The Combined Array for Research in Millimeter-wave Astronomy (CARMA)

Presented by

David W. Hawkins
dwh@ovro.caltech.edu

California Institute of Technology
Owens Valley Radio Observatory
Tutorial coverage:

1. CARMA/SZA Arrays
2. Interferometry
3. CARMA and Linux
4. The Correlator System
5. Software development using the ACE C++ library
The CARMA Array
CARMA

The SZ Array
RF Signals (SZA)

Receiver

Block Downconverter

Downconverter Bank 1

Digitizers Bands 1 to 8

Downconverter Bank 2

Digitizers Bands 9 to 16

Sky spectrum

1GHz to 9GHz Receiver Spectrum

1GHz to 5GHz

500MHz to 1GHz

DC to 500MHz

To the correlation logic
The SZA Correlator System
CARMA downconverter
Noise source

Quadrature modulator
A COBRA Digitizer Board
A COBRA Correlator Board
Sideband separation and lobe rotation

\[ -f_{LO} \]

\[ \delta(f + f_{LO}) \exp(-j\phi) \]

\[ \delta(f - f_{LO}) \exp(j\phi) \]

\[ \text{sky spectrum} \]

\[ \text{LO spectrum} \]

\[ \exp(-j2\pi\tau) \]

\[ \exp\{-j2\pi(f + f_{LO})\tau - j\phi\} \]

\[ \exp\{-j2\pi(f - f_{LO})\tau + j\phi\} \]

positive frequency components

negative frequency components
Delay tracking
Cross-correlation
SiO Maser M82 Spectrum
The Whirlpool Galaxy (M51)
SZ Detection
CARMA and Linux

Control System

Antenna Control
- Frame grabber
- CAN control
- Motors
- Cryogenics
- Receivers
- Environment

Analog Control
- GPIB
- CAN Control
- Lobe rotator
- Noise source
- Downconverters

Digital Correlator
- Digitizers
- Correlators

Workstation-class
Quad Xeon 3.6GHz

Ethernet

PXE booted cPCI x86
1GHz CPU/1GB RAM

XAC 16-bit 33MHz
microcontrollers
w/ custom RTOS

TI C31 32-bit
33MHz DSPs
w/ uCOS-II
Distributed real-time control

- Linux for high-level control
  - Ethernet, PCI, CAN control
  - Nice development environment
  - Resource heavy (GHz clocks, GB memory)

- Microcontrollers and DSPs for hard real-time tasks
  - Single control interface; CAN or PCI
  - 100s of devices operating in parallel
  - Lightweight (MHz clocks, kB memory)
Correlator board block diagram

- XDS-510 Emulator Header
- DSP
- DSP Flash Ram
- DSP SRAM
- SDRAM
- Clock Generator
- Phase Detector, Voltage, and Temperature Monitor
- System Controller
- System Ctrl and PCI Configuration EPROMs
- Test Headers
- PCI Interface
- Power Distribution and Decoupling
- Power LEDs
- 40-Bit User Bus and Ref. Clock
- 32-Bit PCI
- PCI Power
- FPGA Internal Power
- LVDS Transceiver Interface
- Altera JTAG Chain (14 Devices) and Cypress JTAG Header (1 Device)
- Data/Address Bus Buffers
- LVDS Transmit Clocks
- FPGA/Transmit Clocks
System controller block diagram

Four arbitrated resources:
- PLX Bus devices
- DSP Bus devices
- SDRAM
- Peripherals (system controller devices)

Three resource masters:
- PCI read/write
- DSP read/write
- Passive serial (P/S) controller reads

SDRAM Controller

System Controller Internal Peripherals
- Board control registers
- Scratch RAM
- Correlation controller
- Passive serial controller
- 1-Wire controller
- Digitizer Module Controller

System Controller External Peripherals
- 1-Wire devices
- LVDS transceiver control
- JTAG control
- Digitizer Module

