A Many-Core Software Defined Solution for the Development and Deployment of Wireless Systems

SDR'11-WlInnComm
Wednesday, November 30, 2011
Session 4A – SDR Systems

John Irza
Solutions Architect
Coherent Logix, Inc.
Andover, Massachusetts
irza@coherentlogix.com

Bryan Schleck
Applications Engineer
Coherent Logix, Inc.
Austin, Texas
schleck@coherentlogix.com
About Coherent Logix

Coherent Logix, Incorporated
(Incorporated in 2002)

Headquarters:
1120 S. Capital of Texas Hwy
Building 3, Suite 310
Austin, Texas  78746

Maker of ultra low-power, extremely-high performance, C-programmable processors (HyperX™) and RF chipsets (rfX™) for the embedded systems market – enabling low-power, real-time software defined systems.
Customers and Markets

- Commercial Markets
- Mil / Aero Markets

- Wireless
- Image / Video
- Defense
- High-Rel / Rad-Tol
In a word…

COMPLEX
Modern Communications Systems

System-level Complexity
- Multi-carrier waveforms
- Spectral “conformance”
- Interoperability
- Network management
- Cognitive operation

Algorithm-level Complexity
- Channel coding
- Adaptive modulation
- Digital Pre-Distortion
- Interference cancellation
- MAC-PHY interaction

“How do we design a product that addresses these challenges in the most timely manner?”

“How do we verify our design before the product is assembled for the first time?”

“How do we deploy our design to meet SWaP constraints, possibly as a SoC?”
Traditional Design Flows and Development Cycles

Exploration and Science

Prototyping $\{\text{Languages: C, HDL} \newline \text{Targets: GPP, DSP, FPGA}\}$

Programmable Product $\{\text{Languages: C, asm, HDL, RTL} \newline \text{Targets: GPP, DSP, FPGA}\}$

Custom SoC $\{\text{Languages: C, asm, HDL, RTL} \newline \text{Targets: IP Cores: GPP, DSP, FPGA}\}$
Traditional Design Flows and Development Cycles

- **Exploration and Science**
  - Multiple languages
    - Multiple ways to represent a design

- **Prototyping**
  - Multiple targets
    - Multiple implementations of a design

- **Programmable Product**
  - Multiple teams
    - Multiple ways to introduce errors & make a product late

- **Custom SoC**

---

**Copyright © 2011 Coherent Logix, Incorporated. All Rights Reserved.**
Observation: engineers traditionally design to hardware targets

- This leads to a “brittle solution”
  - A change in design requires a change in hardware and vice-versa

- This results in a broken verification flow
  - Cannot connect system-level design verification to implementation-level verification. Multiple languages, multiple tools.

- The design focus is not on system-level solution
Integrated Design Flow and Development Cycle

• System solution is hardware independent
  - Processing capability is sufficient
  - I/O throughput is sufficient
  - SWaP requirements are met
  - Scalable solution

• One language throughout the design flow
  - Preserve the “software stack” as-is, from design exploration through deployment
  - Integrated verification flow
One Language

**Lingua Franca**: “…a language systematically used to make communication possible between people…”


Why?

- Portability
- Efficiency
- Existing libraries
- Legacy project code
- Etc.
Wish list for the ideal embedded computing target:

- As much as processing power as needed
- Scalable with little/no glue logic
- Reconfigurable architecture
- Lots of I/O options (both size & type)
- Easy (preferably “invisible”) to use
- Low power
- Low cost

Implies the following:

- Many-core computing fabric
- Programmable network/topology
- Tools. Tools. Tools!
HyperX

- 100 Core Processor
  - 10 x 10 processor fabric
- Software re-configurable network/topology
  - Memory-Network re-configurable on-the-fly
- 8 DDR2 memory channels
- 16 LVDS I/O channels
  - Zero glue logic for multi-chip scalability
- Integer and floating point processing
  - 8,16, extended precision integer, 32 bit floating point
- 25 GFLOPS 32-bit, 50 GMACs 16-bit

Ultra-low power

hx3100a - total chip power
  - 75 mW < P < 3.5 W
  - (13 Pico-Joules/Op)

hx3100bxx* - total chip power
  - 25 mW < P < 1.75 W
  - (7 Pico-Joules/Op)

* Sampling 2012 Q1/Q2
Demo: Scalable OFDM Reference Waveform Implementation
Radio and Waveform Development System (RWDS)

hxHADS

- Fully modular and customizable
- Plug-and-play capability with HyperX ISDE
- Clear path to form-factor product
- Supports
  - PCIeexpress I/O
  - Data conversion
  - RF/IF inputs

