Computer Architecture

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

SystemC Introduction

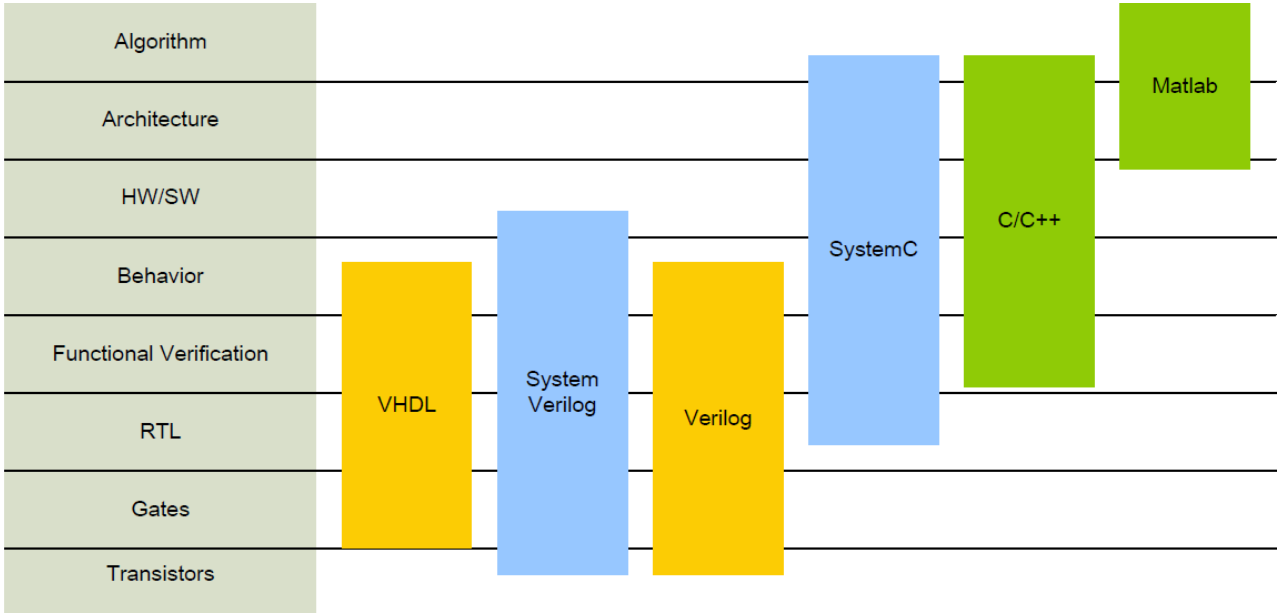
- SystemC is a C++ library defined and promoted by OSCI which provide developer model the hardware module by C++ language.

- 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 model the communication behaviors without real bus protocols.
- Compare to traditional RTL design should model the modules in cycle accurate, TLM have 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

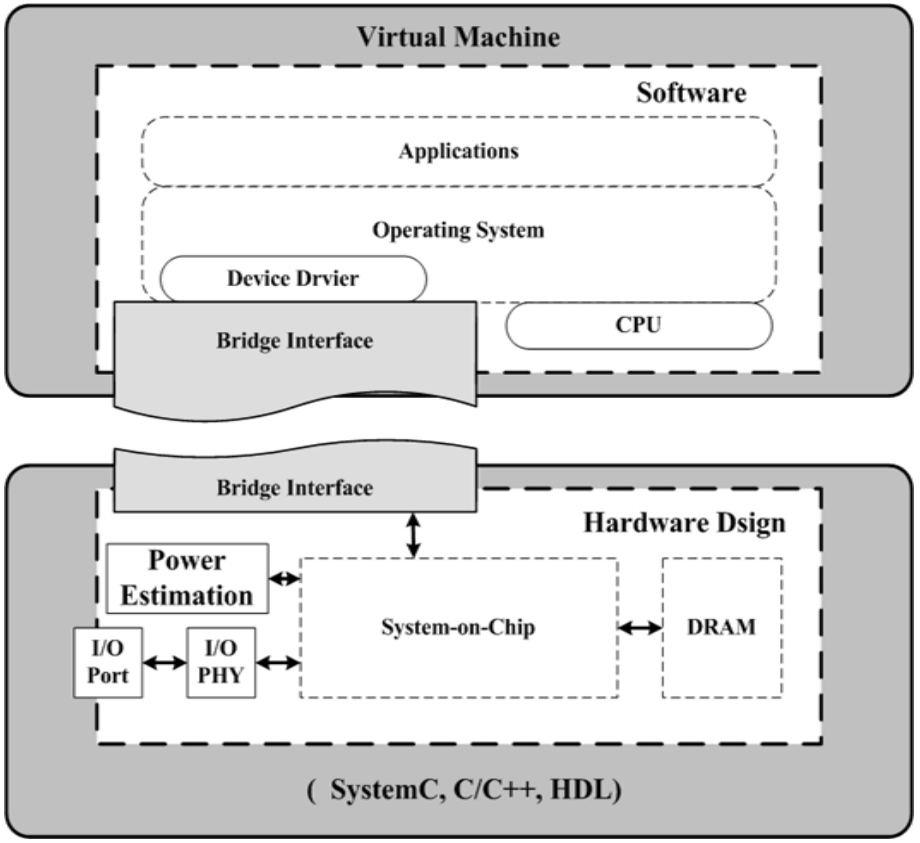
- In the previous LABs, we have introduced **how to implement a virtual hardware in the platform emulated by QEMU**.

