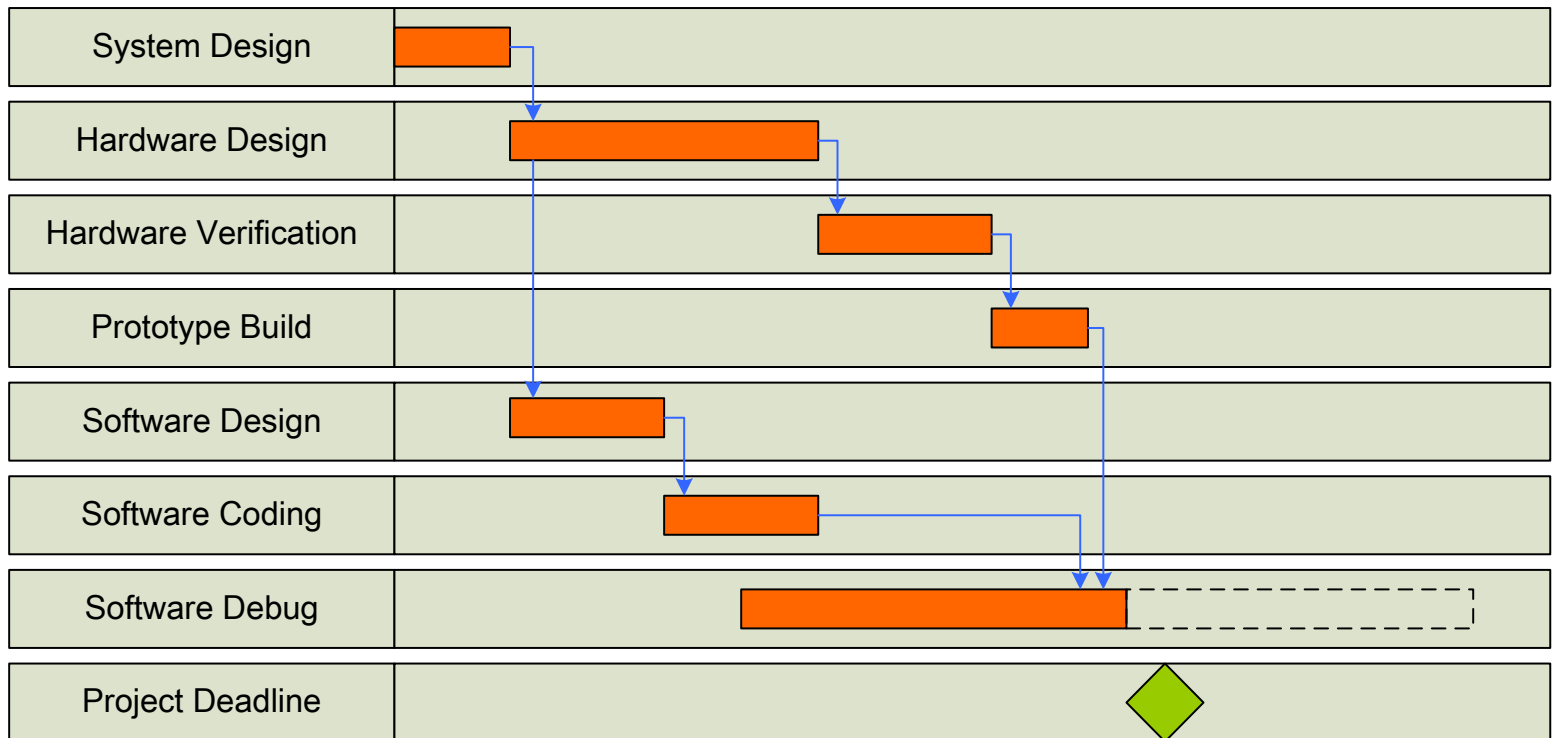
What is Full System Simulation

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

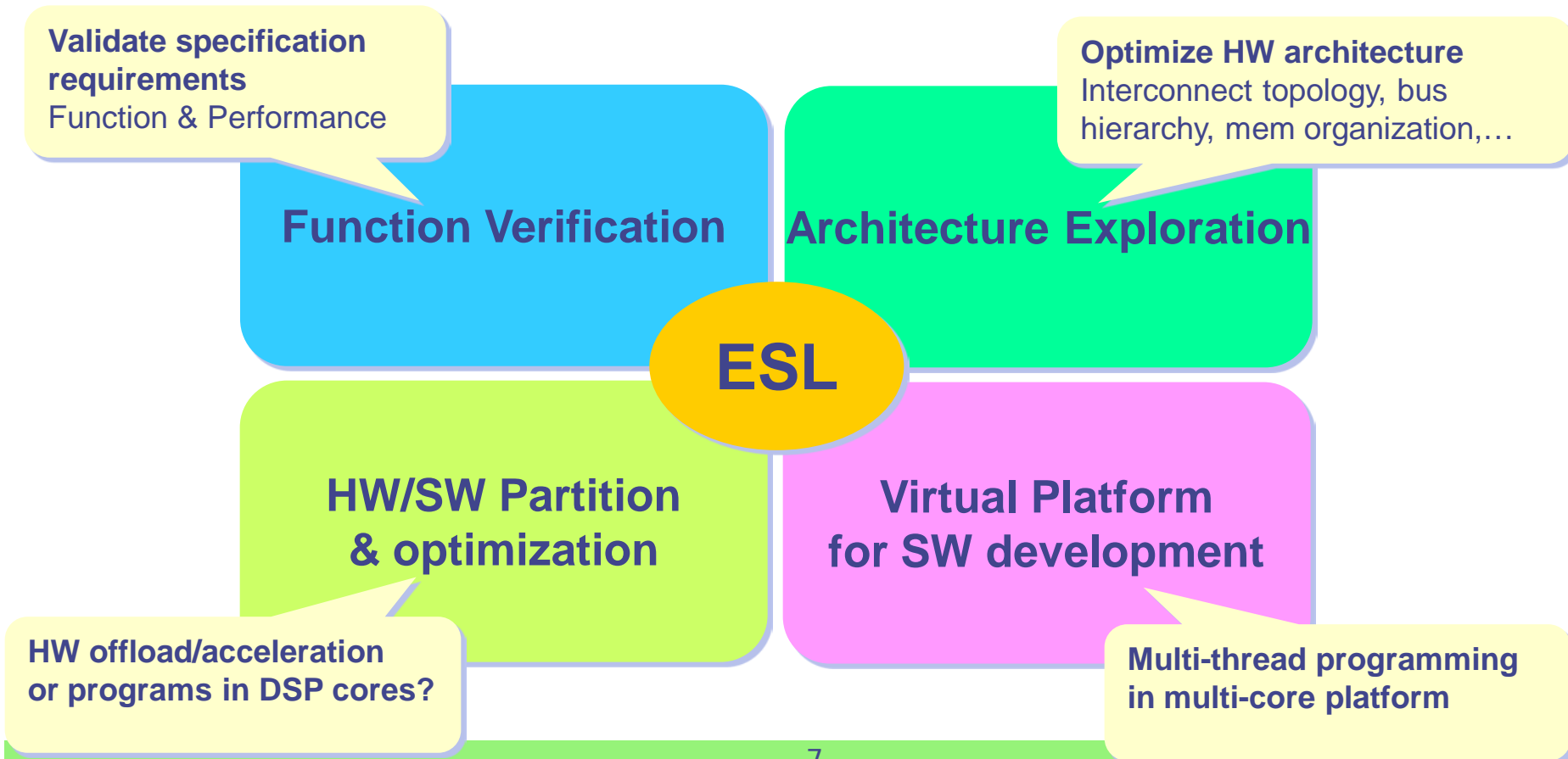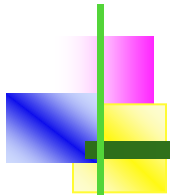 - Software : operating system, device drivers, and applications

Why full system simulation?

