A short introduction to IPbus

董胜
s.dong@mails.ccnu.edu.cn
2020/7/13

IPbus: a flexible Ethernet-based control system

The IPbus suite of software and firmware implement a reliable high-performance control link for particle physics electronics, based on the IPbus protocol.

- IPbus firmware - A module that implements the IPbus protocol within end-user hardware.
- ControlHub - A software application that mediates simultaneous hardware access from multiple uHAL clients, and implements the IPbus reliability mechanism over UDP.
- uHAL - The Hardware Access Library (HAL) providing an end-user C++/Python API for IPbus reads, writes and RMW (read-modify-write) transactions.

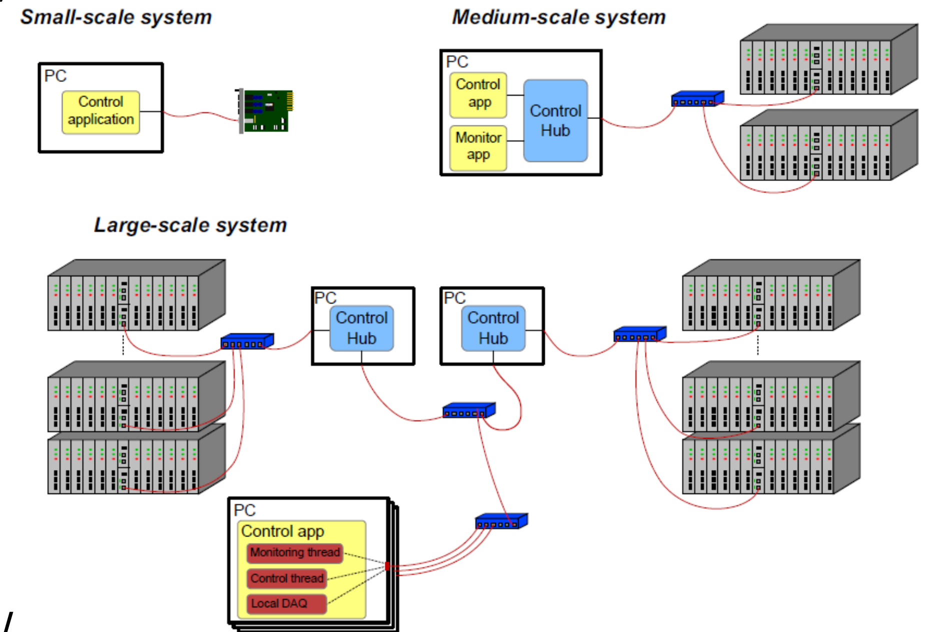
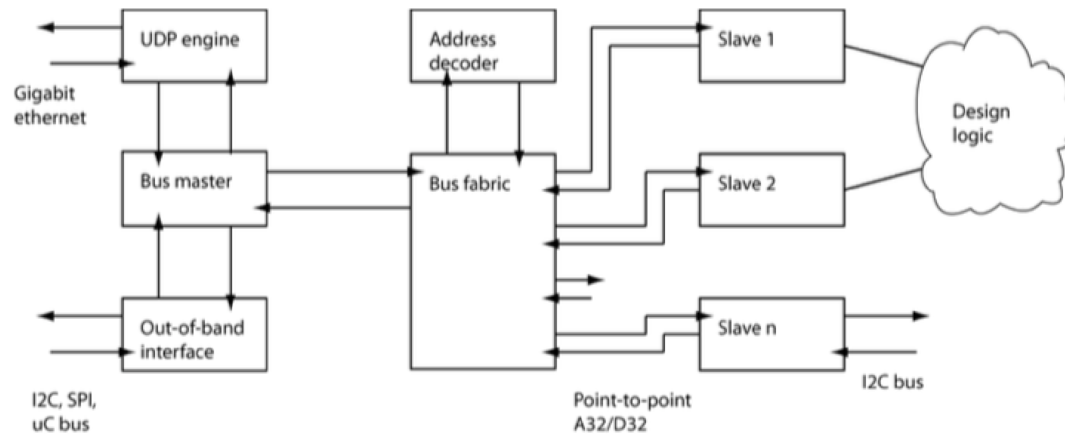


Figure 1. Example topologies of IPbus control systems involving μTCA hardware, from small to large scale.