Data Conversion

PCleexpress

Copyright © 2011 Coherent Logix, Incorporated. All Rights Reserved.
OFDM System Model

**TX**
- Payload Mode Length
- Packet ID
- CRC-24 Generation
- R=1/3 Convolutional Encoder
- TX Buffer
- Scrambler
- QPSK Modulation
- IFFT
- Insert Cyclic Prefix

**RX**
- Descramble
- QPSK Demodulation
- FFT
- Remove Cyclic Prefix
- Retrieve Packet ID
- CRC-24 Check
- Viterbi Decoder
- RX Buffer

\[ S_n \rightarrow w_n \]
\[ r_n \]
User Transport

Payload (240 bits)

PID

CRC-24

Tail (6)

RB+3

RB+2

RB+1

RB-1

RB-2

RB-3

Segment

slot
time
Resource Block Structure

Reference Signals interspersed among Resource Elements
Subcarrier Mapping

Resource Blocks

Resource Block Structure

Cat

Map

Ifft

Replicate with increased bandwidth

Reference Signals

Sync/CE Symbol

Categorical Data

Reference

Maps

Ifft
## OFDM System Comparison

<table>
<thead>
<tr>
<th>Parameter</th>
<th>WiFi (802.11a/g)</th>
<th>Fixed WiMAX (802.16d)</th>
<th>Mobile WiMAX (802.16e)</th>
<th>LTE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DL Access</strong></td>
<td>OFDM</td>
<td>OFDM</td>
<td>S-OFDM</td>
<td>OFDM</td>
</tr>
<tr>
<td><strong>UL Access</strong></td>
<td>OFDM</td>
<td>OFDMA</td>
<td>S-OFDMA</td>
<td>SC-FDMA</td>
</tr>
<tr>
<td><strong>Bandwidth</strong></td>
<td>20</td>
<td>DL 1.75/3/3.5/5.5/7</td>
<td>DL 1.25/2.5/5/10/20</td>
<td>1.4/3/5/10/15/20 MHz</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UL 1.25/3.5/7/14/28</td>
<td>UL 1.25/5/10/20</td>
<td></td>
</tr>
<tr>
<td><strong>N_{FFT}</strong></td>
<td>64</td>
<td>DL 256</td>
<td>128/512/1024/2048</td>
<td>128/256/512/1024/1536/2048 pt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UL 2048</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Δf</strong></td>
<td>312.5</td>
<td>11.16</td>
<td>10.94</td>
<td>15 kHz</td>
</tr>
<tr>
<td><strong>T_{FFT}</strong></td>
<td>3.2</td>
<td>89.6</td>
<td>91.4</td>
<td>66.67 μs</td>
</tr>
<tr>
<td><strong>T_G</strong></td>
<td>$2^2$</td>
<td>$2^3$</td>
<td>$2^{[2:5]}$</td>
<td>$2^{[2, 3.83, 3.678]}$ %$T_{FFT}$</td>
</tr>
<tr>
<td><strong>T_S</strong></td>
<td>4</td>
<td>100.8</td>
<td>114.25, 102.83, 97.11, 94.26</td>
<td>83.27, 71.36, 71.88 μs</td>
</tr>
<tr>
<td><strong>M</strong></td>
<td>BPSK, QPSK, 16/64QAM</td>
<td>BPSK, QPSK, 16/64QAM</td>
<td>BPSK, QPSK, 16QAM, 64QAM (DL)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CC, BTC, CTC, LDPC</td>
<td>CC, BTC, CTC, LDPC</td>
<td>CC, CTC</td>
<td></td>
</tr>
<tr>
<td><strong>c</strong></td>
<td>Coding</td>
<td>RS-CC, BTC, CTC</td>
<td>CC, BTC, CTC, LDPC</td>
<td>CC, CTC</td>
</tr>
<tr>
<td><strong>r</strong></td>
<td>Data Rates</td>
<td>6-54</td>
<td>2-134.4</td>
<td>2-134.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DL 100</td>
<td>UL 50</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Copyright © 2011 Coherent Logix, Incorporated. All Rights Reserved.
## SC-OFDM Timing Related Parameters

