MODULAR FUNCTION UNITS FOR EFFICIENT BRIDGING OF HIGH SPEED NETWORKS AND CONTROL BUSSES

M.Weymann, L.Vivolo, F.H.Worm,
Creative Electronic Systems,
70 Route du Pont-Butin, CH 1213 Petit-Lancy 1/Geneva, Switzerland
email: ces@ces.ch, web: http://www.ces.ch

Abstract

CES VME PowerPC based boards can control up to 6 PCI Mezzanine Cards (PMCs), allowing to build compact and powerful systems for accelerator control as well as for aircraft test and simulation.

One or more of the PMCs attached to the VME board can be a Multifunction Computing Core (MFCC). It combines a PowerPC CPU with a user programmable I/O interface and can be used to increase the CPU power of the system.

The MFCC allows to scale CPU power as network and control interfaces are added to the system. The MFCC has already been used to control ATM, the same techniques can be used to add other I/O channels (e.g. WFIP) without additional load to the central CPU.

1 INTRODUCTION

Building a control or data acquisition system always involves integration of CPUs, busses, I/O interfaces as well as operating system and application software. The functionality of these systems must be maintained for more than a decade – much longer than the typical evolution cycle of modern microprocessors or operating systems. One successful approach to this challenge is to build a system out of *functional units* that interact with each other across well-defined interfaces.

2 FUNCTIONAL UNITS

Functional units are subsystems which include hardware, system software and application software with a well-defined interface to the rest of the system. They may evolve independently of other components in the system as long as this interface is maintained.

A crucial element in constructing these interfaces is the use of *standards.* Standard busses such as PCI, VME or CPCI, proven real time operating systems such as LynxOS, VxWorks or Chorus and standard I/O and network interfaces such as SCSI, ATM, Ethernet (10 Mbit/s – 1 Gbit/s), PCM, MIL 1553, WFIP and others.

2.1 Example: An Avionics Test System

As example of how a system of functional units may evolve, consider a test system for avionics equipment (IENA [1]). One subsystem reads data from PCM, ARINC 429 or MIL 1553, formats the data and delivers it via ATM to another subsystem that records the data via SCSI. Acquisition of each of the front-end busses can be considered as a functional unit of its own, collecting data and sending them to ATM would be another functional unit, ATM reception and recording still another one.

In IENA terminology, the first two types of functional units constitute level N3, the last type constitutes level N4.

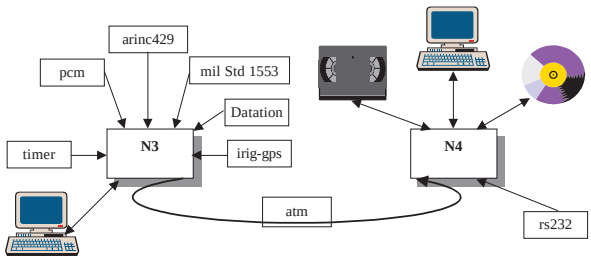

Figure 1: IENA. Avionics data acquisition system

Prototyping of the system started in 1994 (without ATM and recording) using MIPS R3052 based VME processor cards to implement the functional units for acquisition and data collection. A full prototype was installed in 1997, still keeping the MIPS based VME boards for acquisition, but using PowerPC based VME CPU boards together with ATM and SCSI PMCs for data collection and recording.

Evolving from this prototype a first implementation of the final system will start operation early 2000. A functional unit for acquisition will be a combination of a PowerPC based VME board (RIO2), a PMC carrier (PEB) and up to 4 MFCC [2] PMCs. The front-end FPGA of each MFCC is used to decode PCM data, which are then collected by an acquisition process running on the MFCCs CPU under LynxOS. For data collection and recording we

keep basically the same hardware configuration that has been used in the prototype.

In short, starting from the first prototype, the system evolved through several generations of microprocessors (R3052, PPC604@100MHz-PPC604r@300MHz) several generations of operating systems (TC/IX, EP/LX, LynxOS v2.3 – LynxOS v3.01). Each step came with considerable performance enhancements but kept the overall functionality of the previous generation.

2.2 Interfacing functional units

Two paradigms are widely used when interfacing multi-processor systems:

- Shared memory systems, relying on transparent, memory-mapped access with or without DMA.
- Network or message oriented systems.