- References:

  - 1. C. Ghabrous Larrea, K. Harder, D. Newbold, D. Sankey, A. Rose, A. Thea and T. Williams, "IPbus: a flexible Ethernet-based control system for xTCA hardware", JINST 10 (2015) no.02, C02019. DOI: 10.1088/1748-0221/10/02/C02019
  - 2. https://ipbus.web.cern.ch/ipbus/introduction/

Firmware

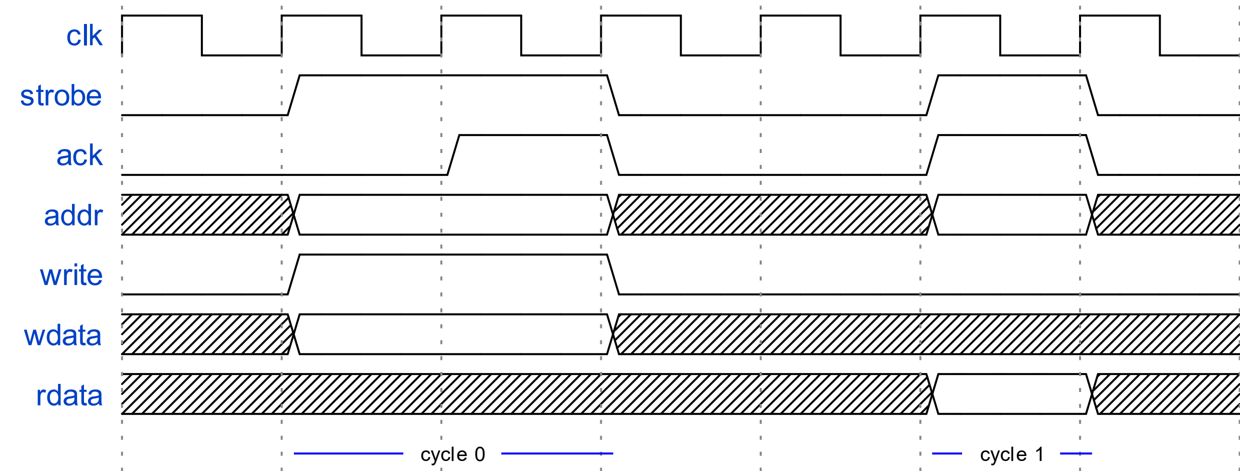


Firmware Structure

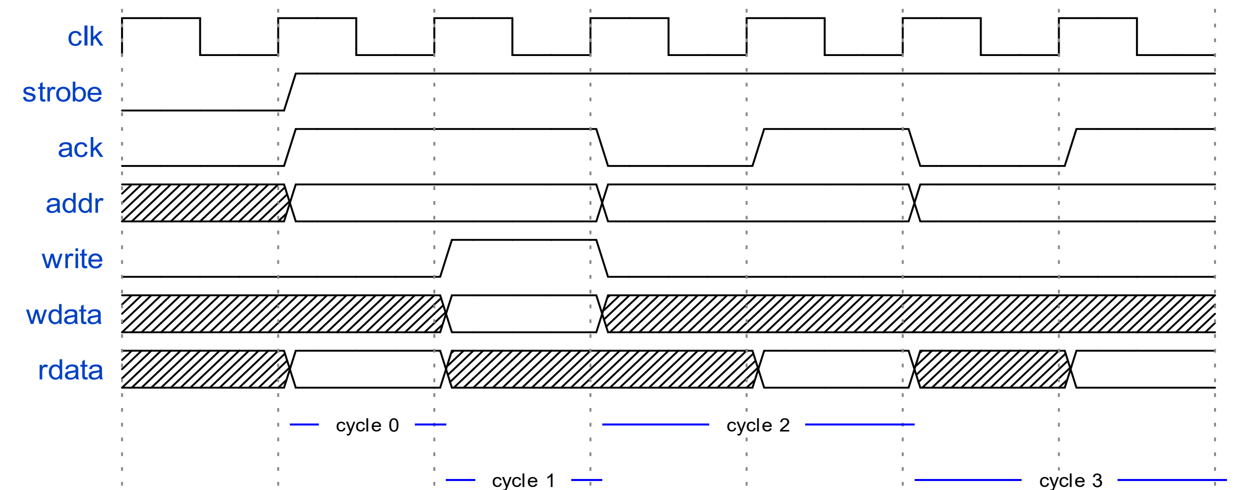
Signal	Direction	Width	Description
ipb_addr	Master to slave	32	Bus address
ipb_wdata	Master to slave	32	Data to be written to slave
ipb_write	Master to slave	1	Asserted for a write cycle, deasserted for read cycle
ipb_strobe	Master to slave	1	Qualifies address and data; assertion marks start of cycle
ipb_rdata	Slave to master	32	Data read from slave
ipb_ack	Slave to master	1	Acknowledge flag; assertion marks end of cycle
ipb_err	Slave to master	1	Bus error flag; assertion marks end of cycle