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

Validate specification requirements
Function & Performance

Optimize HW architecture
Interconnect topology, bus hierarchy, mem organization,…

Function Verification

Architecture Exploration

ESL

HW/SW Partition & optimization

Virtual Platform for SW development

HW offload/acceleration or programs in DSP cores?

Multi-thread programming in multi-core platform

One Example

- TCP/IP offloads

Applications

Operating System

Device Driver

HOST-SA Interface
(PCI-Express, IDE, …)

Host System

Power

System Accelerator
(SA)

DRAM

I/O Port

I/O PHY

Host Bus Adapter
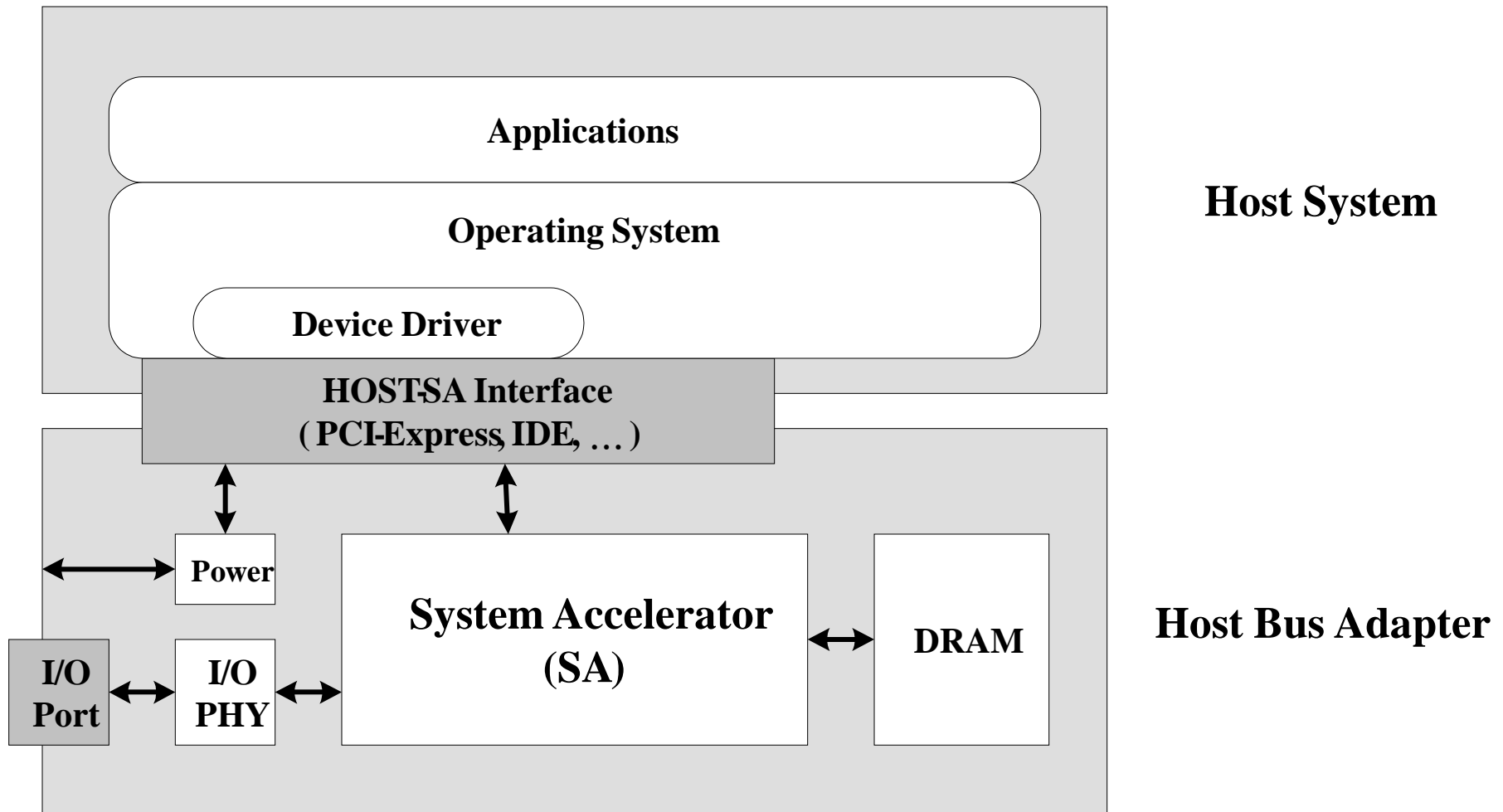
Limitation of Current ESL Simulation Tool

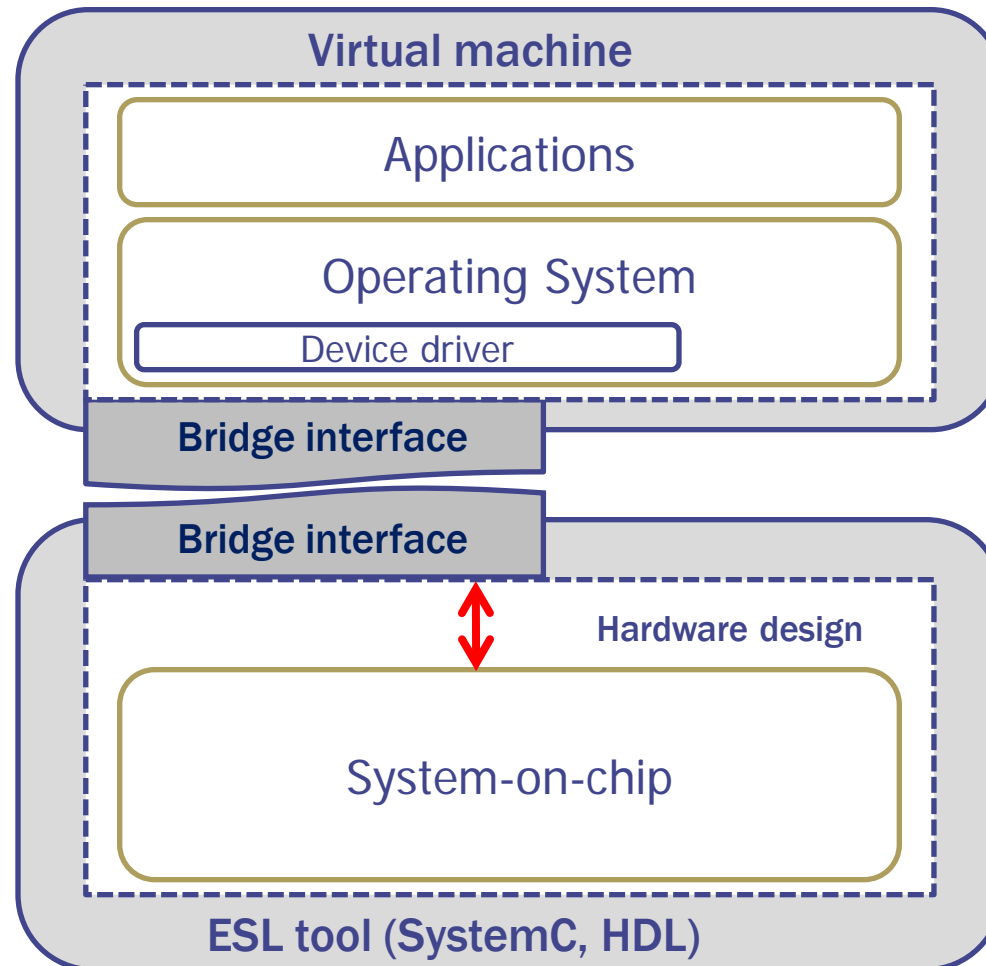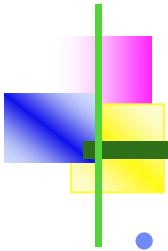- ESL SystemC simulation tool
  - CoWare Platform Architect
- Advantages
  - Ready to use processor/bus models
  - Multiple level of abstractions
    - Transaction level
    - Register transfer level
  - Profiling tool
    - Bus utilization, reads/writes, etc.
- However,
  - Unacceptable OS booting time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

