A Special-Purpose Processor System with Software-Defined Connectivity

Benjamin Miller, Sara Siegal, James Haupt, Huy Nguyen and Michael Vai

MIT Lincoln Laboratory

22 September 2009

This work is sponsored by the Navy under Air Force Contract FA8721-05-0002. Opinions, interpretations, conclusions and recommendations are those of the authors and are not necessarily endorsed by the United States Government.
Outline

• Introduction

• System Architecture

• Software Architecture

• Initial Results and Demonstration

• Ongoing Work/Summary
Why Software-Defined Connectivity?

• Modern ISR, COMM, EW systems need to be flexible
  – Change hardware and software in theatre as conditions change
  – Technological upgrade
  – Various form factors

• Want the system to be open
  – Underlying architecture specific enough to reduce redundant software development
  – General enough to be applied to a wide range of system components
    E.g., different vendors

• Example: Reactive electronic warfare (EW) system
  – Re-task components as environmental conditions change
  – Easily add and replace components as needed before and during mission
Special Purpose Processor (SPP) System

- System representative of advanced EW architectures
  - RF and programmable hardware, processors all connected through a switch fabric

Enabling technology: bare-bone, low-latency pub/sub middleware
Mode 1: Hardwired

- Hardware components physically connected
- Connections through backplane are fixed (no configuration management)
- No added latency but inflexible
Mode 2: Pub-Sub

Thin Communications Layer (TCL) Middleware

- Everything communicates through the middleware
  - Hardware components have on-board processors running proxy processes for data transfer
- Most flexible, but there will be overhead due to the middleware
Mode 3: Circuit Switching

- Configuration manager sets up all connections across the switch fabric
- May still be some co-located hardware, or some hardware that communicates via a processor through the middleware
- Overhead only incurred during configuration
- TCL middleware developed to support the SPP system
  - Essential foundation
- Resource Manager sets up (virtual) connections between processes
Outline

- Introduction
- System Architecture
- Software Architecture
- Initial Results and Demonstration
- Ongoing Work/Summary
System Configuration

- 3 COTS boards connected through VPX backplane
  - 1 Single-board computer, dual-core PowerPC 8641
  - 1 board with 2 Xilinx Virtex-5 FPGAs and a dual-core 8641
  - 1 board with 4 dual-core 8641s
  - Processors run VxWorks
- Boards come from same vendor, but have different board support packages (BSPs)
- Data transfer technology of choice: Serial RapidIO (sRIO)
  - Low latency important for our application
- Implement middleware in C++
System Model

Application Components

Signal Processing Lib

System Control Components

Vendor Specific

BSP1

BSP2

HW (Rx/Tx, ADC, etc.)

VxWorks

VPX + Serial Rapid I/O

Physical Interface

MIT Lincoln Laboratory
System Model

New SW components

Application Components

Application Components

System Control Components

Signal Processing Lib

TCL Middleware

BSP1

BSP2

VxWorks

VPX + Serial Rapid I/O

New HW components

Vendor Specific

New components can easily be added by complying with the middleware API

VPX + Serial Rapid I/O

VxWorks
Outline

• Introduction

• System Architecture

• **Software Architecture**

• Initial Results and Demonstration

• Ongoing Work/Summary
Publishing application doesn’t need to know where the data is going . . .

Middleware acts as interface to both application and hardware/OS

. . . and the subscribers are unconcerned about where their data comes from

OS
Proc. $k$

OS
Proc. $l_1$

OS
Proc. $l_2$

Switch Fabric

Publish/Subscribe Middleware

Process $k$

Process $l_1$

Process $l_2$

TCL Middleware

Topic T
Subscribers

send to

deliver

notify

send to

send to

OS
Proc. $l_1$
Abstract Interfaces to Middleware

- Middleware must be abstract to be effective
  - Middleware developers are unaware of hardware-specific libraries
  - Users have to implement functions that are specific to BSPs
Resource manager is currently in the form of an XML parser
- XML file defines topics, publishers, and subscribers
- Parser sets up the middleware and defines virtual network topology
Middleware Interfaces