- 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

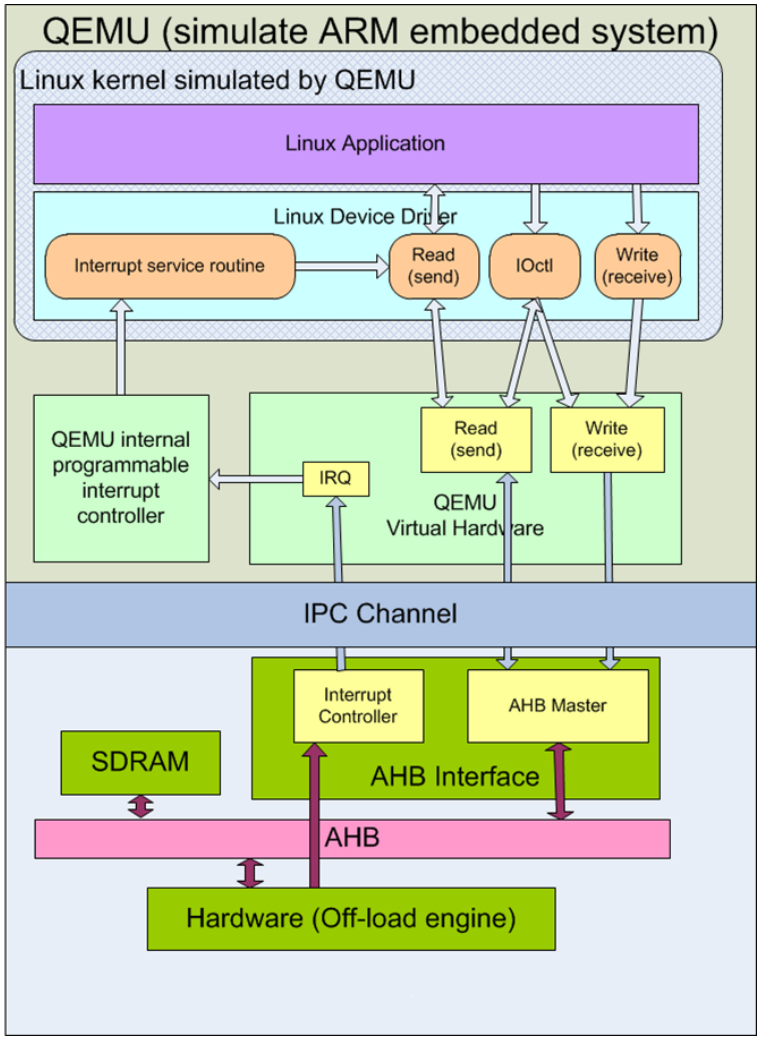
Linux Device Driver API

QEMU Virtual Hardware (C, C++)

Hardware Module (SystemC, HDL)

Socket Call Management Function

Socket Call Channel



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
  - 此實驗所採用的是2.3.3
  - TLM版本採用2.3.3所附的TLM 2.0.5

- 以下步驟將SystemC安裝至~/workspace/SystemC

```
# tar -xvf systemc-2.3.3.gz -C ~/workspace/Src
# cd ~/workspace/Src/systemc-2.3.3 && mkdir build
# cd build
# ../configure --prefix=${HOME}/workspace/SystemC
# make && make install
# echo 'export
LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:~/workspace/SystemC/lib-linux64' >>
~/.bash_aliases
```

SystemC_Module (LAB4 attached file)

- 檔案結構示意圖(LAB4 attached file)：

```
SystemC_Module
↳ 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
  Makefile
```

- 請同學將於makefile中將兩個參數 SYSTEMC_DIR 修改成實際的 SystemC 安裝路徑（如照講義路徑則不需修改）

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

- Makefile
  - 定義各個compiler參數，其中EXECUTABLE為我們所要編譯的目標，範例為qsysbridge，編譯完成後可在~/workspace/SystemC_Module/底下找到編譯完成的執行檔。
  - INCDIR、 SRCDIR則是目標程式碼的資料夾，資料夾的擺放規則為.cpp檔放於Source資料夾各自目錄中，而標頭檔(.h)則放在Include內對應的資料夾目錄