Virtual machine

Applications

Operating System

Device driver

Bridge interface

Bridge interface

Hardware design

System-on-chip

ESL tool (SystemC, HDL)

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

AP

QEMU (C, C++)

ARM System Emulator (ARM Versatile baseboard)

AHB Interface Virtual Hardware

QEMU-SystemC Wrapper

Communication Mechanism

Host OS   Socket (): for interprocess   communication

Communication Mechanism

CoWare-SystemC Wrapper

VM bridge interface

VM Access Port

Interrupt Controller

DRAM SystemC Module

M   S   S   S

AP

On-Chip-Bus (AHB, AXI, OCP,…)

M   S

PAC

CoWare PA (SystemC, HDL)

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

Qemu
(C, C++)

ARM System Emulator (ARM Versatile baseboard)
ARM926EJS
PL190 Vectored Interrupt Controller
Four PL011 UARTs
SMC 91c111 Ethernet adapter
PL050 KMI with PS/2 keyboard and mouse.
...

AHB Interface Virtual Hardware

Qemu-SystemC Wrapper

Socket Interface (Client/Server)
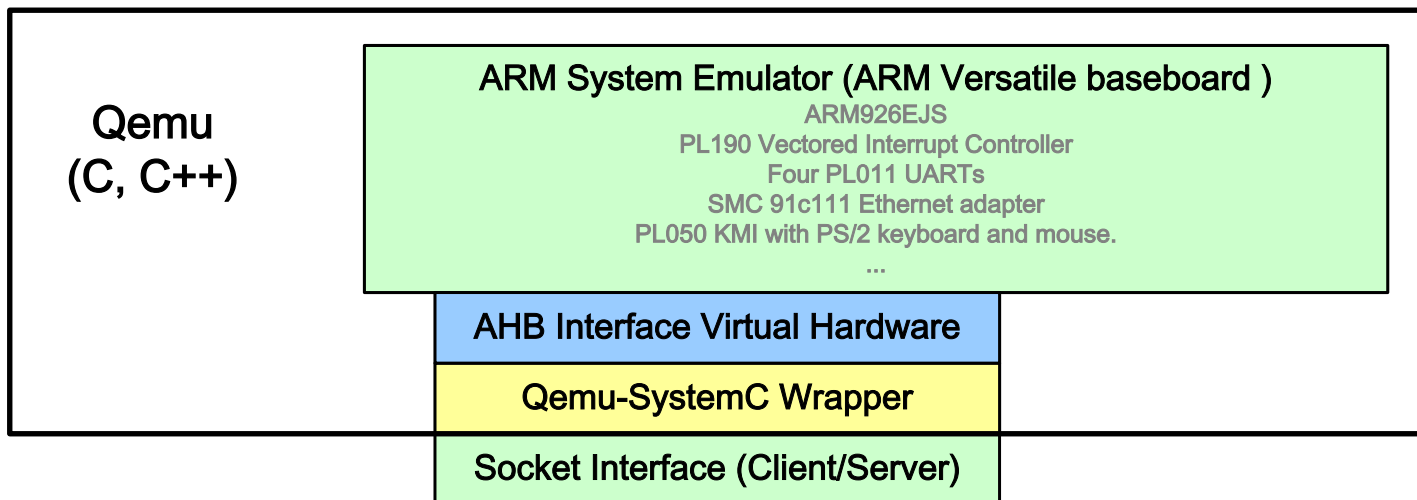
CoWare Side Details

- ## 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

Socket Interface (Server/Client)

CoWare-SystemC Wrapper

CoWare (SystemC, HDL)