SDRAM Memory

SDRAM Bus Address/Data Control signals

Address/Data Control signals

PCI Bus

PLX-9054 PCI Controller

Address/Data Control signals

PLX Bus

Address/Data Control signals

M

DSP Bus Master/Target Interface

M

DSP Bus Master/Target Interface

DSP Bus Control Signals

DSP Bus Device Control Signals

DSP Bus Address/Data

TMS320LC31 DSP

PCI

R/W

DSP

R/W

P/S

R

DSP

R/W

PCI

R/W

PCI

R/W

DSP

R/W

Address/Data Control signals

Four arbitrated resources:
- PLX Bus devices
- DSP Bus devices
- SDRAM
- Peripherals (system controller devices)

Three resource masters:
- PCI read/write
- DSP read/write
- Passive serial (P/S) controller reads

SDRAM Controller

System Controller Internal Peripherals
- Board control registers
- Scratch RAM
- Correlation controller
- Passive serial controller
- 1-Wire controller
- Digitizer Module Controller

System Controller External Peripherals
- 1-Wire devices
- LVDS transceiver control
- JTAG control
- Digitizer Module
Correlator FPGA block diagram

Telescope Data Inter- and Intra-FPGA Data Pipeline
pipeline.vhd

Test Pattern Generator and Input Data Multiplexer.
inputmux.vhd

Correlator Lags
lags.vhd

Correlation (Lag Dump) Controller and Correlator FPGA Bus Interface
correlator.vhd

FPGA RAM
ram.asp.vhd

Correlator FPGA Bus to DSP/PCI
Cross-correlation calculation

\[ x(t_k) \quad y(t_k) \]

\[ x(t) \uparrow \quad y(t) \uparrow \]

negative lags

\[ -6 \rightarrow -5 \rightarrow -4 \rightarrow -3 \rightarrow -2 \rightarrow -1 \]

positive lags

\[ 0 \rightarrow 1 \rightarrow 2 \rightarrow \ldots \]
DSP real-time processing

- Read FPGA lags
- Normalization/Float conversion
- FFT
- Walsh demodulation
- Delay correction (sub-ns)
- Averaging

- IEEE floating-point conversion
- Transfer to SDRAM
- Transfer to Host
The COBRA driver
The SZA Correlator Control System

- Control system is a Bands client
- Bands server is a Band client (connecting to 16 bands)
- Band server is a Board client (connecting to 7 boards)
- Each server component contains:
  - Control server
  - Data server
  - Monitor server
The CARMA Correlator Control System

- Control system manipulates 8 bands of remote objects

- CORBA-to-sockets conversion

- Band server is a Board client (connecting to 19 boards)

- Each server component contains:
  - Control server
  - Data server
  - Monitor server
Software development using the ACE C++ library:

1. ACE overview

2. ACE use in COBRA

3. ACE components
   3.1. Binary data and formatted data streams
   3.2. Reactor
   3.3. Message Queues
   3.4. Tasks
   3.5. Streams
   3.6. Acceptor/Connector
ACE References:

- ACE: ADAPTIVE Communication Environment
- ACE homepage
- Books
ACE Overview

- OS Adaption layer
  - shields ACE components from platform-specific APIs
  - small footprint POSIX-like API

- C++ wrappers
  - typesafe C++ interfaces
  - overhead minimized via inlines and the use of templates rather than inheritance

- Frameworks
  - design patterns and application building-blocks
ACE use in COBRA

General layout of the data and monitor server components
Binary data

- **Problem:**
  - Machine endian differences
  - Machine word-size differences
  - Machine floating-point format differences
  - Compiler byte packing differences
  - Byte-streams are not framed

- **Solution:**
  - Use a machine independent binary format
  - Add a layer of framing protocol
Machine independent binary formats

- Non-binary is not efficient
  - ASCII, HTML, XML

- XDR (external data representation)
  - Used by sunrpc, NFS
  - *.x description file auto-generates C code
  - big-endian format

- CDR (common data representation)
  - Used by CORBA
  - *.idl description file auto-generates code
  - supports both endian formats; "reader makes right"
Binary framing protocol

