Locking and Synchronisation with IPBus

2012/05/25
Rob Frazier, Marc Magrans, Dave Newvold

Table of Contents

One of the design objectives of the IPBus system is to allow concurrent access to hardware by multiple control applications. Currently, the IPBus system guarantees in-order and atomic processing of transactions from a given client. Where a client is either a thread or a process hosted by one or multiple machines.

This is, the IPBus system currently provides sequentially consistent read and write for individual transactions to a given IP endpoint.

This behaviour contrasts with the currently used at CMS for on the VMEBus system (i.e. HW + PCI access service + driver + HAL), which assures in-order and atomic access to the crate.

Unfortunately, the safety properties of IPBus and VMEBus are not enough to cover the use cases needed for hardware access. Currently, there are situations that require exclusive access to the hardware during multiple read and/or write transactions. The most usual use cases are:

- Concurrent JTAG access
- Consistent snapshot of the hardware state

The goal of this document is to specify the changes required in the IPBus system to provide exclusive access to a given IP endpoint for a sequence of read and/or write transactions. This additional concurrency property should keep the sequentially consistent view of the IP endpoint by the clients.

IPBus protocol extension

The extension IPBus system extension requires a corresponding extension in the IPBus protocol [IPBus].

There are two new transactions: lock and unlock

Lock request (Type ID = 0x10)

The lock transaction requests the exclusive access to the IP endpoint until the unlock transaction is sent. The owner of the lock is identified by the source IP address and port, and for the client ID word. The client ID should be the process thread id or an equivalent unique identifier in a per process basis.
The lock request is also released automatically if the owner of the lock (identified by IP address, port, and client ID) does not does not send (or receive) additional requests (or responses) for a given time period.
The lock automatic release timeout period should be smaller than the network timeout.

31	**24**	**23**	**16**	**15**	**8**	**7**	**0**
Ver. = 4	Words=2			Transaction ID		Type=0x10	Inf.=0xf
CLIENT ID							
TIMEOUT_MS							

Lock response

31	**24**	**23**	**16**	**15**	**8**	**7**	**0**
Ver. = 4	Words=0			Transaction ID		Type=0x10	Inf.=0x0

Unlock request (Type ID = 0x11)

31	**24**	**23**	**16**	**15**	**8**	**7**	**0**
Ver. = 4	Words=1			Transaction ID		Type=0x11	Inf.=0xf
CLIENT ID							

Unlock response

31	**24**	**23**	**16**	**15**	**8**	**7**	**0**
Ver. = 4	Words=0			Transaction ID		Type=0x11	Inf.=0x0

The protocol requires **three new error responses** (i.e. codes 0x6-0x8):

Info Code	Direction	Meaning
0x6	Response	Timeout on read, write, or lock due to lock not being released
0x7	Response	Can not unlock due to the lock has been released automatically

UHAL API extension

The proposed uHAL API extension provide a HwInterface::lock, HwInterface::unlock methods, and a ScopedLock class. The ScopedLock class is meant to assure the exception safety of the client code. This is, the ScopedLockproposed API uses RAII (Resource Acquisition Is Initialization) for the lock and unlock transaction pairs.

The proposed API is independent of the synchronization primitive/s finally used in the IPBus system.

Use case 1: Lock a sequence to a an IP endpoint:

```
ConnectionManager m(“connections.xml”);
HwInterface hw = m.getDevice(“hcal.crate1.board1”);

hw.lock();

ValWord<uint32_t> x = hw.getNode(“REG1”).read();
//...
ValWord<uint32_t> y = hw.getNode(“REG2”).read();

hw.unlock();

hw.dispatch();
```

Where the hw.dispatch() will hang until the the lock is released or a network timeout exception is raised.

Notice that we do not need to set a timeout, because the lock/unlock pair will typically happen within a single IPBus UDP packet. In any case, it is advisable because the packet boundaries are defined at run time.

In any case, the lock is always eventually unlocked either by the client or by a timeout mechanism.

The scoped lock increases the execution time at most by 2 Round Trips (~400 us).

Use case 2: Lock a sequence to a an IP endpoint using a scoped lock:

```
ConnectionManager m(“connections.xml”);
HwInterface hw = m.getDevice(“hcal.crate1.board1”);

//When the object “scoped” is created, an implicit hw.lock() and
hw.dispatch are executed.
ScopedLock scoped(hw, 1000);

ValWord<uint32_t> x = hw.getNode(“REG1”).read();
//...
ValWord<uint32_t> y = hw.getNode(“REG2”).read();
hw.dispatch();

//When the object “scoped” goes out of scope, an implicit hw.unlock
followed by hw.dispatch ae executed.
```

Notice that in this case, the default timeout is overrriden to a 1000 ms value.

Use case 3: Lock a sequence of transactions for more than one IP endpoint using a scoped lock:

```
ConnectionManager m(“connections.xml”);
HwInterface hw1 = m.getDevice(“hcal.crate1.board1”);
HwInterface hw2 = m.getDevice(“hcal.crate1.board2”);

ScopedLock scoped1(hw1,1000);
ScopedLock scoped2(hw2,1000);

ValWord<uint32_t> x = hw1.getNode(“REG1”).read();
```

```
ValWord<uint32_t> y = hw2.getNode("REG2").read();

hw1.dispatch();
hw2.dispatch();
```

Notice that the lock of multiple devices could lead to a timeout exception due to a dead-lock if the order of locking is not identical in all the clients. This is considered a programming error.

The implicit hw.unlock() and hw.dispatch() due to the ScopedLock will happen in the reverse order than construction.

References

[IPBus] IPBus protocol,
https://svnweb.cern.ch/trac/cactus/export/156/trunk/doc/ipbus_protocol.pdf
[uHAL] See uHAL C++ API, https://svnweb.cern.ch/trac/cactus/browser/trunk/uhal