VM Access Port — M | S

Interrupt Controller — P | P — PAC — M | S

On-Chip-Bus (AHB,AXI,OCP...)
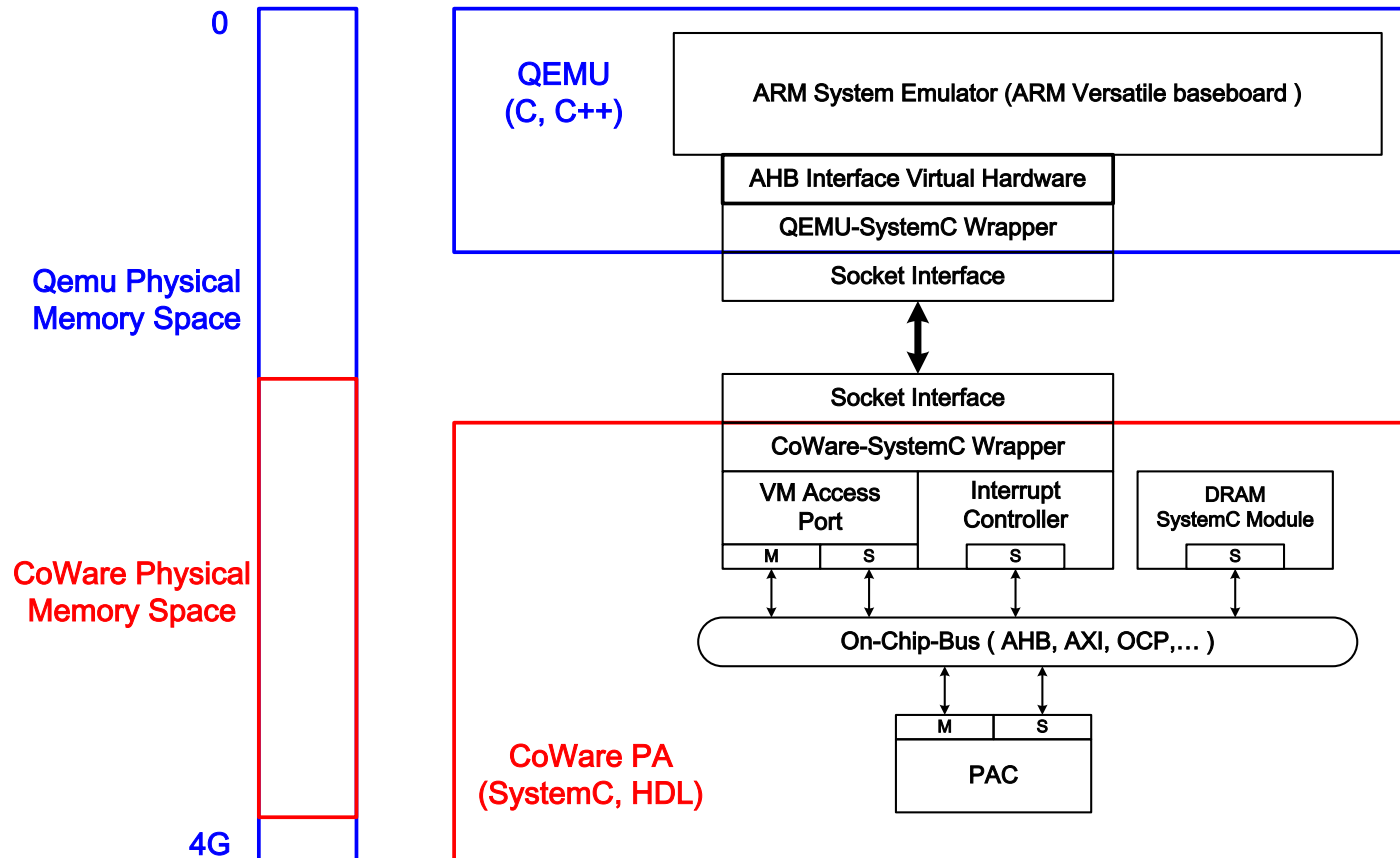
S — SDRAM Module
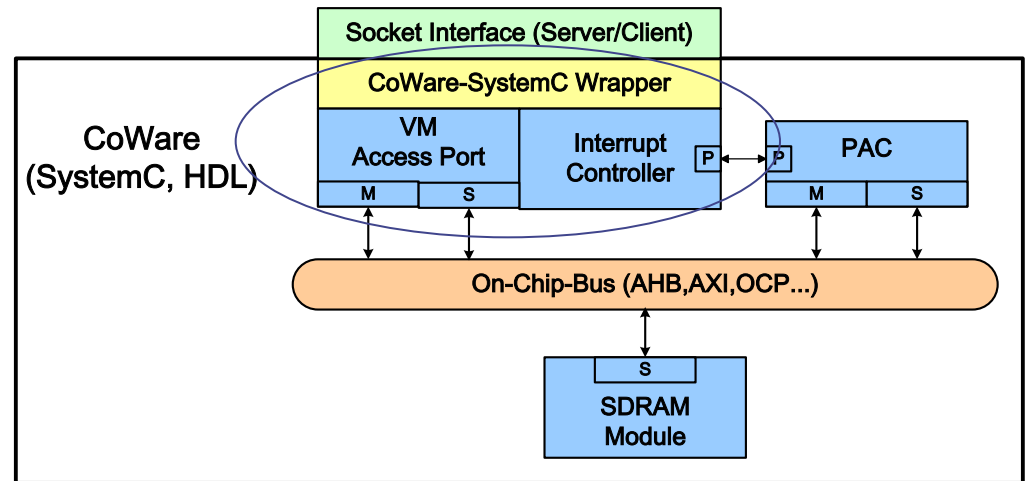
Communication Mechanism

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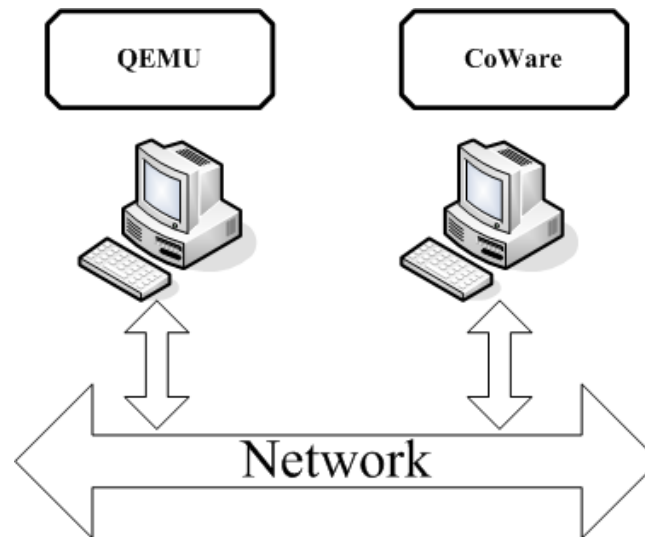
System Memory Allocation

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

0

Qemu Physical Memory Space

CoWare Physical Memory Space

4G

QEMU (C, C++)

| ARM System Emulator (ARM Versatile baseboard) |
| AHB Interface Virtual Hardware |
| QEMU-SystemC Wrapper |
| Socket Interface |

| Socket Interface |
| CoWare-SystemC Wrapper |

| VM Access Port | Interrupt Controller | DRAM SystemC Module |
| M | S | S | S |

On-Chip-Bus (AHB, AXI, OCP,...)

CoWare PA (SystemC, HDL)

| M | S |
| PAC |

Examples of Application

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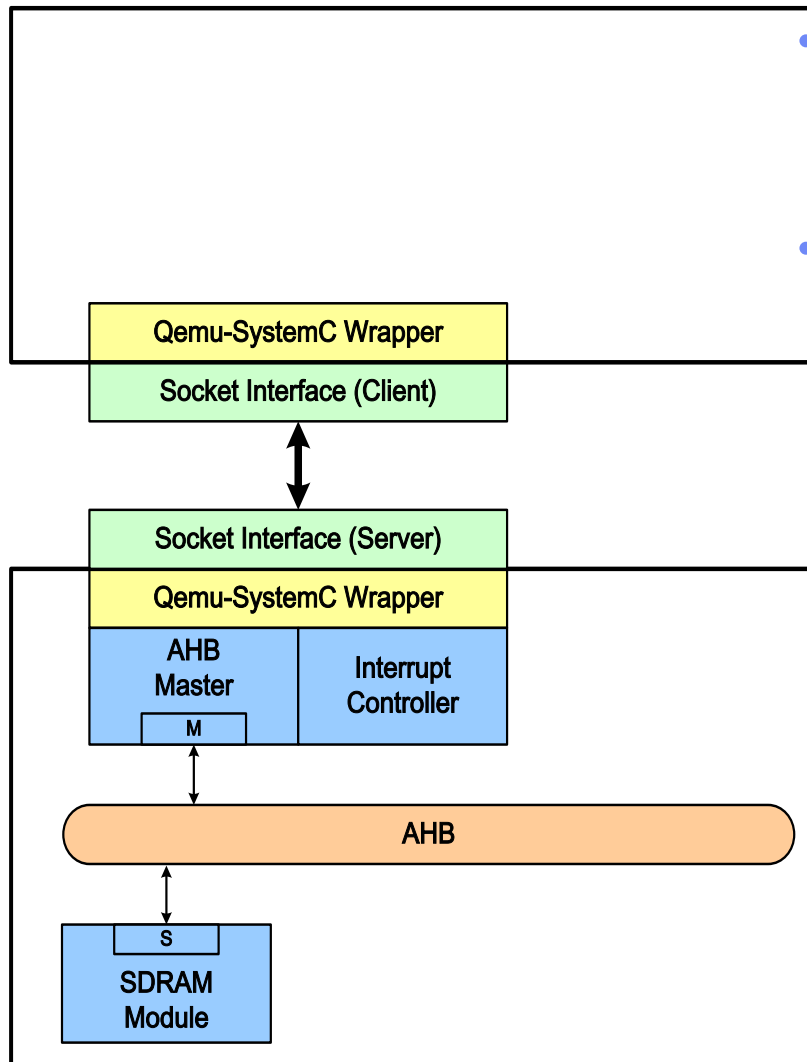- Network SCTP/IP offload design

