Intel Core i7 - PCIExpress

Intel Core i7 platform with PCIExpress

a604dc8f (/platforms/i7-pcie/commit/a604dc8ff0e55a133a998f0c9e3476ae54a8fb34) release version 1.0 (/platforms/i7-pcie/commit/a604dc8ff0e55a133a998f0c9e3476ae54a8fb34) · 3 years ago by **KONRAD Frederic** (mailto:fred.konrad@greensocs.com)

README

Features of qemu_sc and QBox

Both QBox and qemu_sc are based on QEMU. For more information about QEMU, visit QEMU.org. QBox and qemu_sc both allow SystemC models to be used in conjunction with QEMU. For more information about SystemC please visit accellera.org. QBox allows QEMU to be embedded inside a SystemC simulation environment, while qemu_sc allows a SystemC models to be embedded inside a QEMU simulator. You should use QBox when you need a CPU simulator inside a wider platform model. You should use qemu_sc when you would like to add SystemC models to an existing QEMU platform.

TLM-2.0 suport

Both QBox and qemu_sc use TLM-2.0 as the bus API. Specifically GreenSocket is used to implement the TLM-2.0 standard so that binding checks and memory management are handled correctly to all TLM-2.0 compatible models. GreenSocket is also used for the interrupt (slave) sockets for both qemu_sc and QBox. Finally, qemu_sc supports a DMA channel (slave bus port) which is also implemented with a GreenSocket (TLM-2.0 slave).

DMI support

QBox supports DMI. It follows the following heuristic to find memories. It will look for any GreenSocket TLM bus sockets registering it’s address with a router (it uses the fact that GreenSocket TLM Sockets include a parameter that identifies their base address to the router). QBox will refuse to execute code outside of DMI space, hence it is imperative that the memory model supports DMI. qemu_sc does not include this feature, as the memory is expected to be within QEMU.

Quantum support

Both qemu_sc and QBox use quantums. This allows the CPU model to execute more efficiently. They are strict quantums (as defined by TLM-2.0). However QBox also allows the CPU to execute inside a different thread from the rest of SystemC and QBox ensures that SystemC execution only happens within the SystemC thread. This is relatively expensive, so QBox IO accesses are typically very slow and incur multiple context switches. On the other hand, as the SystemC model is embedded within qemu_sc, it is executed within the same thread (officially the QEMU IO thread) and no context switching is required. The length of the quantum is set via a parameter.

Writing SystemC code that works well with qemu_sc and QBox

Since QBox and qemu_sc will call directly into the SystemC model via TLM-2.0 b_transport (blocking transport), it is highly undesirable to call wait(). Indeed calling wait will have the effect of moving the SystemC time forward outside of the quantum mechanism which may mean that time moves more than one quantum, which could break other models. Events posted into the future will be executed either at the end of the current quantum in the case of QBox or at the approximate CPU time in the case of qemu_sc.

It is extremely bad practise to write SystemC models that 'poll' or 'loop' as the hardware would do. Rather a model should be written to be responsive to IO requests, and respond to those requests allow whenever possible.

Bus access

Both QBox and qemu_sc include GreenLib, in which can be found GreenReg. GreenReg allows models to be written simply and quickly. For instance, the following example shows how to construct a single register in your model. When that register is written, a function is called.

```
    r.create_register(
    "InputA",                   /* register name for external access          */
    "First array input buffer", /* description of register                    */
    0x00,                       /* offset of register relative to device base */
    gs::reg::STANDARD_REG       /* type of register:                          */
    | gs::reg::SINGLE_IO        /*  combination of register_type, register_io,*/
    | gs::reg::SINGLE_BUFFER    /*  register_buffer,                          */
    | gs::reg::FULL_WIDTH,      /*  register_data (orientation)               */
    0x00,                       /* default data                               */
    0x00,                       /* write mask for data coming in from the bus */
    32,                         /* width (should be the reg container width)  */
    0x00);                      /* lock_mask (not implemented).               */
```

GreenReg includes the ability to define callbacks before and after reads and writes, and to define a wide range of different sorts of registers. See greensocs.com/GreenLib for more details.

```
GR_FUNCTION(SimplePCIDevice, setDataBuffer1);
GR_SENSITIVE(r[0x00].add_rule(gs::reg::POST_WRITE,
                              "add data to the first buffer",
                              gs::reg::NOTIFY));
```

Interrupt generation

To generate an interrupt, the model acts as a master - and sends the interrupt to the QBox or qemu_sc. An example of such an interrupt generation would be as follows:

```
void SimplePCIDevice::updateIRQ()
{
  bool irqData = false;
  bool ackRequirement = false;
  sc_core::sc_time time = sc_core::SC_ZERO_TIME;
  gs_generic_signal::gs_generic_signal_payload* payload;

  irqData = irq_signal;


  /*
   * The IRQ is modeled with GreenSignal, no need to create transaction each
   * time: it is done with get_transaction() and release_transaction().
   */
  payload = irq_socket.get_transaction();
  payload->set_data_ptr((unsigned char*) &irqData);
  payload->set_ack_requirement(ackRequirement);
  irq_socket.validate_extension<IRQ>(*payload);
  irq_socket->b_transport(*payload, time);
  irq_socket.release_transaction(payload);
}
```

ToDo:

- Coming Soon... Multi-host thread support.
- Mac OS X and Windows MinGW support

For more information about QEMU SystemC and training on how to use QEMU SystemC and the GreenSocs SystemC library please contact us (http://www.greensocs.com/contact).