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interrput必須由bridge代收，所以有個sc_in<bool> 的irq接收port。
  - 而為了可能需求，額外於bridge內增加reset pin，可接收Bus Command來觸發Reset signal。
  - 在此版本中，我們採用share memory當成與QEMU之間的資料傳輸方法，而為了節省CPU資源，採用Semaphore作為Bus事件觸發來源，因此需要一組Share memory ID、三組Semaphore ID以建立雙向溝通。

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透過確認DATA semaphore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。
  - Calculator的rst_in必須連結到qsbridge的rst_out。
  - 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_socker.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；第三個參數式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.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

```
# cd ~/workspace/SystemC_Module
# make
```

QEMU

- 為了讓qemu可以和SystemC_Module連結，我們採用share memory做為IPC，Share memory 的key與semaphor的Path必須與systemc模組內使用的相同。
  - qemu-3.0.0/hw/char/caslab_sysbridge.c

- 設計流程與LAB3的caslab_calculator.c一樣，但是caslab_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去等待是否有IRQ要傳送回來、進行更新。

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。

- 為了讓先前的driver可以不做任何修正，位址一樣訂為**0x80000000**，IRQ皐號一樣是**30**。

```
sysbus_create_simple("caslab_sysbridge",0x80000000, pic[30]);
```

- 如同LAB3，我們必須在Makefile.target中適當位置加入

```
obj-$(CONFIG_REALVIEW += caslab_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

```
build]# ../configure --target-list=arm-softmmu --
prefix=${HOME}/workspace/ qemu-bin-3.0.0 --extra-ldflags=-
lpthread
build]# 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/qsysbridge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到qemu-test的狀況下執行)

```
# ./qemu-system-arm -M realview-eb -kernel zImage -initrd
initrd.gz -cpu arm1136 -dtb arm-realview-eb.dtb
# ./qsysbridge
```

```
/home # ./calc 155 5
IOCTL CMD = 3, value = 1
IOCTL CMD = 4, value = 1
received interrupt from caslab_calc_hw
Release interrupt
IOCTL CMD = 3, value = 2
IOCTL CMD = 4, value = 1
received interrupt from caslab_calc_hw
Release interrupt
IOCTL CMD = 3, value = 3
IOCTL CMD = 4, value = 1
received interrupt from caslab_calc_hw
Release interrupt
IOCTL CMD = 3, value = 4
IOCTL CMD = 4, value = 1
received interrupt from caslab_calc_hw
Release interrupt
155 + 5 = 160
155 - 5 = 150
155 * 5 = 775
155 / 5 = 31
/home #
```

(QEMU)

```
        SystemC 2.3.3-Accellera --- Dec 11 2018 12:15:42
        Copyright (c) 1996-2018 by all Contributors,
        ALL RIGHTS RESERVED
map_size: 0x3FFFFF, check i: 0x3FFFFF
mapping_size: 4 (KiB), slave id: 0
base address: 0x0, end address: 0x1000
Start Simulating.......
Waiting QEMU UP....
Connected to QEMU.
[QSBRIDGE]: BUS Reset
[QSBRIDGE]: BUS Write, Write 0x009b to 0x00000008
[QSBRIDGE]: BUS Write, Write 0x0005 to 0x0000000c
[QSBRIDGE]: BUS Write, Write 0x0001 to 0x00000004
[QSBRIDGE]: BUS Read, read from 0x00000014
[QSBRIDGE]: BUS Write, Write 0x0001 to 0x00000000
[QSBRIDGE]: BUS Write, Write 0x0004 to 0x00000000
[QSBRIDGE]: BUS Read, read from 0x00000010
[QSBRIDGE]: BUS Write, Write 0x0002 to 0x00000004
[QSBRIDGE]: BUS Read, read from 0x00000014
[QSBRIDGE]: BUS Write, Write 0x0001 to 0x00000000
[QSBRIDGE]: BUS Write, Write 0x0004 to 0x00000000
[QSBRIDGE]: BUS Read, read from 0x00000010
[QSBRIDGE]: BUS Write, Write 0x0003 to 0x00000004
[QSBRIDGE]: BUS Read, read from 0x00000014
[QSBRIDGE]: BUS Write, Write 0x0001 to 0x00000000
[QSBRIDGE]: BUS Write, Write 0x0004 to 0x00000000
[QSBRIDGE]: BUS Read, read from 0x00000010
[QSBRIDGE]: BUS Write, Write 0x0004 to 0x00000004
[QSBRIDGE]: BUS Read, read from 0x00000014
[QSBRIDGE]: BUS Write, Write 0x0001 to 0x00000000
[QSBRIDGE]: BUS Write, Write 0x0004 to 0x00000000
[QSBRIDGE]: BUS Read, read from 0x00000010
```

(SystemC)