<table>
<thead>
<tr>
<th>Nominal System BW</th>
<th>0.96</th>
<th>1.92</th>
<th>3.84</th>
<th>7.68</th>
<th>15.36</th>
<th>23.04</th>
<th>30.72 MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>Occupied System BW</td>
<td>0.80</td>
<td>1.58</td>
<td>3.14</td>
<td>6.26</td>
<td>12.50</td>
<td>18.74</td>
<td>24.98 MHz</td>
</tr>
<tr>
<td>FFT Size</td>
<td>64</td>
<td>128</td>
<td>256</td>
<td>512</td>
<td>1024</td>
<td>1536</td>
<td>2048 Samples</td>
</tr>
<tr>
<td>Data Subcarriers</td>
<td>48</td>
<td>96</td>
<td>192</td>
<td>384</td>
<td>768</td>
<td>1152</td>
<td>1536 kHz</td>
</tr>
<tr>
<td>Pilot Subcarriers</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>32</td>
<td>64</td>
<td>96</td>
<td>128 kHz</td>
</tr>
<tr>
<td>Occupied Subcarriers</td>
<td>53</td>
<td>105</td>
<td>209</td>
<td>417</td>
<td>833</td>
<td>1249</td>
<td>1665 kHz</td>
</tr>
<tr>
<td>Resource Blocks/slot</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>32</td>
<td>48</td>
<td>64 Samples</td>
</tr>
<tr>
<td>Data Symbols per RB</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>Training Symbols per RB</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Subcarrier Spacing</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15 kHz</td>
</tr>
<tr>
<td>Useful Period, $T_{FFT}$</td>
<td>66.67</td>
<td>66.67</td>
<td>66.67</td>
<td>66.67</td>
<td>66.67</td>
<td>66.67</td>
<td>66.67 μs</td>
</tr>
<tr>
<td>$T_{CP}$ extended</td>
<td>N_{FFT}/4</td>
<td>16</td>
<td>32</td>
<td>64</td>
<td>128</td>
<td>256</td>
<td>384</td>
</tr>
<tr>
<td>$T_{SYM}$ extended</td>
<td>16.67</td>
<td>83.33</td>
<td>83.33</td>
<td>83.33</td>
<td>83.33</td>
<td>83.33</td>
<td>83.33 μs</td>
</tr>
<tr>
<td>Block Period</td>
<td>583.33</td>
<td>583.33</td>
<td>583.33</td>
<td>583.33</td>
<td>583.33</td>
<td>583.33</td>
<td>583.33 μs</td>
</tr>
<tr>
<td>Modulation</td>
<td>QPSK</td>
<td>QPSK</td>
<td>QPSK</td>
<td>QPSK</td>
<td>QPSK</td>
<td>QPSK</td>
<td>QPSK</td>
</tr>
<tr>
<td>Tail bits</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>Code Rate</td>
<td>0.33</td>
<td>0.33</td>
<td>0.33</td>
<td>0.33</td>
<td>0.33</td>
<td>0.33</td>
<td>0.33</td>
</tr>
<tr>
<td>Coded Bits Per Block</td>
<td>576</td>
<td>1152</td>
<td>2304</td>
<td>4608</td>
<td>9216</td>
<td>13824</td>
<td>18432</td>
</tr>
<tr>
<td>Uncoded Bits + Tail Per Block</td>
<td>192</td>
<td>384</td>
<td>768</td>
<td>1536</td>
<td>3072</td>
<td>4608</td>
<td>6144</td>
</tr>
<tr>
<td>Uncoded Bits Per Block</td>
<td>186</td>
<td>378</td>
<td>762</td>
<td>1530</td>
<td>3066</td>
<td>4602</td>
<td>6138</td>
</tr>
<tr>
<td>Bitrate</td>
<td>0.32</td>
<td>0.65</td>
<td>1.31</td>
<td>2.62</td>
<td>5.26</td>
<td>7.89</td>
<td>10.52 Mb/s</td>
</tr>
<tr>
<td></td>
<td>39.86</td>
<td>81.00</td>
<td>163.29</td>
<td>327.86</td>
<td>657.00</td>
<td>986.14</td>
<td>1315.29 kBytes/s</td>
</tr>
</tbody>
</table>

**Notes:**
- $T_{CP}$ and $T_{SYM}$ are extended for longer block periods.
- Uncoded Bits + Tail Per Block includes the number of pilot symbols and tail bits.
- Bitrate is calculated based on the number of uncoded bits per block.
Spinning Box Demo on hxHADS – hx3100A

Motion JPEG image sequence sent over the SC-OFDM PHY transport:

- QAM with convolutional encoding
- BW scalability according to I/FFT dimension
- AWGN impairment configurable at runtime to enable swept error rate studies over a selected SNR range
- CRC protected packet transport mapped to a fixed Resource Block structure
- ACK|NACK signaled out of band
hxHADS Slot Configuration

