



Computer Architecture

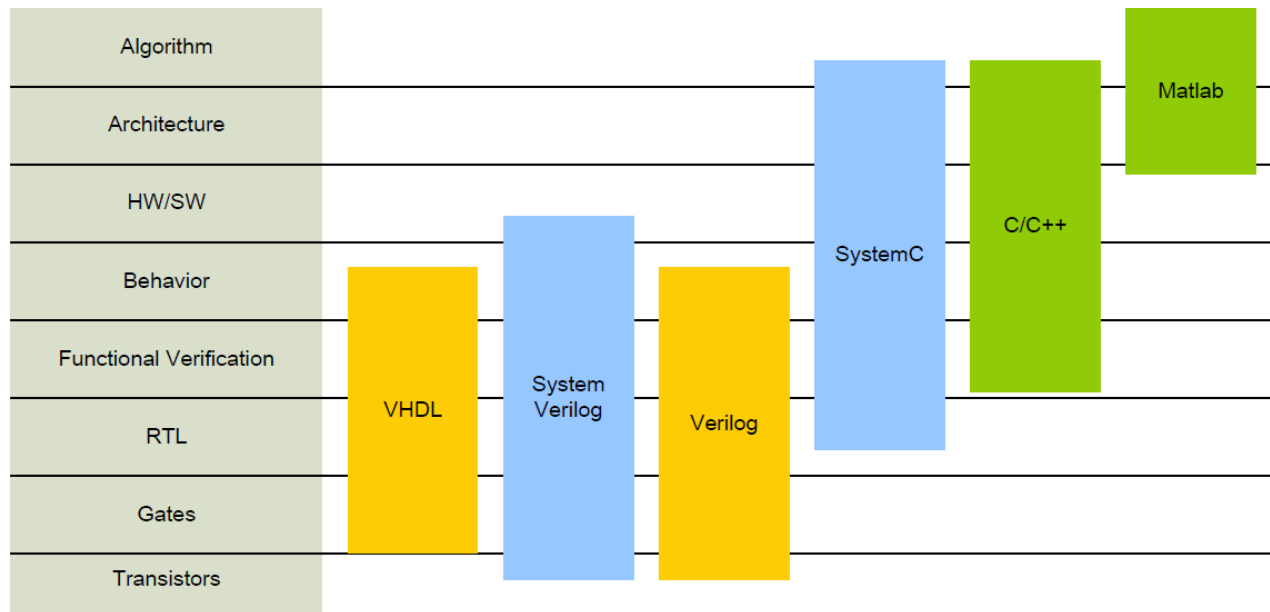
LAB4: SystemC Module & Full System
Simulation using QEMU-SystemC

SystemC Introduction

- SystemC is a C++ library defined and promoted by OSCI which provide developer model the hardware module by C++ language.
- Event driven simulation and translation level modeling (TLM) make SystemC is able to simulate the complex system in the acceptable time that is difficult achieve in traditional RTL simulation such as Verilog.
- The hardware behavior is written in C++ language and simulated by SystemC STL using **SC_METHOD**, **SC_THREAD** or **SC_CTHREAD**.

Translation Level Modeling

- TLM simplifies the communication between the modules by modeling the communication behaviors without real bus protocols.
- Compared to traditional RTL design, TLM should model the modules in cycle accurate, TLM has more abstraction levels.



Full System Simulation

- Simulation
 - Keep only the necessary detail of the real world, while getting the information we want.
 - Benefits: Economic, parameterizable modules, profiling
- Full system simulation platform
 - Design computer hardware and software including device drivers and applications simultaneously.
- Problems in hardware/software co-design
 - Tradeoff:
 - Accurate simulators is too slow to run an operating system like Linux.
 - Functional simulations run fast but lose accuracy of hardware behavior.
 - It's also difficult to develop a device driver with the cycle-accurate simulator in early stage.