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
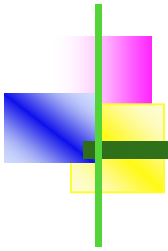
Heterogeneous Multi-Core

- 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

NCKU EE CASLab

FULL SYSTEM VERIFICATION PLATFORM FOR MULTI-VIEW GPU

| Qemu-SystemC Wrapper |
| Socket Interface (Client) |

| Socket Interface (Server) |
| Qemu-SystemC Wrapper |

| AHB Master | Interrupt Controller |
| M | |

AHB

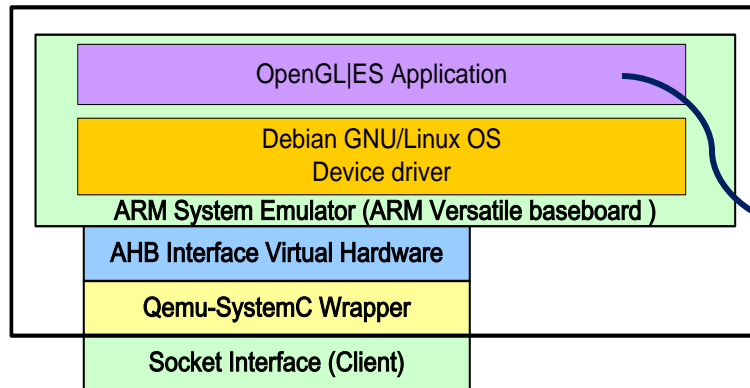| S |
| SDRAM Module |

- 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

Qemu (C, C++)

| OpenGL|ES Application |
| Debian GNU/Linux OS Device driver |
| ARM System Emulator (ARM Versatile baseboard) |
| AHB Interface Virtual Hardware |
| Qemu-SystemC Wrapper |
| Socket Interface (Client) |

```
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);
```

SystemC

CoWare (SystemC, HDL)

| Socket Interface (Server) |
| Qemu-SystemC Wrapper |

| AHB Master | Interrupt Controller | P |
| M | | |

| Geometry Engine | P |
| M | S |

| AHB |

| S | | | M | S | P | | M | S | P |
| SDRAM Module | | | Rasterizer Engine | | | DIBR Engine | |
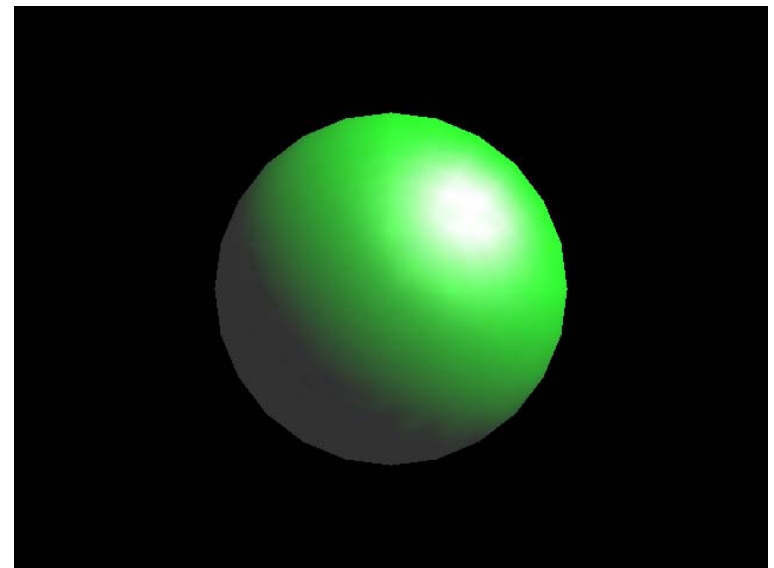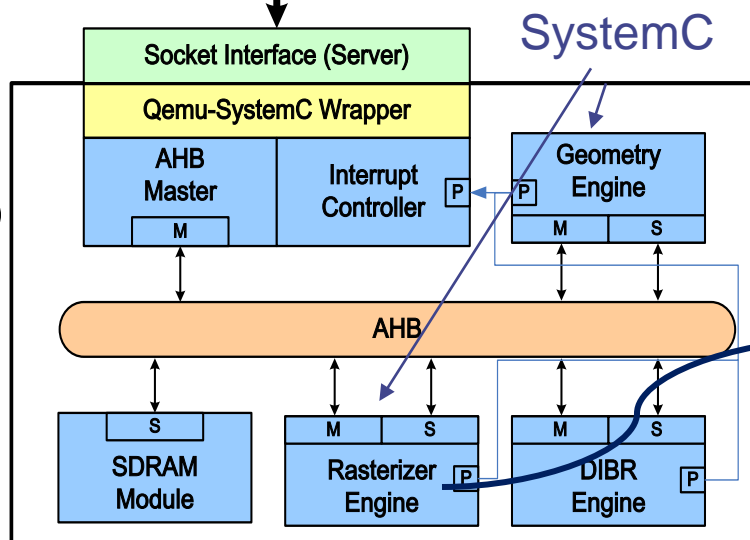
GPU in fresh RTL modules

- GPU with RTL encapsulation

Qemu (C, C++)

- OpenGL|ES Application
- Debian GNU/Linux OS Device driver
- ARM System Emulator (ARM Versatile baseboard)
- AHB Interface Virtual Hardware
- Qemu-SystemC Wrapper
- Socket Interface (Client)

```
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);
```
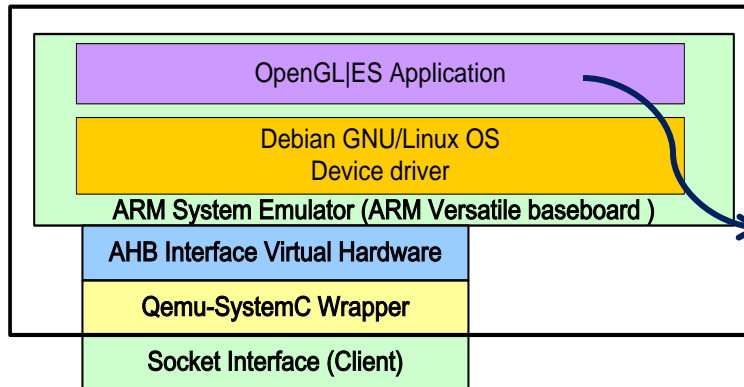
CoWare (SystemC, HDL)