• **Encode: Endianness flag, length, data**

```c
// Encode user data
ACE_OutputCDR payload;
payload << user_data;

// Construct the header
ACE_OutputCDR header(ACE_CDR::MAX_ALIGNMENT + 8);
header << ACE_OutputCDR::
     from_boolean(ACE_CDR_BYTE_ORDER);
header << payload.total_length();

// Send the header+payload
ACE_Message_Block *mblk = header.begin()->duplicate();
mblk->cont(cdr.begin()->duplicate());
peer_.send_n(mblk, timeout);
mblk->release();
```
// Decode the header
ACE_Message_Block *mblk = new ACE_Message_Block(
    ACE_CDR::MAX_ALIGNMENT + 8);
ACE_CDR::mb_align(mblk);
peer_.recv_n(mblk->wr_ptr(), 8, timeout);
mblk->wr_ptr(8);
ACE_InputCDR header(mblk);
ACE_CDR::Boolean endian;
header >> ACE_InputCDR::to_boolean(endian);
header.reset_byte_order(endian);
ACE_CDR::Long length;
header >> length;
mblk->release();

// Decode the payload
mblk = new ACE_Message_Block(
    ACE_CDR::MAX_ALIGNMENT + length);
ACE_CDR::mb_align(mblk);
peer_.recv_n(mblk->wr_ptr(), length, timeout);
mblk->wr_ptr(length);
ACE_InputCDR payload(mblk);
payload.reset_byte_order(endian);
payload >> user_data;
mblk->release();
ACE Reactor

- Event-driven programming
- Common interface for handling;
  - File descriptors (files, sockets, devices)
  - Signals
  - Timers
  - Message queue events
  - Inter-thread events

- **ACE_Reactor**: event dispatch
  - POSIX/Unix/Linux: `select()`
  - Windows: `WaitForMultipleObjects()`

- **ACE_Event_Handler**: event handler
ACE

ACE_Event_Handler

class ACE_Event_Handler {
	public:

	...
	noctorial int handle_input(
		ACE_HANDLE fd = ACE_INVALID_HANDLE);
	noctorial int handle_output(
		ACE_HANDLE fd = ACE_INVALID_HANDLE);
	noctorial int handle_timeout(
		const ACE_Time_Value &current_time,
		const void *act = 0);
	noctorial int handle_signal(
		int signum, siginfo_t * = 0,
		ucontext_t * = 0);
	noctorial int handle_exception(
		ACE_HANDLE fd = ACE_INVALID_HANDLE);
	noctorial int handle_close(
		ACE_HANDLE handle,
		ACE_Reactor_Mask close_mask);

	...

};
ACE Reactor Example

int main(int argc, char *argv[])
{
    SignalHandler sh;
    ACE_Reactor *reactor = ACE_Reactor::instance();

    sh.reactor(reactor); // set the reactor

    // Register signals to respond to
    reactor->register_handler(SIGINT, &sh);
    reactor->register_handler(SIGUSR1, &sh);
    reactor->register_handler(SIGUSR2, &sh);

    // Run the event handling loop
    reactor->run_reactor_event_loop();

    sh.reactor(0); // clear the reactor
    return 0;
}
ACE Message Queue

- Thread-to-thread communications
ACE Tasks

- An object-oriented thread class
- Built-in ACE_Message_Queue
- Part of the ACE_Streams framework

```cpp
template <ACE_SYNCH_DECL>
class ACE_Task : public ACE_Task_Base
{
public:
...
  virtual int open(void *args = 0);
  virtual int close(u_long flags = 0);
  virtual int svc(void);
  virtual int put (ACE_Message_Block *,
                   ACE_Time_Value * = 0);
  ...
};
```
ACE Streams

- A container for tasks
- Automates the connection of task-to-task queues
- The abstraction helps in building reusable tasks
- Example; output-server component reuse
  - BandDataServer
  - BandMonitorServer
  - BandsDataServer
  - BandsMonitorServer
ACE Acceptor/Connector

- Abstraction of communication roles;
  - Passive connection establishment; accept()
  - Active connection establishment; connect()
  - Communication once a connection is established

- ACE classes
  - ACE_SOCK_Acceptor
  - ACE_SOCK_Connector, ACE_FILE_Connector
  - ACE_Svc_Handler<PEER_STREAM, SYNCH_STRATEGY>

- ACE templates
  - ACE_Acceptor<SVC_HANDLER, PEER_ACCEPTOR>
  - ACE_Connector<SVC_HANDLER, PEER_CONNECTOR>