Full System Simulation

■ Level of Detail

- Run unmodified commercial operating systems (OS)
- Run realistic workload under reasonable simulation time
- Provide certain hardware timing accuracy
- Provide certain verification & profiling tools

■ Examples:

- Simics - A Full System Simulation Platform
 - 2002 IEEE Computer
 - P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt and B. Werner
- The M5 Simulator - Modeling Networked System
 - 2006 IEEE Micro.
 - N. L. Binkert, R. G. Dreslinski, L. R. Hsu, K. T. Lim, A. G. Saidi and S. K. Reinhardt

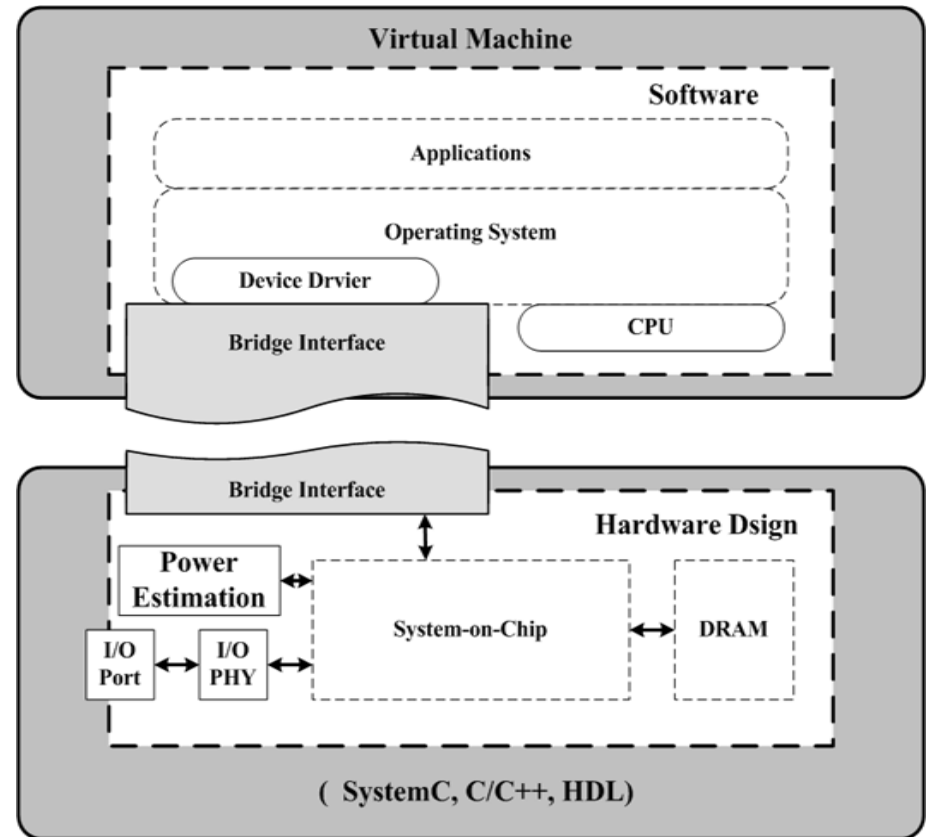


Full System Simulation Platform

- In the previous LABs, we have introduced **how to implement a virtual hardware in the platform emulated by QEMU.**
- We also mention **the Linux device driver**, so we can create a Linux environment and use device driver to control the virtual hardware emulated in QEMU.
- In order to make our calculator simulation more precisely, the next step is **modeling the calculator in SystemC.**

Overview: Hardware/Software Co-Design

- ⊕ Hardware
 - SystemC
 - RTL model
- ⊕ Software
 - Linux OS
 - Applications
 - Device driver
- ⊕ Bridge interface
 - Inter-Process Communication(IPC)
 - Socket call
 - Shared memory



