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

SDR'11-WInnComm
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 (*rf*X™) for the embedded systems market – enabling low-power, real-time software defined systems.

Customers and Markets

- Wireless
- Image / Video
- Defense
- High-Rel / Rad-Tol

- Commercial Markets
- Mil / Aero Markets

Modern Communications Systems

In a word…

COMPLEX

Modern Communications Systems

Coherent Logix™

System-level Complexity

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

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

Algorithm-level Complexity

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

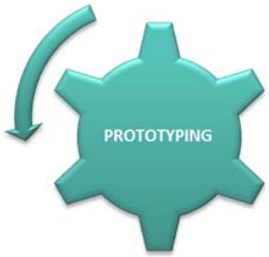
“How do we verify our design before the product is assembled for the first time?”

“How do we deploy our design to meet SWaP constraints, possibly as a SoC?”

Traditional Design Flows and Development Cycles

Coherent Logix™

Exploration and Science



Prototyping

Languages: C, HDL
Targets: GPP, DSP, FPGA

Programmable Product

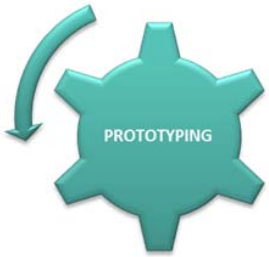
Languages: C, asm, HDL, RTL
Targets: GPP, DSP, FPGA

Custom SoC

Languages: C, asm, HDL, RTL
Targets: IP Cores: GPP, DSP, FPGA

Traditional Design Flows and Development Cycles

Exploration and Science



Prototyping

Programmable Product

Custom SoC

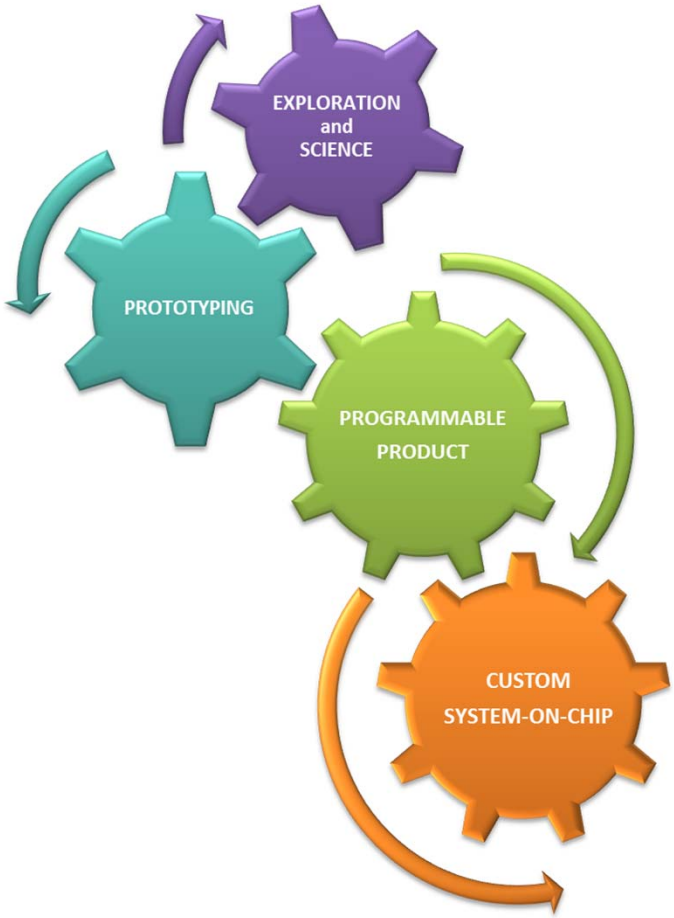
- Multiple languages
  - Multiple ways to represent a design
- Multiple targets
  - Multiple implementations of a design
- Multiple teams
  - Multiple ways to introduce errors & make a product late

Traditional Design Flows and Development Cycles

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…"*

Source: Wikipedia

"C"

Why?

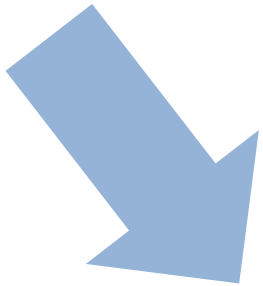
- Portability
- Efficiency
- Existing libraries
- Legacy project code
- Etc.

One Target

Coherent Logix™

Wish list for the ideal embedded computing target:

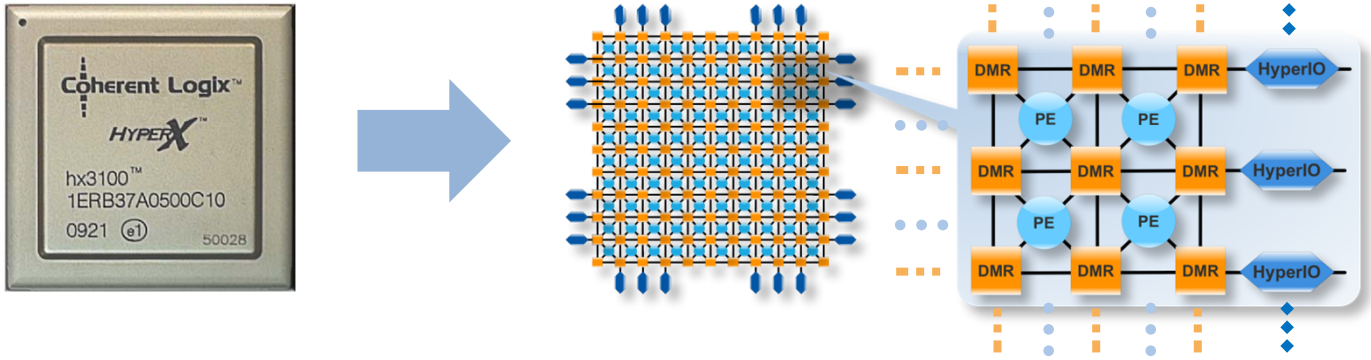
- As much as processing power as needed
- Scalable with little/no glue logic
- Reconfigurable architecture
- Lots of I/O options (both size & type)
- Easy (preferably “invisible”) to use
- Low power
- Low cost