Platform

This demonstration creates a PCIE device (SimplePCIDevice) inside QEMU.

This device is an example which is able to:

- compute an addition of two arrays.
- read information from a file.

In order to compute an addition of two arrays:

- First array must be sent to the Array1 input buffer by writing to the right register.
- Second array must be sent to the Array2 input buffer.
- The computation command must be sent (writing to the Computation start register).
- When the computation is done an IRQ is triggered.
- The IRQ must be cleared by reading the status register.

- The status register gives the amount of data present in the output buffer.
- The Output buffer must be read to get the computed array.

In order to read from the file:

- Simply read the output file buffer.

This device uses GreenReg registers with the following mapping: offset | Use

```
0x00 | Array1 input buffer.
0x04 | Array2 input buffer.
0x08 | Computation start.
0x0C | Status register.
0x10 | Output array buffer.
0x14 | Output file buffer.
```

Requirements

QBox requires :

- git > 1.8.2
- systemc >= 2.3.0
- cmake > 2.8
- libboost > 1.34
- python dev
- glib2 dev
- pixman dev
- lua 5.2 dev
- swig

OS support :

- Linux (Debian/Ubuntu, CentOS)
- OS X : soon
- Windows : soon

Download and install requirements

Be sure to clone the repository, including the sub-modules (see below). After cloning the repository, you should run:

```
$ git submodule update --init
$ cd libs/qemu_sc
$ git submodule update --init dtc
$ cd ../..
```

Project (/platforms/i7-pcie) Activity (/platforms/i7-pcie/activity) Repository (/platforms/i7-pcie/tree/master)

Ubuntu / Debian :

```
$ sudo apt-get install cmake libboost-dev python-dev libglib2.0-dev libpixman-1-dev liblu
```

CentOS / Fedora :

```
$ sudo yum install cmake boost-devel python-devel glib2-devel pixman-devel lua-devel swig
```

Compilation

```
$ cmake -DCMAKE_INSTALL_PREFIX=[install dir] -DSYSTEMC_PREFIX=[systemc prefix]
```

Example :

```
$ cmake -DCMAKE_INSTALL_PREFIX=build -DSYSTEMC_PREFIX=/usr/local/lib/systemc-2.3.0/
```

(The default value of `CMAKE_INSTALL_PREFIX` is `/usr/local`).

Compile with :

```
$ make
```

Installation

```
$ make install
```

Run

You can run qemu_sc directly as the images are included in the platform :

```
$ ./libs/qemu_sc.build/bin/qemu-system-x86_64 --enable-kvm -cpu Nehalem -smp 8 --kernel ..
```

Command line parameters

enable-kvm: enable kvm so the simulation is faster. cpu: the kind of CPU you want to emulate or just '?' to have a list. smp: the number of cpu to simulate. kernel: the zImage to load. initrd: the rootfs to load.

A lot of other command line parameter are available look at the qemu documentation:

Lua configuration file

The device load a lua file SimplePCI.lua. Some parameters can be given in this file.

```
-- Global parameters for SimplePCIDevice --
SimplePCIDevice={}
SimplePCIDevice.product_id = "0x4321"

-- Device instance configuration --
Simple={}
Simple.buffer_size = "128"

quantum="100"
```

quantum is given in nanosecond.

You can just copy the SimplePCI.lua file given in the conf directory.

Guest

A precompiled linux guest with the driver for the example device and an application example is provided with this demonstration platform.

The device driver creates a character device: /dev/example-device when the PCI module is found.

When the boot is finished you can log in with "default" user. The application example is vector_addition it takes the character device as a parameter.

```
$ vector_addition /dev/example-device.
```

The application does several thing:

- It reads the file buffer.
- Send two arrays to the Array's buffer.
- Start an addition.
- Get the computed array from the output array buffer.

Eclipse environment

Eclipse can be used to run the platform and debug either the guest or the hardware. A complete tutorial is available in the docs directory.

Modification

The SystemC part of the wrapper is present in ./hw/systemC directory. The device is present in the ./hw/systemC/module directory.

Contributing

QBox is an open source, community-driven project. If you'd like to contribute, please feel free to fork project and open merge request or to send us (http://www.greensocs.com/contact) patch.