Example: QEMU and HW Simulator for ARM embedded system

⊕ Hardware

⊕ Hardware offload engine

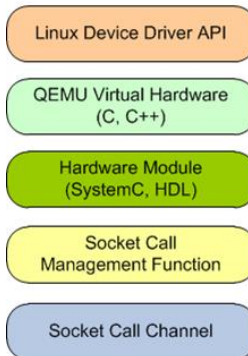
⊕ calculator

⊕ AHB Interface

⊕ Qemu-SystemC bridge

⊕ AHB Bus

⊕ TLM Bus



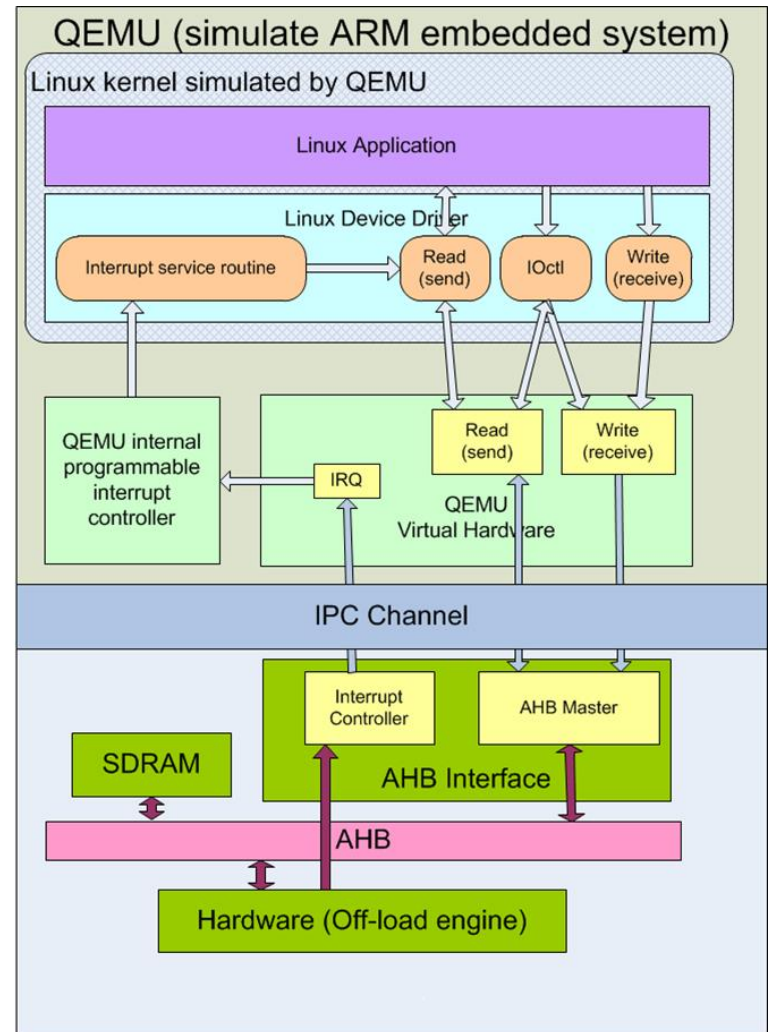
⊕ Software

⊕ Linux kernel

⊕ Application and driver for calculator

⊕ Bridge interface

⊕ Socket call between QEMU virtual hardware and AHB interface in SystemC



Summary

- 為了實現QEMU-SystemC全系統模擬平台，必須完成下列事項。
 1. 將硬體從QEMU內純C語言模擬轉換成以SystemC語法實現。
 2. 必須實現SystemC模組與QEMU之間的連結橋梁。
 3. 實現此連結橋梁以及硬體之間的傳輸方法。
- 連結橋梁的部分又分為QEMU端與SystemC端，QEMU和SystemC之間會用IPC溝通
 - 本次實驗採用share memory的方式進行兩端的溝通
- 連結橋樑與硬體之間的傳輸方法將會採用TLM bus的方式實作System C 模組之間的溝通

Install SystemC and TLM

- 下載SystemC與TLM

- <http://www.accellera.org/downloads/standards/systemc>
- 目前最新版本為2.3，此實驗所採用的是2.2
- TLM版本採用2.0.1

- 以下步驟將SystemC安裝至~/toolchain/systemc-2.2

```
# tar -xvf systemc-2.2.0.tgz
# mkdir ~/toolchain/systemc-2.2
# cd systemc-2.2.0
# ./configure --prefix=/home/{username}/toolchain/systemc-2.2
# make && make install
# cp TLM-2.0.1.tgz ~/toolchain/.
# tar -xvf ~/toolchain/TLM-2.0.1.tgz
# mv -rf ~/toolchain/TLM-2009-07-15 ~/toolchain/TLM-2.0
```

SystemC_Module (LAB4 attached file)

- 檔案結構示意圖(LAB4 attached file) :

SystemC_Module (建議將此置於家目錄)



qemu_sc_bridge (連結QEMU與SystemC連結橋樑模組)

calculator (與LAB3功能相同的計算機硬體模組)

TLM_BUS

Defs.mk

Makefile

- 請同學將於**Defs.mk**中將兩個參數 **SC_INC** 與 **SC_LIB** 修改成實際的 SystemC和TLM-2.0安裝路徑內的include資料夾與lib資料夾

範例:

```
SC_INC      :=      -I/home/caslab/toolchain/systemc-2.2/include/ -I/home/caslab/toolchain/TLM-2.0/include/tlm/  
SC_LIB     :=      -L/home/caslab/toolchain/systemc-2.2/lib-linux64
```

SystemC_Module (LAB4 attached file)