- Base classes
  - `DataReader`, `DataReaderListener` and `DataWriter` interface with the application
  - `Builder` interfaces with BSPs
- Derive board- and communication-specific classes
Follows the Builder pattern in *Design Patterns*.*

Provides interface for sRIO-specific tasks
  - e.g., establish sRIO connections, execute data transfer

Certain functions are empty (not necessarily virtual) in the base class, then implemented in the derived class with BSP-specific libraries

Publishers and Subscribers

DataReaders, Data Writers and DataReaderListeners act as “Directors” of the Builder
- Tell the Builder what to do, Builder determines how to do it

DataWriter used for publishing, DataReader and DataReaderListener used by subscribers

Derived classes implement communication(sRIO)-specific, but not BSP-specific, functionality
- e.g., ring a recipient’s doorbell after transferring data

//member functions
virtual STATUS
DataWriter::write(message)=0;

virtual STATUS
SrioDataWriter::write(message)
{
  ...
  myBuilder->performDmaXfer(...);
  ...
}
Outline

- Introduction
- System Architecture
- Software Architecture
- Initial Results and Demonstration
- Ongoing Work/Summary
Software-Defined Connectivity
Initial Implementation

- Experiment: Process-to-process data transfer latency
  - Set up two topics
  - Processes use TCL to send data back and forth
  - Measure round trip time with and without middleware in place

```
<Topic>
  <Name>Send</Name>
  <ID>0</ID>
  <Sources>
    <Source>
      <SourceID>8</SourceID>
    </Source>
  </Sources>
  <Destinations>
    <Destination>
      <DSTID>0</DSTID>
    </Destination>
  </Destinations>
</Topic>

<Topic>
  <Name>SendBack</Name>
  <ID>1</ID>
  <Sources>
    <Source>
      <SourceID>0</SourceID>
    </Source>
  </Sources>
  <Destinations>
    <Destination>
      <DSTID>8</DSTID>
    </Destination>
  </Destinations>
</Topic>
```
Software-Defined Connectivity
Communication Latency

- One-way latency ~23 us for small packet sizes
- Latency grows proportionally to packet size for large packets

- Reach 95% efficiency at 64 KB
- Overhead is negligible for large packets, despite increasing size
Demo 1: System Reconfiguration

Objective: Demonstrate connectivity reconfiguration by simply replacing the configuration XML file.
Demo 2: Resource Management

TCL Middleware

Control  Receive  Proc #1  Proc #2  Transmit

Transmit  Proc #2

Receive  Proc #1

I've detected signals!

I will process this new information!

Low-latency predefined connections allow quick response
Demo 2: Resource Management

TCL Middleware

Control  Receive  Proc #1  Proc #2  Transmit

Resource manager sets up new connections on demand to efficiently utilize available computing power.

I need more help to analyze the signals!

I will determine what to transmit in response!

Working...

I will determine what to transmit in response!
Demo 2: Resource Management

TCL Middleware

Control  Receive  Proc #1  Proc #2  Transmit

I will transmit the response!

I have determined an appropriate response!

Proc #1/Transmit are publisher/subscriber on topic TransmitWaveform

Working...
Demo 2: Resource Management

TCL Middleware

Control | Receive | Proc #1 | Proc #2 | Transmit

I will transmit the response!

Transmit

Proc #2

I have determined an appropriate response!

Receive

Proc #1

Done!

After finishing, components may be re-assigned
Outline

• Introduction

• System Architecture

• Software Architecture

• Initial Results and Demonstration

• Ongoing Work/Summary
Ongoing Work

- Develop the middleware (configuration manager) to set up fixed connections
  - Mode 3: Objective system

- Automate resource management
  - Dynamically reconfigure system as needs change
  - Enable more efficient use of resources (load balancing)
Summary

• Developing software-defined connectivity of hardware and software components

• Enabling technology: low-latency pub/sub middleware
  – Abstract base classes manage connections between nodes
  – Application developer implements only system-specific send and receive code

• Encouraging initial results
  – At full sRIO data rate, overhead is negligible

• Working toward automated resource management for efficient allocation of processing capability, as well as automated setup of low-latency hardware connections