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

Coherent Logix™

Demo: Scalable OFDM Reference Waveform Implementation

Radio and Waveform Development System (RWDS)

Coherent Logix™

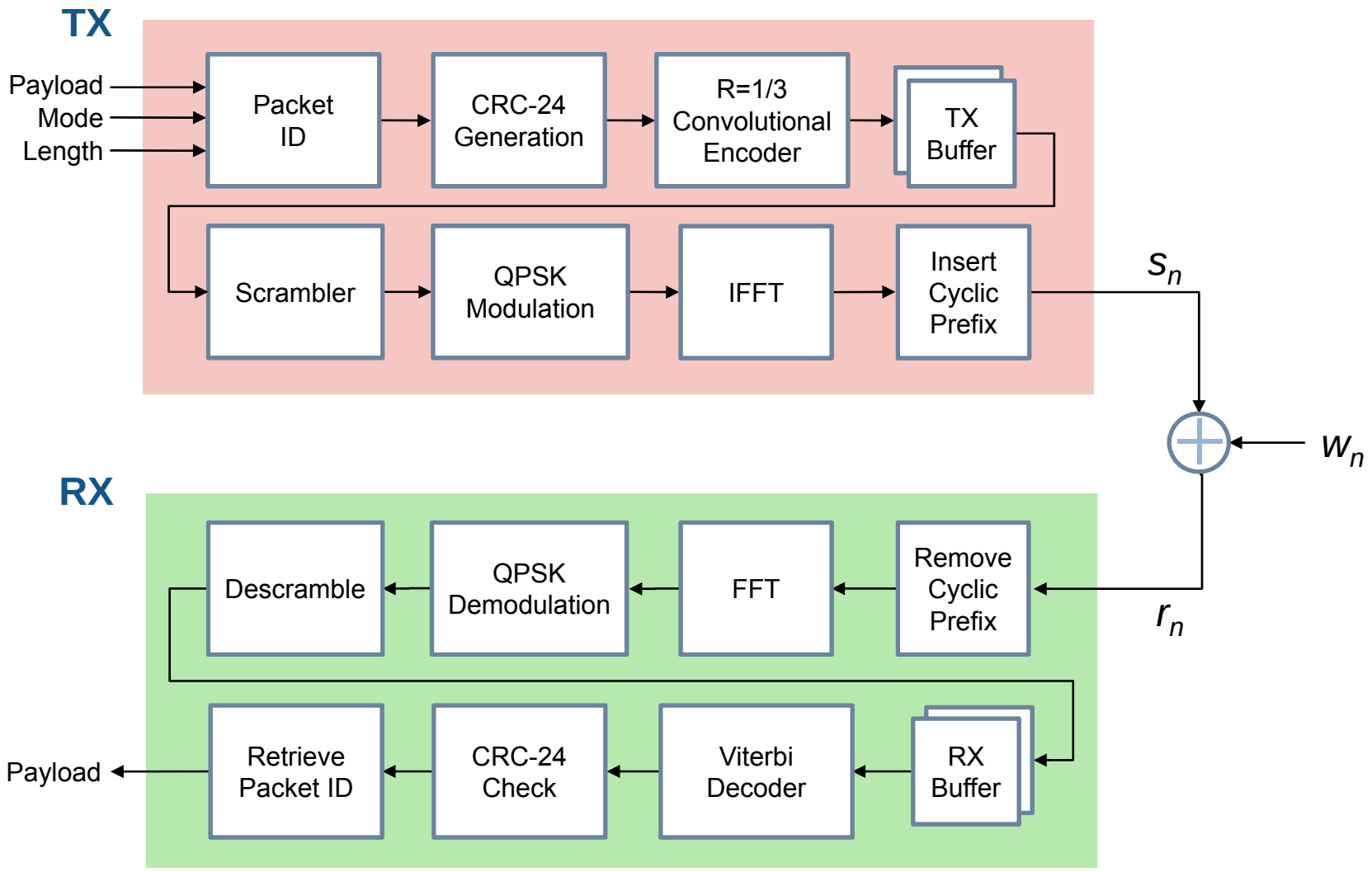
hxHADS

Data Conversion

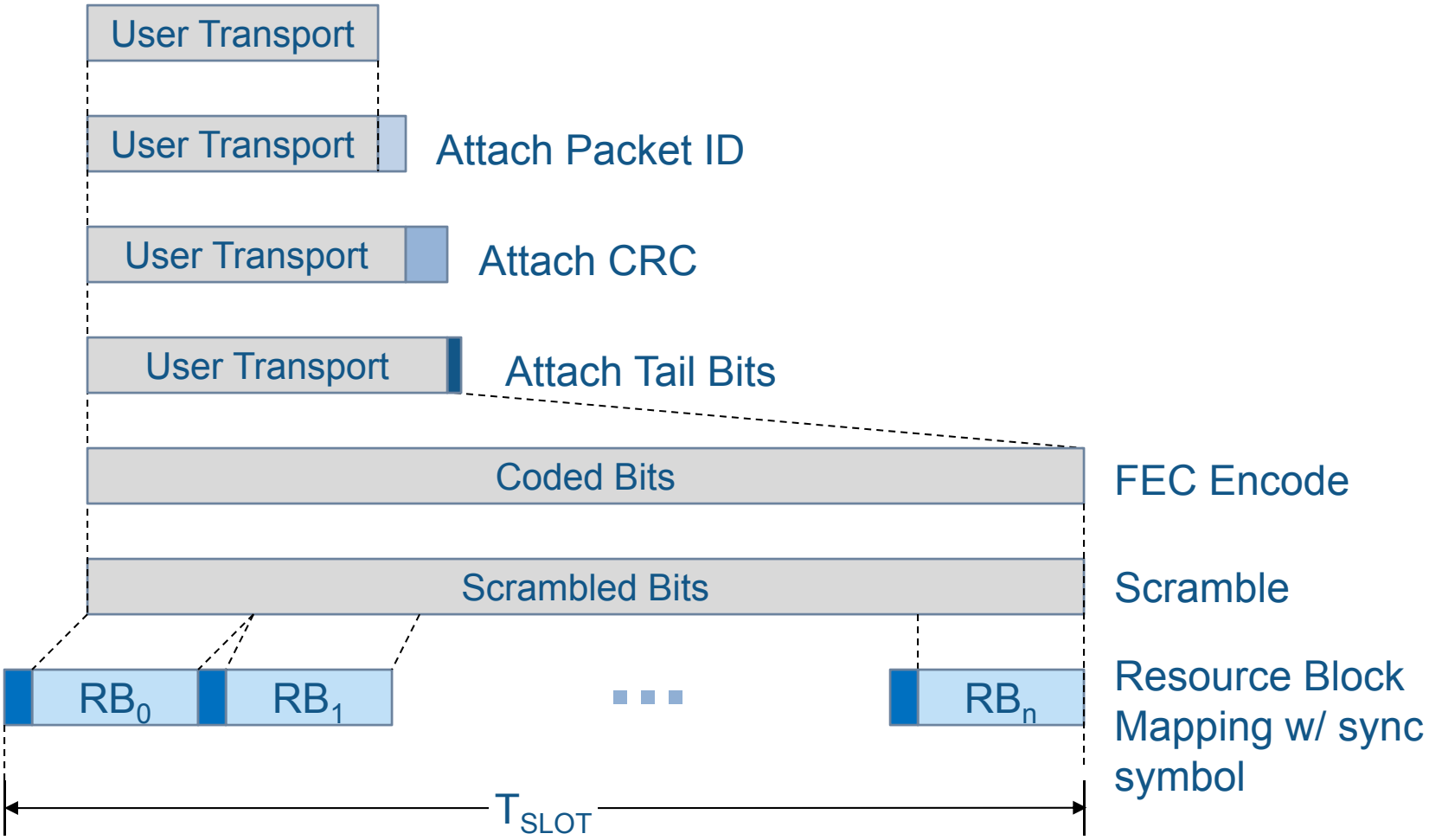
PCIexpress

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

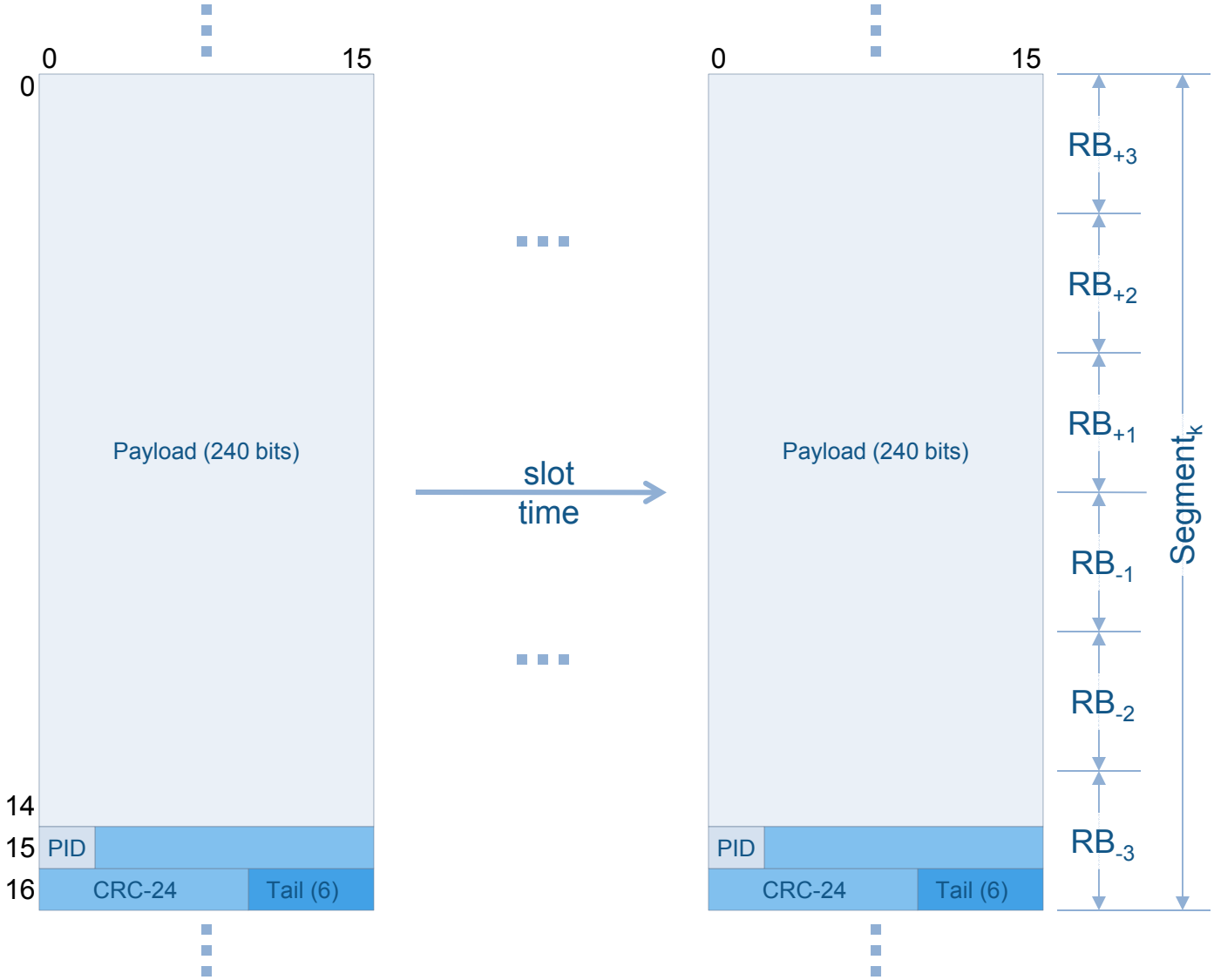