On-Chip Bus signals

Example timing diagrams



Example 1: Write transaction, then read transaction



Example 2: RMW transaction, then 2 read transactions

Slaves

>> The IPbus firmware repository, contains a library of generic slaves that have been used in a wide range of designs.

Registers

ipbus_ctrlreg_v	Bank of control / status registers of parameterized size (bus has only read access to 'status' register
ipbus_reg_v	Bank of registers of parameterized size (bus has read & write access to all of them)
ipbus_roreg_v	Block of read-only registers (i.e. values specified at build time, e.g. for version numbers)
ipbus_syncreg_v	Bank of control / status registers with clock-domain crossing

Counters

ipbus_ctrs_ported	Block of counters, accessed like ported RAM
ipbus_ctrs_samp	Interface to block of externally-provided counters (values of all counters sampled on same clock c
ipbus_ctrs_v	Block of counters

Slaves

RAMs

The depth of each RAM slave is parameterised through the generic `ADDR_WIDTH`. The 'ported' RAM slaves have zero wait states; all other RAM slaves have one wait state.

`ipbus_dpram`	32b (or less) wide dual-port memory with IPbus access on one side
`ipbus_dpram36`	36b wide dual-port memory with IPbus access on one side
`ipbus_ported_dpram`	
`ipbus_ported_dpram36`	
`ipbus_ported_dpram72`	
`ipbus_ported_sdpram72`	
`ipbus_sdpram72`	