- Socket Interface (Server)
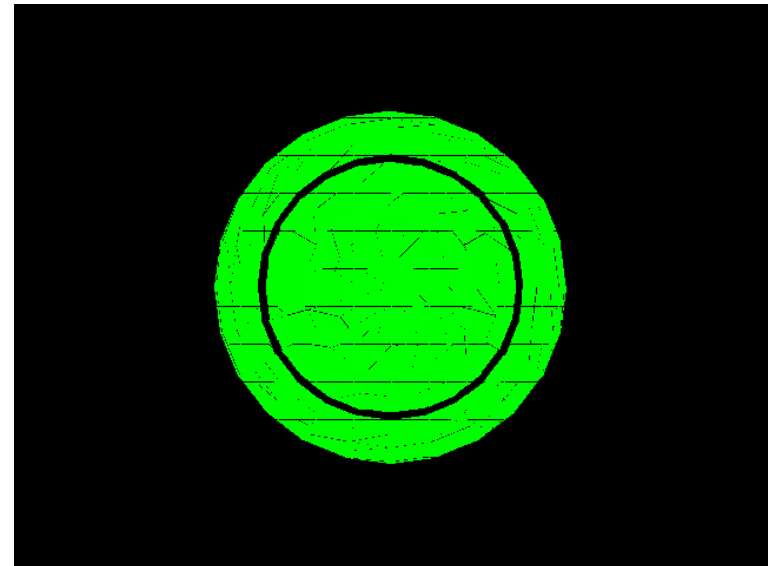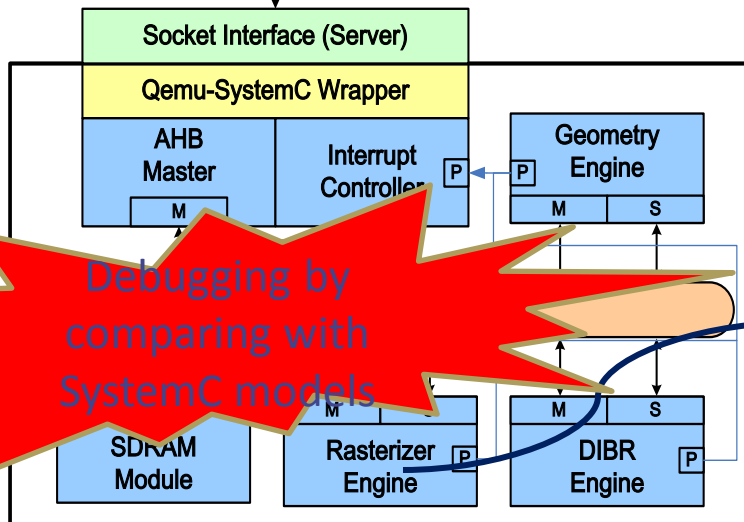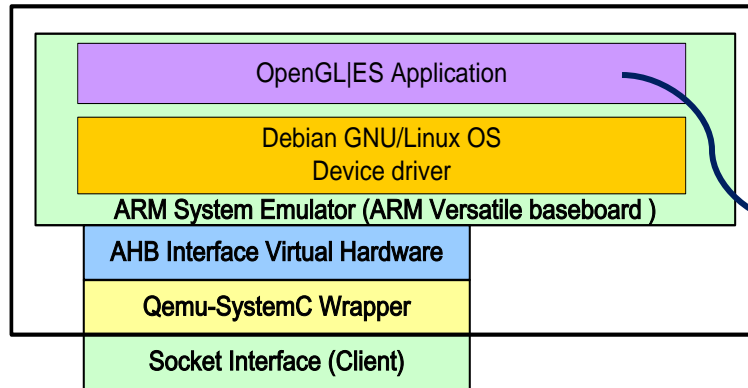- Qemu-SystemC Wrapper
- AHB Master   M
- Interrupt Controller   P   P
- Geometry Engine   M   S
- SDRAM Module
- Rasterizer Engine   M   P
- DIBR Engine   M   S   P

Debugging by comparing with SystemC models

100 % FULL SYSTEM VERIFICATION

- GPU with RTL encapsulation
- RTL verification confirmed

Qemu (C, C++)

| OpenGL|ES Application |
| Debian GNU/Linux OS Device driver |
| ARM System Emulator (ARM Versatile baseboard) |
| AHB Interface Virtual Hardware |
| Qemu-SystemC Wrapper |
| Socket Interface (Client) |

```
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);
```
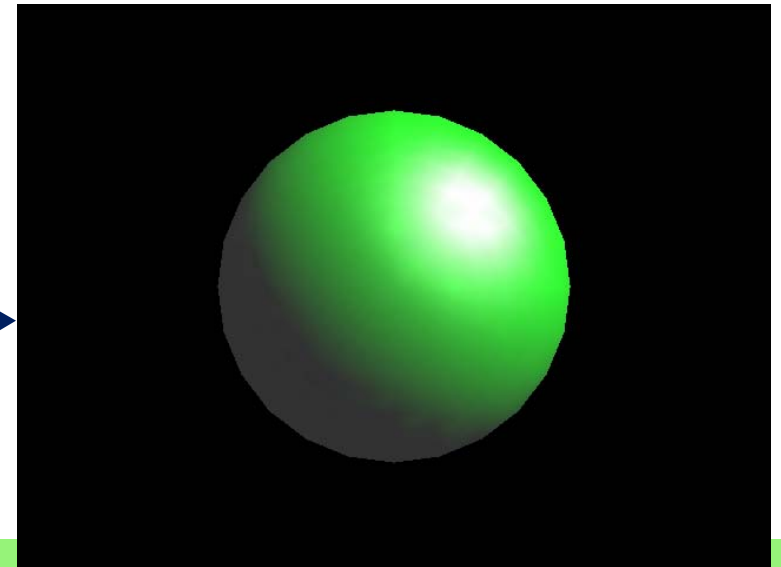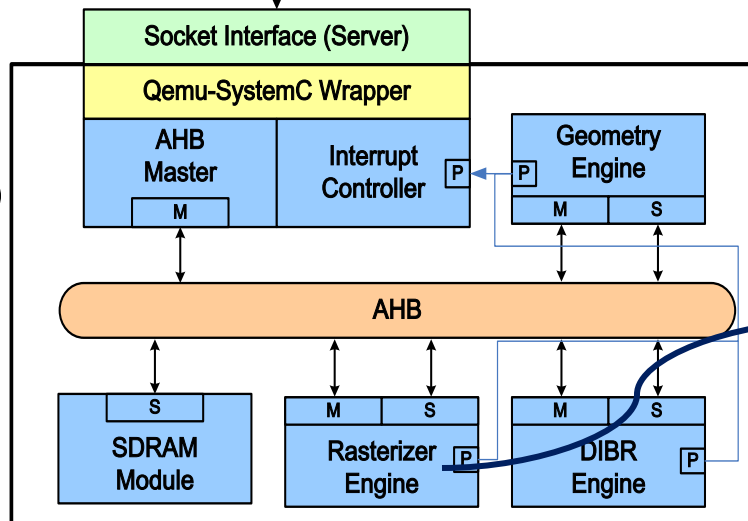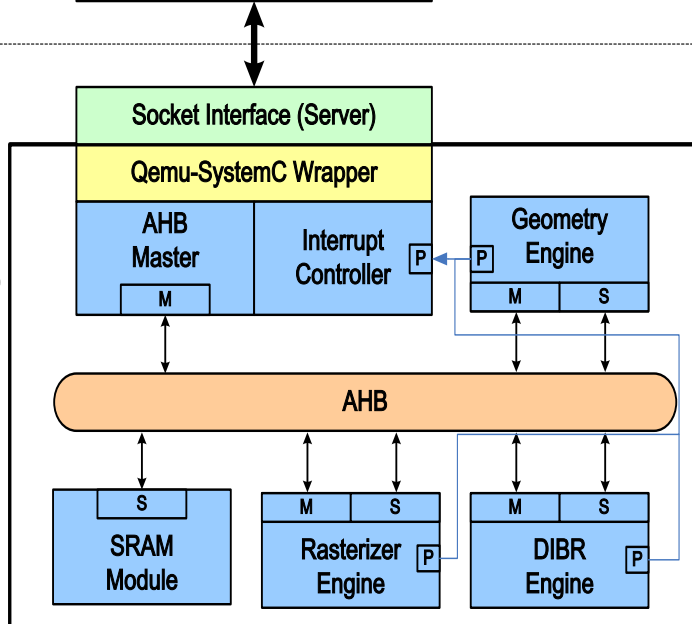
CoWare (SystemC, HDL)