OFDM System Model



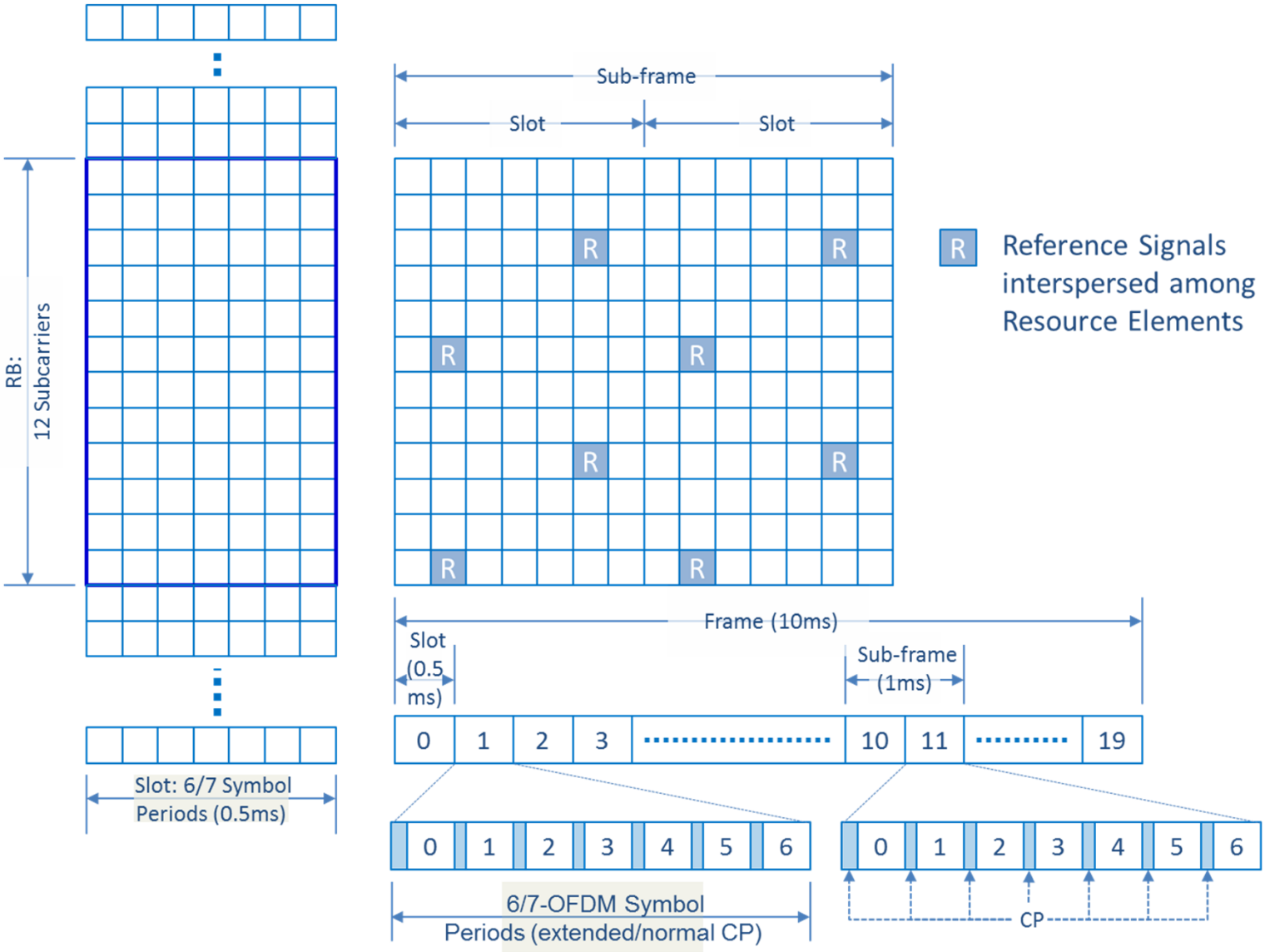
Frame Structure



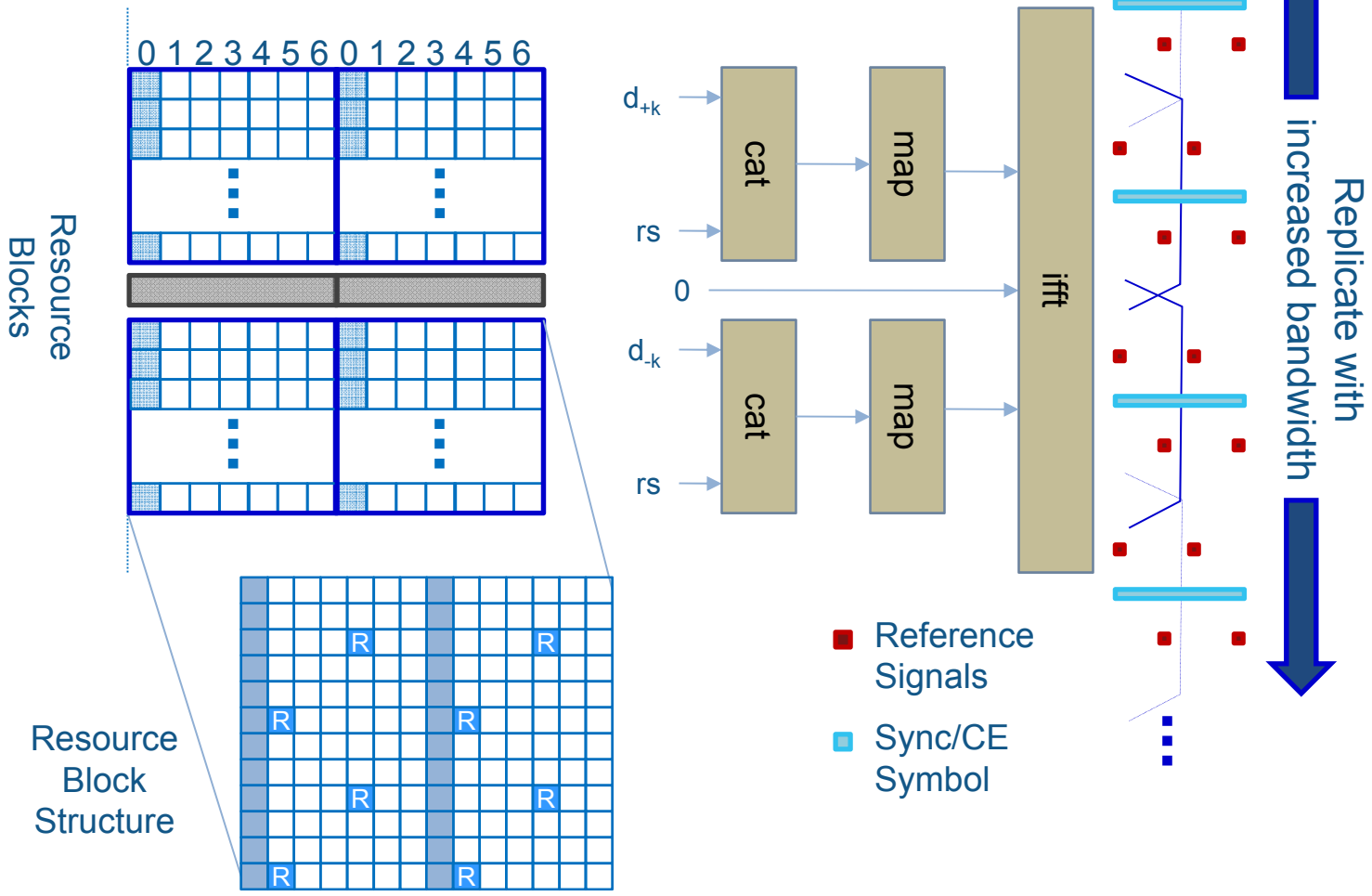
User Transport



Resource Block Structure



Subcarrier Mapping



OFDM System Comparison