Digital
I/Q

ACK/NACK

hxA3100A
i.MX

Cable

Ethernet Hub

Configuration and control via Ethernet

PC

hxA3100A
i.MX

Connect Time: 0:00:00
Loss of TCP packet sync

Payload
Messages: 941
Byte Rate: 31.00 MB/sec
Frame Rate: 6.1

Payload
Messages: 977
Byte Rate: 32.00 MB/sec
Frame Rate: 6.1
Errors: 0

Overlay
AWGN Impairment
OFDM RX PHY Resource Allocation
TX Input data process with CRC, ACK/NACK

```
133  /**********************************************************************
134  * TASK 1: Streams data in from parallel port (16-bit packed data), computes  
135  * and appends a 32-bit CRC word at the end of every resource block, then  
136  * forwards the resource block (with CRC) to Task 3 unpacked (one bit per word).  
137  * This process uses a double-buffering mechanism.  
138  **************************************************************************/
139
140  else if (CLX_RANK == 1)
141  {
142      //Configure parallel port for input
143      CLX_StartStreamingIn( ROUTE_PP_INPUT );
144
145      //Initialize route from Task 1 to Task 3
146      CLX_Initroute(111);
147
148      //Receive a resource block's worth of data from parallel port (non-blocking).
149      CLX_DRecv( &data_ping[0], (RB_WORDS-2), CLX_INT, ROUTE_PP_INPUT);
150
151      while (1)
152      {
153          int i;
154
155          // Receive PONG buffer in the background while processing PING buffer.
156          //******************************************************************************
157          CLX_Rwait(ROUTE_PP_INPUT);
158          CLX_DRecv( &data_pong[0], (RB_WORDS-2), CLX_INT, ROUTE_PP_INPUT);
159
160          // Compute CRC on resource block and append.
161          crc_reg = CRC_RESET_VALUE;
162          for (i = 0 ; i < (RB_WORDS-2) ; i++) {
163              doCRCTable( data_ping[i] );
164          }
165
166          data_ping[RB_WORDS-2] = (int)(crc_reg);
167          data_ping[RB_WORDS-1] = (int)(crc_reg >> 16);
168
169          // Unpack and send a resource block. One bit per word.
170          UnpackSend( &data_ping[0], RB_WORDS);
```
Double-buffered main processing loop

Process PING data while receiving next set of data (PONG) using DMA

Process PONG data while receiving next set of data (PING) using DMA

```
click blaming current::
while (1) {
  int i;
  // ******************************************************
  // Receive PONG buffer in the background while processing PING buffer.
  // ******************************************************
  CLX_Fsm (ROUTE_FP_INPUT);
  CLX_Fsm (&data_pong[i], (BU_WORDS-2), CLX_INT, ROUTE_FP_INPUT);
  // Compute CRC on resource block and append.
  crc_reg = CRC_RESET_VALUE;
  for (i = 0 ; i < (BU_WORDS-2) ; i++) {
    doCRCTable ( data_pong[i] );
  }
  data_pong [BU_WORDS - 2] = (int) (crc_reg );
  data_pong [BU_WORDS - 1] = (int) (crc_reg >> 16);
  // Unpack and send a resource block. One bit per word.
  Unpack_Send (&data_pong[0], BU_WORDS);
  // Receive ACK/NACK signal. Re-transmit if NACK. Send new BU if ACK.
  CLX_Fsm (crc_pass[0], 5, CLX_INT, ROUTE_ACK_NACK);
  while ( crc_pass[0] != 1 ) {
    Unpack_Send (&data_pong[0], BU_WORDS);
    CLX_Fsm (&crc_pass[0], 5, CLX_INT, ROUTE_ACK_NACK);
  }
  // ******************************************************
  // Receive PING buffer in the background while processing PONG buffer.
  // ******************************************************
  CLX_Fsm (ROUTE_FP_INPUT);
  CLX_Fsm (&data_ping[i], (BU_WORDS-2), CLX_INT, ROUTE_FP_INPUT);
  // Compute CRC on resource block and append.
  crc_reg = CRC_RESET_VALUE;
  for (i = 0 ; i < (BU_WORDS-2) ; i++) {
    doCRCTable ( data_ping[i] );
  }
  data_ping [BU_WORDS - 2] = (int) (crc_reg );
  data_ping [BU_WORDS - 1] = (int) (crc_reg >> 16);
  // Unpack and send a resource block. One bit per word.
  Unpack_Send (&data_ping[0], BU_WORDS);
  // Receive ACK/NACK signal. Re-transmit if NACK. Send new BU if ACK.
  CLX_Fsm (crc_pass[0], 5, CLX_INT, ROUTE_ACK_NACK);
  while ( crc_pass[0] != 1 ) {
    Unpack_Send (&data_ping[0], BU_WORDS);
    CLX_Fsm (&crc_pass[0], 5, CLX_INT, ROUTE_ACK_NACK);
  }
}
```
Cell Instantiation