| Socket Interface (Server) |
| Qemu-SystemC Wrapper |

AHB Master | Interrupt Controller | P | P | Geometry Engine
M | | | M | S

AHB

S | M | S | M | S

SDRAM Module | Rasterizer Engine | P | DIBR Engine | P

Flexibility

Qemu
(C, C++)

OpenGL|ES Application

Debian GNU/Linux OS
Device driver

ARM System Emulator (ARM Versatile baseboard)

AHB Interface Virtual Hardware

Qemu-SystemC Wrapper

Socket Interface (Client)

Socket Interface (Server)

Qemu-SystemC Wrapper

CoWare
(SystemC, HDL)

AHB Master | Interrupt Controller | P | P | Geometry Engine
M | M | S

AHB

S | M | S | M | S
SRAM Module | Rasterizer Engine | P | DIBR Engine | P

- 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

Computer 1

Physical Host 1

QEMU	QEMU
Virtual Host 1 *(O/S, Driver, Application…)*	**Virtual Host 2** *(O/S, Driver, Application…)*
Qemu-SystemC Interface	*Qemu-SystemC Interface*
Network Offload Engine	*Network Offload Engine*
Virtual MAC	*Virtual MAC*

**② **

Virtual Network

CoWare (SystemC, C/C++, HDL)

①

Raw Socket API

NIC (Network Interface Card)

Ethernet

Computer 2

Physical Host 2
(O/S, Driver, Application…)

NIC

1. Functional verification
2. Connection with real world (path1)
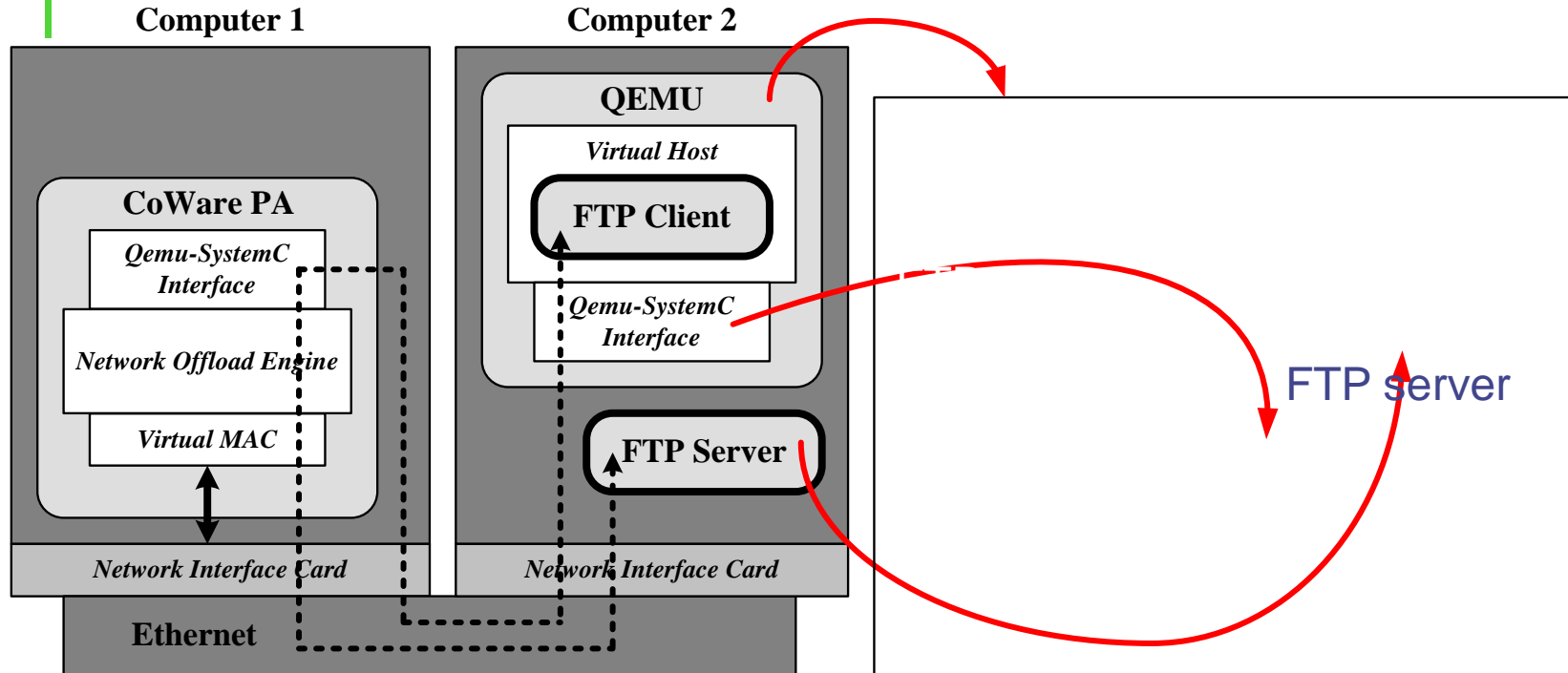3. Performance evaluation for 10 Gb (path 2)
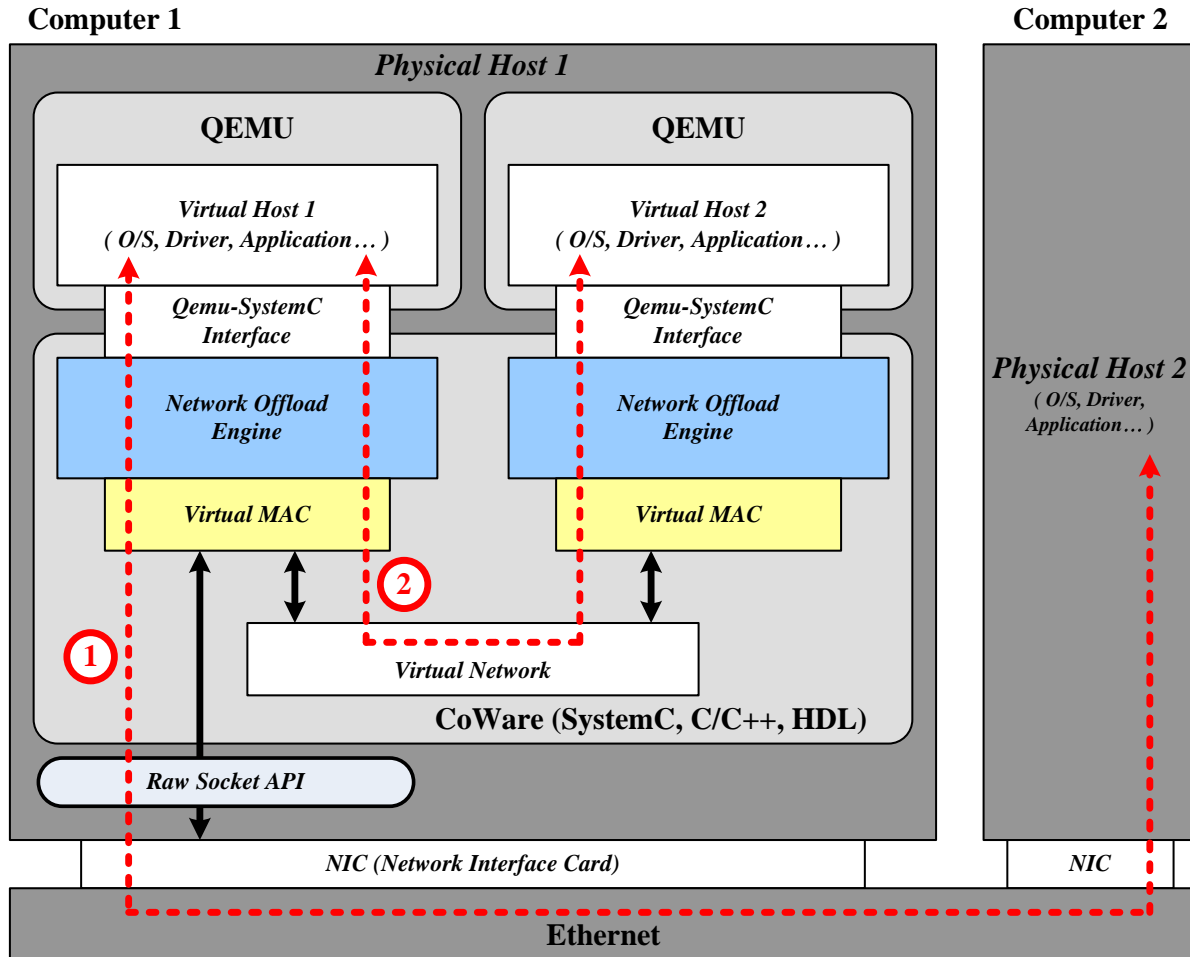
SCTP/IP Offload System