Traditionally, shared memory systems present lower latencies and, if efficient DMA mechanisms are available, may use almost the complete bandwidth of the bus on which they are implemented. Message based solutions can cover a much wider range of hardware implementations, at the expense of a protocol overhead that increases latencies and limits the fraction of bandwidth available to transport data.

Given that CPU power has increased much faster than access to memory not to mention external busses, the overhead induced by CPU intensive protocols tends to become less important. On the other hand, it pays off to invest considerable CPU time in determining the most efficient way to transport data.

With the MFCC, CES has been faced with the challenge to provide a tool that allows to interface systems build from several high performance CPUs, coupled via PCI, PVIC [3], VME or CPCI. This tool should be easy to use, scalable – adding new CPUs should be possible without changing the structure of the application, and it should use the available hardware features in an optimal way.

3 MULTI-PROCESSOR SUPPORT

The MFCC does not have console or network interfaces of its own. It is coupled to its host only via its PCI bridge which renders its memory dual-ported between PCI and CPU and provides two sets of FIFOs and a DMA controller. It was clear that porting an operating system like LynxOS or VxWorks to the MFCC would imply an IP emulation across these resources.

Given the considerations above, we decided to aim at a tool with a much wider functionality. In the resulting system based on *channel* connections, IP emulation is just one particular application using the underlying *channels*. POSIX Semaphores and shared-memory mechanisms are other applications.

3.1 Channels

Channels are bi-directional links between two processes. A channel is established if two processes *connect* to the same *port* which may either be an explicit address (host number, CPU number) or a symbolic *label*. The number of open channels in a system is basically unlimited.

Once a channel is established, each process can write data to it and read data from it. The system guarantees that all data are delivered. Reading from a channel blocks until the number of bytes requested has been received or an error is detected (timeout or disconnect). Writing to a channel blocks until all data written are delivered to the destination or an error is detected.

A process may *listen* on a port to find out if another process is connecting to this port. It may also *select* among several channels the one who currently requires service.

A channel disappears when both processes *disconnect* from it Read or write operations on a channel will return an error if the connection has been broken (or not yet been established).

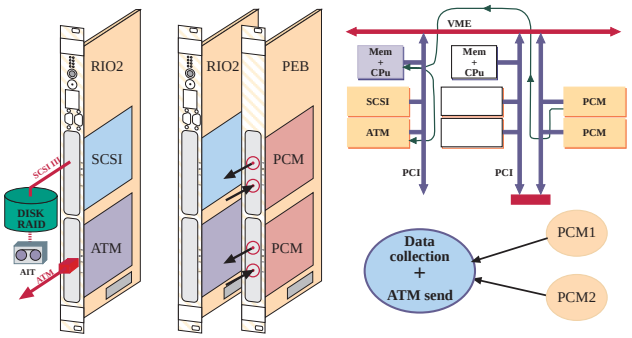

Figure 2: Channels connecting functional units

3.2 Applications

The IENA system cited above is one of the first applications using this channel architecture. Channels are used to link acquisition processes running on an MFCC to a data collection process running on another RIO2. Using *labels*, the data collection process could just as well be run on the MFCC itself or on the RIO2 carrying the MFCC.

Channel connections by *label* may also be used to build a dynamically re-configurable system. A server process that listens to connections on a given label may be shut down and moved to another processor. Client processes connected to it will be notified of the shutdown and can reconnect once the server is reconfigured.

3.3 Implementation

Channels are implemented using a combination of message and shared memory mechanisms. FIFOs (both on the RIO2 and on the MFCC) are used to notify the destination that a message arrived. The message body is

passed using either programmed I/O or DMA according to configuration choices and available hardware. Data are only copied once, there is no intermediate copy into driver or network buffers.

4 THE NEXT STEPS

4.1 MFCC based functional units

Using the flexibility offered by the MFCCs front-end FPGA, a whole family of high performance I/O interfaces will be implemented. Current developments include the PCM (IRIG 106-86) interface (538x) developed in collaboration with Aerospatiale and a MIL 1553 / STANAG 3910 interface designed at DASA. A functional unit consisting of a RIO2 and two MFCCs can handle one MIL 1553B (BC/mRT) and one high speed STANAG 3910 channel in a single VME slot.