Instantiate different cells, with an interface composed of communication routes

Cells can be instantiated multiple times for scalability and code re-use

Cells are self-contained hierarchical processes. Can be composed of any number of processing elements

```c
/* CELL INSTANTIATIONS */

if (CLX_RANK == CLX_INSTS) {
    // Transmitter
    encoder_tx: encoder( ROUTE_CONTROL_TO_ENCODER,
                        ROUTE_SOURCE_TO_ENCODER,
                        ROUTE_ENCODER_TO_MODULATOR);
    modulator_tx: modulator( ROUTE_CONTROL_TO_MODULATOR,
                             ROUTE_ENCODER_TO_MODULATOR,
                             ROUTE_MODULATOR_TO_IFFT_REAL,
                             ROUTE_MODULATOR_TO_IFFT_IMG);
    ifft_tx: ifft( ROUTE_CONTROL_TO_IFFT,
                  ROUTE_MODULATOR_TO_IFFT_REAL,
                  ROUTE_MODULATOR_TO_IFFT_IMG,
                  ROUTE_IFFT_REAL_TO_CHANNEL,
                  ROUTE_IFFT_IMG_TO_CHANNEL);
}
return 0;
```
Example cell: Modulator

Cell declaration with communication routes interface

Call to modulation processing subroutine

```c
/* Modulator cell */

clx_cell modulator( clx_input routeControl,
                    clx_input routeBitstreamIn,
                    clx_output routeDataOutputReal,
                    clx_output routeDataOutputImg )
{
    int i, carrierIndex=0, carrierStride, symbolCount=0;

    // Clear Symbol Data
    for( i = 0; i < 2048; i++ ) {
        dataReal[i] = dataImag[i] = 0;
    }

    // Receive and apply the control and configuration parameters (i.e. type of
    // modulation and symbol size).
    CLX_recv(&controlWord, routeControl);
    decodeControlWord();

    while(1) {

        // Receive a block of data bits
        CLX_recv(bitsIn, bitTransferSize, CLX_UNSIGNED, routeBitstreamIn);

        // Modulate the block of data bits
        modulate(carrierIndex);

        // Output the modulated symbols over two routes (I and Q) simultaneously.
        CLX_Send( dataReal, symbolSize, CLX_INT, routeDataOutputReal );
        CLX_Send( dataImag, symbolSize, CLX_INT, routeDataOutputImg );
        CLX_Swait( routeDataOutputReal ); CLX_Endroute( routeDataOutputReal );

        // Print statement (for debugging)
        clx_print_cycle("\t Modulated Symbol=", symbolCount++);
    }
}
```

---
Modulation processing subroutine

ANSI C version (hardware-independent)

HyperX Assembly version
Real-Time Analysis (RTA) Example Framework

- Configurable Data Source
- I/Q Display and AWGN Control
- Configurable Data Sink

![Diagram showing a block diagram of RTA example framework with nodes for video source, transmitted video display, received video display, scalable OFDM PHY (Tx) and (Rx), AWGN, constellation, and RWDS.]
Real-Time Analysis: BER / PER Characterization

RX Constellation Intensity Plot

BER PER

Link Performance

Data Servers

AWGN CHANNEL

TX

RX
Demo Summary

- High level representation of SC-OFDM model PHY
  - Implemented on a many-core low-power processor

- Tools solve the mystery of parallel programming for you
  - Tools extract parallelism from your code and map to processor fabric
  - Manual parallelization and processor mapping also allowed

- Real-time analysis and visualization with no performance penalty
  - Use many-core processor as a HW simulation accelerator
  - Finish BER tests faster (in minutes, not days/weeks)

- No change to source code moving from sim to deployment
  - Preserve verification flow
  - Eliminate introduction of errors
  - Get to market faster
Take Aways

- One code base: for both development and deployment
  - Preserving the software stack
- One hardware target: for simulation acceleration and deployment
  - Scales transparently to multi-chip implementations
- Faster design exploration
  - Characterize systems in minutes, not days/weeks
- Faster time to product
  - Eliminate multiple language & design flows
Thank you for attending!

More demos and Q&A in the Exhibit Hall at booth #10