- ## 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)

Computer 1   **Computer 2**

QEMU

Virtual Host

FTP Client

CoWare PA

Qemu-SystemC Interface

Qemu-SystemC Interface

Network Offload Engine

FTP Server

Virtual MAC

Network Interface Card   *Network Interface Card*

Ethernet
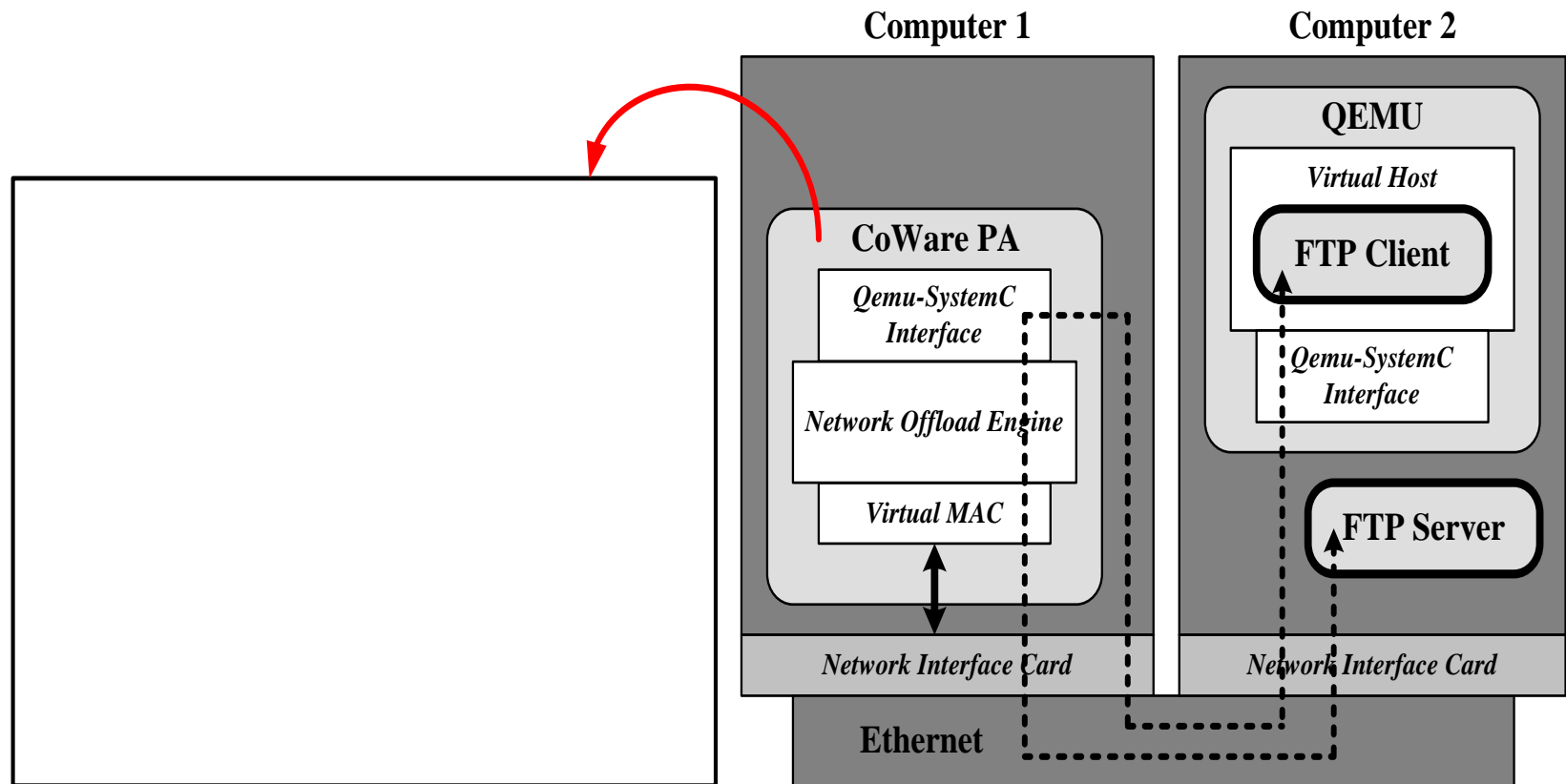
Network Offload System

- 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