Coherent Logix™

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

SC-OFDM Timing Related Parameters

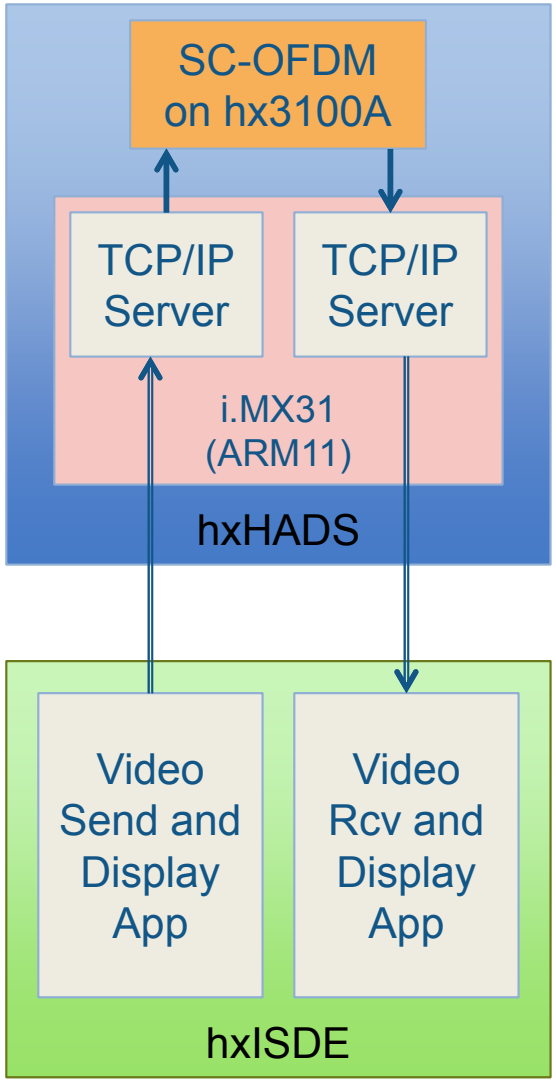
Coherent Logix™

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