- 以下簡介每個檔案的用途以及模擬方法，假設同學皆熟悉SystemC語法
- Defs.mk
 - 定義各個compiler參數，其中platform與target為我們所要編譯的目標，範例為cal，編譯完成後可在~/SystemC_Module/bin底下找到編譯完成的執行檔。
 - sub_dir則是目標程式碼的資料夾，資料夾的擺放規則為.cpp檔直接放於資料夾中，而標頭檔(.h)則必須在資料夾內新增include資料夾並放在include資料夾內。

Calculator (SystemC module)

■ Calculator class (calculator.h)

- C++ 為物件導向程式設計，因此每個systemc module都是以一個class 宣告。
- 在calculator此物件中以sc_in_clk宣告clock的input port以及當運算結束後，calculator必須發送interttupt的output port，以sc_out<bool> 宣告
- read, write函示提供給master存取的方法，而local_access函式則是提供與TLM bus接軌。

Calculator (SystemC module)

- Calculator computing (calculator.cpp)
 - calculator.cpp為QEMU的虛擬硬體轉移至SystemC的硬體模組程式碼
 - 本實驗所提供的Calculator的設計為一種序向電路，主要執行函示(run)採用SC_THREAD模擬，在每次clock positive edge被觸發一次
 - SC_THREAD模擬的過程中，將會以wait()表示此clock執行結束，待下次clock觸發時，會從wait()的下一行開始做起。因此我們設定加與減執行1個clock，乘為4個而除則需要13個cycle。
 - SC_THREAD(run);
sensitive << clk.pos();

Qemu-SystemC bridge

- Bridge class(qsbridge.h)
 - 與calculator一樣，有個clk的input port。
 - 由於bridge必須將calculator的interrupt回送給QEMU，因此interrupt必須由bridge代收，所以有個sc_in<bool>的irq接收port。
 - 在此版本中，我們採用share memory當成與QEMU之間的溝通方法，而且data傳送以及IRQ的傳送必須依靠兩組IPC，因此有兩組share memory KEY。

Qemu-SystemC bridge

- 與calculator相同，bridge一樣是以clock positive edge觸發當成執行條件。
- 每次clock正緣觸發時，bridge透過讀取share memory內的值去判斷QEMU端是否有data要寫入calculator的memory或register。
- 而在run函式一開始，也必須判斷calculator是否有interrupt要回傳給QEMU。
- Bridge內也必須實現read, write function以與TLM bus接軌。透過bus_b_access可以直接鏈結到calculator中的local_access。

TLM BUS

- 本實驗所提供的TLM bus包含AHB master與slave，可以採用function call之方式進行記憶體位址存取的模擬
- Master模組(此為qsbridge)必須繼承ahb_master_if，並在其read, write函示中透過bus_b_access將所要讀寫的address及data傳送出去
- Slave 模組 (此 為 calculator) 則 繼 承 ahb_slave_if，並且實現local_access函式，承接bus_b_access的資訊並做出相對應的動作
- 將兩者對應鏈結起來的方法為TLM_BUS資料夾中decoder.cpp的程式碼

Top Module

- 在calculator資料夾中的top.cpp將qsbridge與calculator做接線連結。
 - 首先兩者都連到global clock，在這模擬為4Mhz的clock。
 - calculator的irq_out必須連結到qsbridge的irq_in。
 - 宣告global bus，bus的class名稱為ahb
 - Master模組透過ahb_master_socket.bind(my_bus
->from_master_socket)鍊結至TLM bus。
 - Slave模組則透過ahb_to_slave_socket.bind(cal
->ahb_slave_socket)與TLM bus連結
 - Slave模組還必須加上address的mapping，採用add_mapping函式，有三項參數：第一為slave_id，TLM bus透過此ID判斷要將data送給哪個slave；第二項為mapping的初始位址；第三個參數是mapping的大小，目前以0x1000為最小分配單位。

main_module

- SystemC的模擬進入點為**sc_main()** (如同 C的main function)
- 在main_module.cpp中先宣告我們剛才接線完成的top module。
- 完成硬體設計之後，最後再呼叫**sc_start()**開啟SystemC硬體模擬。
- 所有程式碼都完成後就可以透過make來編譯我們的程式，編譯完成的程式會在bin資料夾中
- 並對Defs.mk做以下修改，再執行以下編譯

```
ifeq "$(MAKECMDGOALS)" "cald"
platform          :=      cald
target           :=      cald
sub_dirs         +=      qemu_sc_bridge calculator TLM_BUS
CPPFLAG          +=      $(DEBUG_FLAG)
endif
```

```
# cd ~/SystemC_Module
# make cald
```