Together with the PCM 5367, a functional unit is developed which is able distribute central timestamps with 1µs precision. The DAT 5384 PMC maintains a 48-bit time counter synchronized via a 1 Hz top with an external time reference (typically GPS). It synchronizes via a dual-redundant serial connection (P2 or front-panel) the corresponding counters in the FPGA of the acquisition PMCs. In this way data can be correlated system-wide to within 1µs.

4.2 Next generation MFCC

The existing MFCC 8441 (PPC603r@300MHz, 32-bit PPCbus) is being complemented with the MFCC 8442 (PPC750 or G4 @ >450Mhz, 64-bit PPC bus, L2cache). The new PMC is mainly intended to boost the CPU power of a system – it will be the first CES board carrying a G4 processor. Although the electrical adapter is no longer present on the MFCC 8442, it does include a front-end FPGA connected to the host boards I/O connector (P2 on RIO2).

4.3 Network interfaces

Several new ATM PMCs based on the IDT 77252 SAR chip will be available soon. They will allow for copper connections and multiple ATM channels per PMC. A functional unit combining an ATM PMC and an MFCC controlling it has been prototyped using a VxWorks ATM driver running on the MFCC. With this technique the I/O capacity of a system can be increased without additional load to the host (RIO2) processor. It could also be used to control a WFIP PMC (CEGELEC..[]) for which a LynxOS driver on the RIO2 is already operational.

A Gbit-Ethernet PMC is planned.

4.4 Next generation processors

With the RIO3 8064, an architecture that was stable over 5 years and a more than 10-fold increase in CPU power will change significantly.

The RIO3 will carry a PPC750 or G4 CPU at the maximum available frequency. The CPU-L2cache-memory complex will be directly accessible from two independent 64-bit PCI busses and from VME. The direct coupling of VME (on the RIO2 VME is coupled via PCI) will finally allow to reduce VME cycle latency considerably. The RIO3 will be fully VME 64x/LI compliant and will provide 2eSST and broadcast.

One of the two PCI busses is routed to P0. Together with a new PMC carrier board, the PEB 6416, this allows to control up to 10 PMCs from a single RIO3. Alternatively, this 'backplane PCI' bus can be used in parallel with VME to interconnect several RIO3 boards.

As the RIO2, the RIO3 will have a CPCI 'twin', the RIOC 4065.

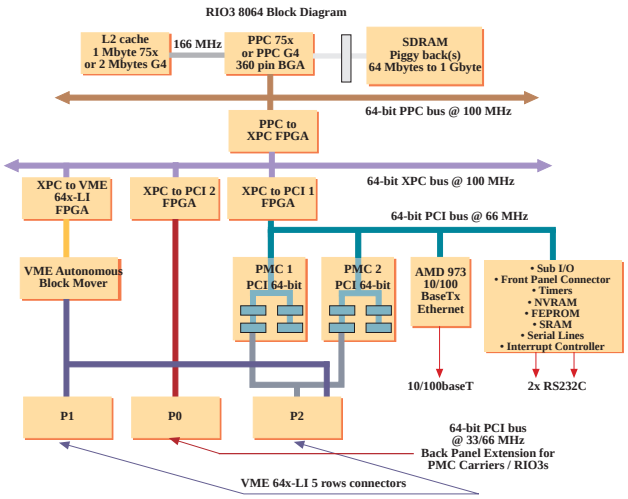

Figure 3: RIO3 8064 block diagram

5 CONCLUSION

Constructing data acquisition or control systems out of functional units whose hardware and software elements can evolve while the systems functionality is maintained has been proven to be a successful strategy. With its recently developed multi-processor support tool, CES has built another powerful tool which should help to rapidly convert the major technological innovations brought about by the next generation of hardware into improved the system performance on the application level.

REFERENCES

[1] F.H.Worm, J.P.Mao, "Modular Function Units for Flight Test an Validation Systems", ETTC99, Paris, June 1999.

[2] M.Weymann, "A PMC Based Computing Core", SYSCOMMS98, CERN, Geneva, March 1997.

[3] M.Weymann, "Linking PCI based processor platforms", CHEP97, Berlin, April 1997.