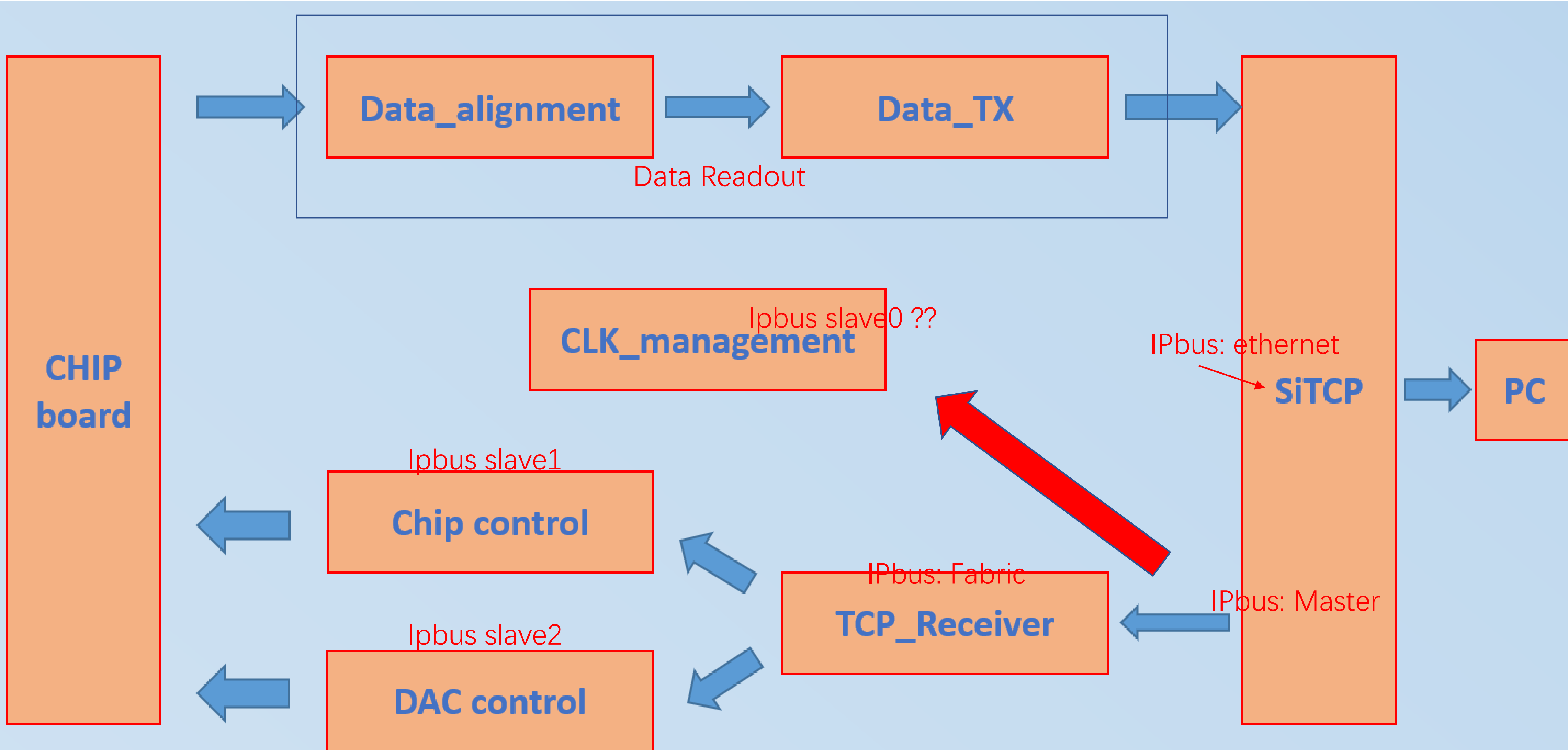
Miscellaneous

`ipbus_ctr`	Bus cycle counter (for debugging)
`ipbus_drp_bridge`	Interface to Xilinx DRP slave (for access to MGT, MAC, etc)
`ipbus_emac_hostbus`	
`ipbus_freq_ctr`	Generic clock frequency monitor
`uc_pipe_interface`	

We have also developed IPbus slaves that provide an interface to the Opencores SPI and I2C master cores - these can be found at:

- `components/opencores_spi/firmware/hdl/ipbus_spi.vhd`
- `components/opencores_i2c/firmware/hdl/ipbus_i2c_master.vhd`

Firmware structure – Based on IPbus



Software design using uHAL

- C++/Python API
- Address table(.xml file)

```python
#!/usr/bin/env python

#######################################################################

import sys
import uhal

#######################################################################

if __name__ == '__main__':

    if len(sys.argv) < 3:
        print "Please specify the device IP address" \
            " and the top-level address table file to use"
        sys.exit(1)

    device_ip = sys.argv[1]
    device_uri = "ipbusudp-2.0://" + device_ip + ":50001"
    address_table_name = sys.argv[2]
    address_table_uri = "file://" + address_table_name

    uhal.setLogLevelTo(uhal.LogLevel.WARNING)
    hw = uhal.getDevice("dummy", device_uri, address_table_uri)

    reg_name_base = "sysmon."
    regs = ["temp",
            "vccint",
            "vccaux",
            "vrefp",
            "vrefn",
            "vccbram"]
    max_len = max([len(i) for i in regs])

    print "IPBus SysMon/XADC demo:"
    for reg_name_spec in regs:
        reg_name = reg_name_base + reg_name_spec
        node = hw.getNode(reg_name)
        val_raw = node.read()
        hw.dispatch()

        # NOTE: Yes, flexible but ugly.
        val = val_raw.value()
        tags = hw.getNode(reg_name).getTags()
        magic = tags.split(";")[0]
        unit = tags.split(";")[1]
        exec(magic)
        msg_base = "  {1:{0:d}s}: {2:5.2f} {3:s}"
        print msg_base.format(max_len, reg_name_spec, conversion, unit)

#######################################################################
```

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>

<node>
    <node id="csr" address="0x0" description="ctrl/stat register" fwinfo="endpoint;width=1">
        <node id="ctrl" address="0x0">
            <node id="rst" mask="0x1"/>
            <node id="nuke" mask="0x2"/>
            <node id="led" mask="0x4"/>
        </node>
        <node id="stat" address="0x1"/>
    </node>
    <node id="reg" address="0x2" description="read-write register" fwinfo="endpoint;width=0"/>
    <node id="ram" address="0x1000" mode="block" size="0x400" description="1kword RAM" fwinfo="e
    <node id="pram" address="0x2000" description="1kword peephole RAM" fwinfo="endpoint;width=1"
        <node id="addr" address="0x0"/>
        <node id="data" mode="port" size="0x400" address="0x1"/>
    </node>
</node>
```