QEMU

- 為了讓 qemu 可以和 SystemC_Module 連結，我們採用 share memory 做為 IPC，DATA 和 IRQ 的 key 必須與 systemc 模組內使用的相同。
 - `qemu-0.15.1/hw/sysbridge.c`
- 設計流程與 LAB3 的 `calculator.c` 一樣，但是 `calculator.c` 收到 CPU 的寫入命令時直接做出相對應行為，而 `sysbridge` 則是將這些命令再透過 share memory 轉送給 SystemC_Module。
- 由於 IRQ 發送的時間無法確定，所以我們用 `pthread` 來實現，讓一條 thread 去 polling 是否有 IRQ 要傳送回來。

QEMU

- 本實驗一樣採用 realview-eb 作為模擬平台，因此一樣在 hw/realview.c 中加入我們新設計的 calculator。
- 將付Lab4附檔
 - sysbridge.c 複製到 qemu-0.15.1/hw/
 - Makefile.target 複製到 qemu-0.15.1/
 - 以下以完成於付檔
 - 為了讓先前的 driver 可以不做任何修正，位址一樣訂為 0x80000000，IRQ 阜號一樣是 30。
sysbus_create_simple("sysbridge", 0x80000000, pic[30]);
 - 如同 LAB3，我們必須在 Makefile.target 中適當位置加入
obj-arm-y += sysbridge.o
- 本實驗將使用 pthread，請注意編譯 QEMU 前的 configure 動作要通知 compiler 將 pthread library 載入

```
# ./configure --target-list=arm-softmmu --prefix={your-dir}/qemu-bin --extra-ldflags=-lpthread
# make && make install
```

Full System Simulation

- 以上完成以後，我們就可以透過新的qemu-system-arm與SystemC_Module/bin/cal來做full system simulation。
- 分別在兩個terminal下進行
 - (以下指令為先前有將這些檔案link/copy到qemu-test的狀況下執行)
./cald #(at host SystemC_model/bin
./qemu-system-arm -M realview-eb -kernel zImage -initrd initrd.gz -cpu arm1136

```
##In_qemu (do as lab3)
```

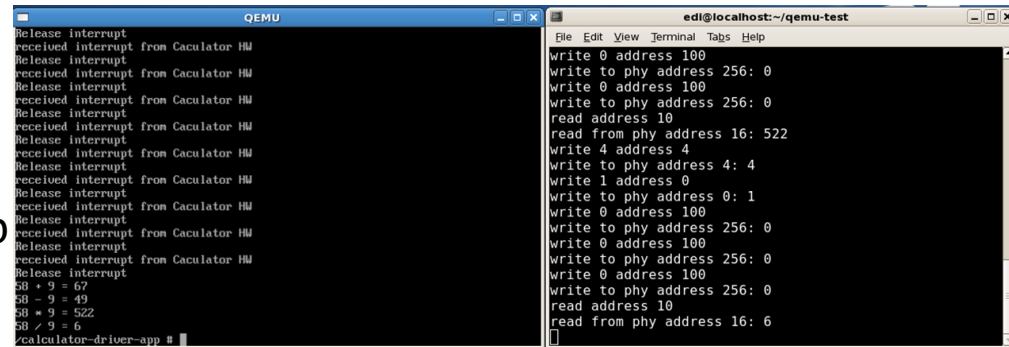
```
$ cd calculator-driver-app
```

```
$ insmod cal_drv.ko
```

```
$ cat /proc/devices
```

```
$ mknod /dev/calculator c 254 0
```

```
$ ./calculate 4 6
```



The image shows two terminal windows side-by-side. The left window is titled 'QEMU' and shows a series of 'Release interrupt from Calculator HW' messages, followed by calculations: '58 * 9 = 67', '58 * 9 = 49', '58 * 9 = 522', and '58 / 9 = 6'. The right window is titled 'edi@localhost:~/qemu-test' and shows hardware-level operations: 'write 0 address 100', 'write to phy address 256: 0', 'write 0 address 100', 'write to phy address 256: 0', 'read address 10', 'read from phy address 16: 522', 'write 4 address 4', 'write to phy address 4: 4', 'write 1 address 0', 'write to phy address 0: 1', 'write 0 address 100', 'write to phy address 256: 0', 'write 0 address 100', 'write to phy address 256: 0', 'write 0 address 100', 'write to phy address 256: 0', 'read address 10', and 'read from phy address 16: 6'.

(QEMU)

(SystemC)