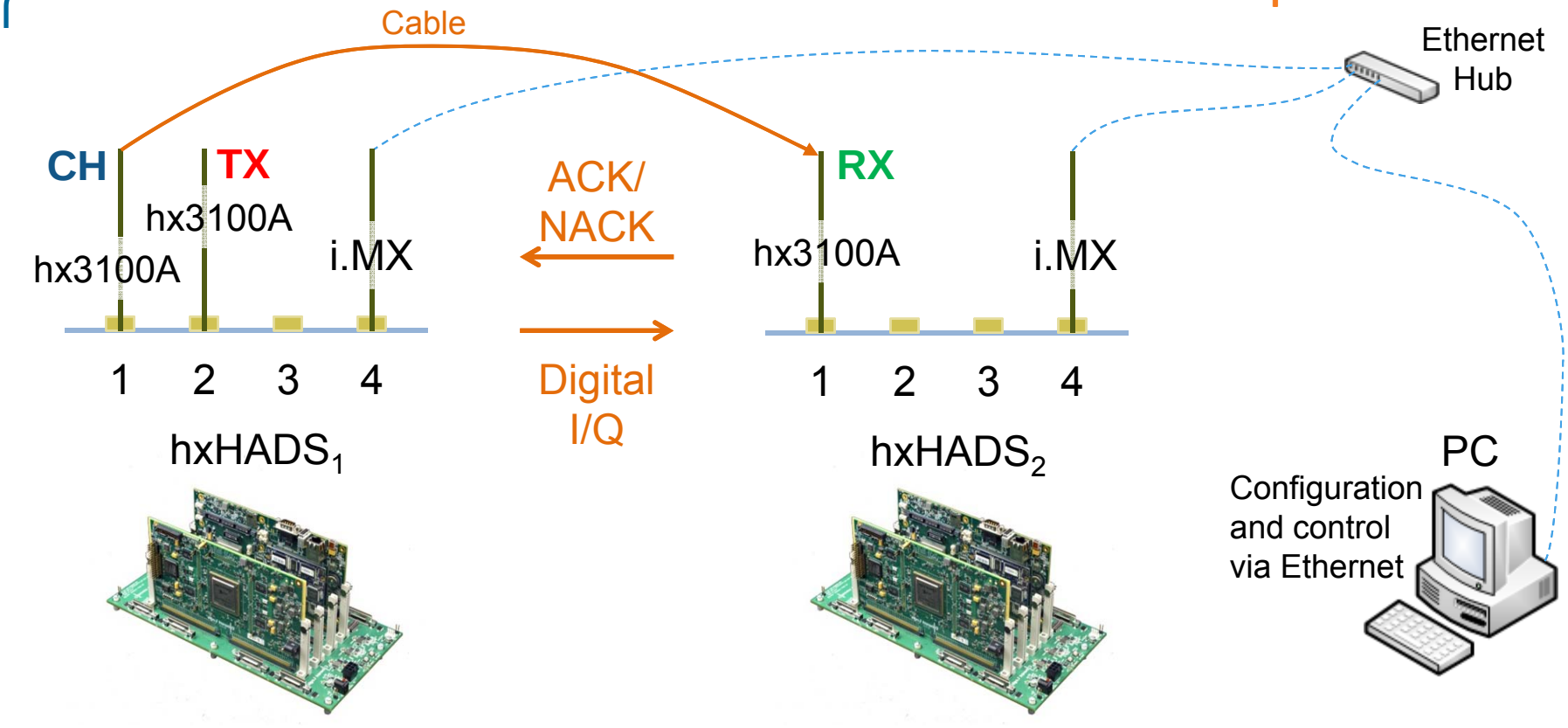
Spinning Box Demo on hxHADS – hx3100A

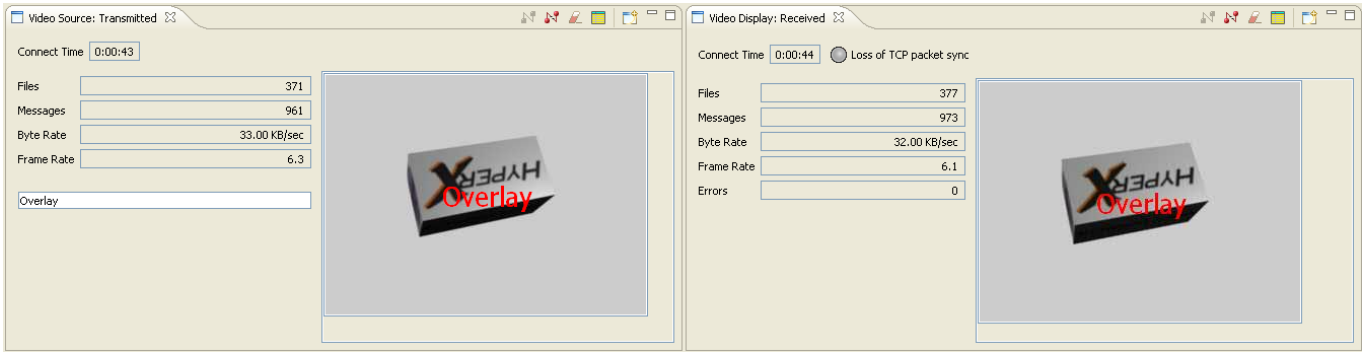
Motion JPEG image sequence sent over the SC-OFDM PHY transport:

- 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

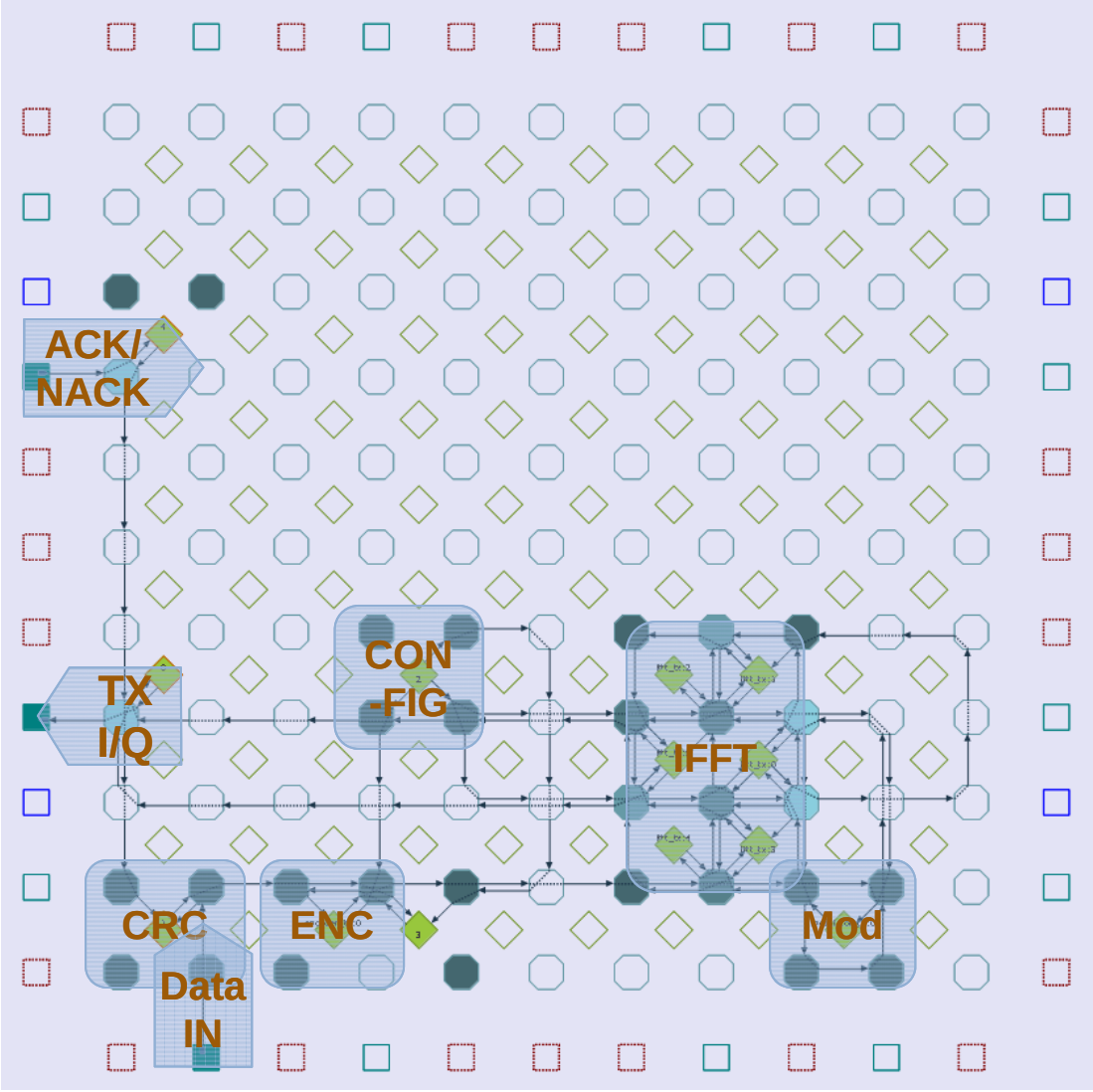


hxHADS Slot Configuration

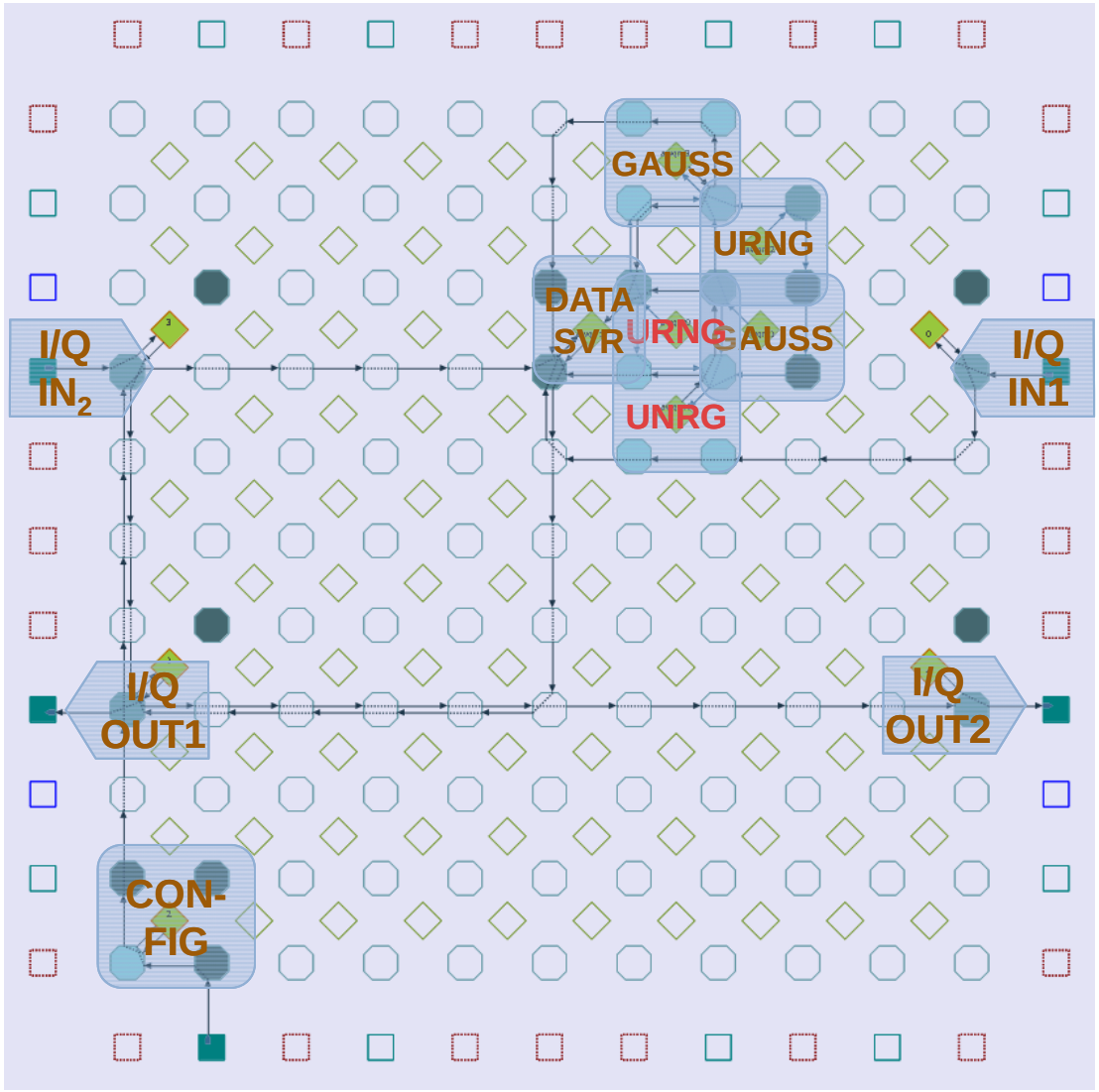




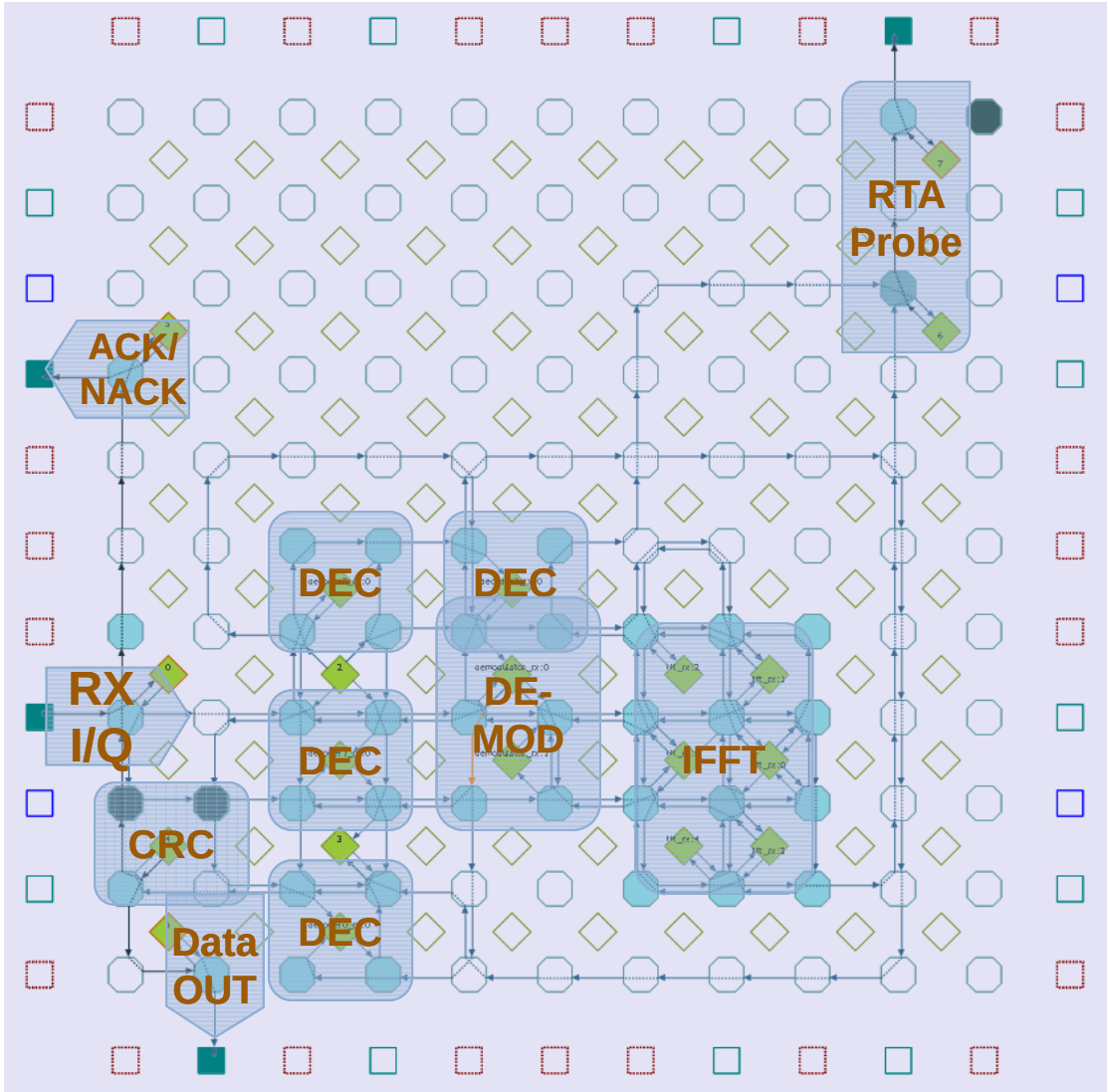
OFDM TX PHY Resource Allocation



AWGN Impairment



OFDM RX PHY Resource Allocation



TX Input data process with CRC, ACK/NACK

Coherent Logix™

Configure I/O

Data transfer (receive)

Data processing (CRC)

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

```
152     while (1)
153     {
154       int i;
155       // ********************************************************************
156       // Receive PONG buffer in the background while processing PING buffer.
157       // ********************************************************************
158       CLX_Rwait (ROUTE_PP_INPUT);
159       CLX_DIrecv (&data_pong[0], (RB_WORDS-2), CLX_INT, ROUTE_PP_INPUT);
160
161       // Compute CRC on resource block and append.
162       crc_reg = CRC_RESET_VALUE;
163       for (i = 0 ; i < (RB_WORDS-2) ; i++) {
164         doCRCTable ( data_ping[i] );
165       }
166
167       data_ping [RB_WORDS - 2] = (int) (crc_reg );
168       data_ping [RB_WORDS - 1] = (int) (crc_reg >> 16);
169
170       // Unpack and send a resource block. One bit per word.
171       Unpack_Send (&data_ping[0], RB_WORDS);
172
173       // Receive ACK/NACK signal. Re-transmit if NACK. Send new RB if ACK.
174       CLX_Drecv (&crc_pass[0], 5, CLX_INT, ROUTE_ACK_NACK);
175
176       while ( crc_pass[0] != 1 ) {
177         Unpack_Send (&data_ping[0], RB_WORDS);
178         CLX_Drecv (&crc_pass[0], 5, CLX_INT, ROUTE_ACK_NACK);
179       }
180
181       // ********************************************************************
182       // Receive PING buffer in the background while processing PONG buffer.
183       // ********************************************************************
184       CLX_Rwait (ROUTE_PP_INPUT);
185       CLX_DIrecv (&data_ping[0], (RB_WORDS-2), CLX_INT, ROUTE_PP_INPUT);
186
187       // Compute CRC on resource block and append.
188       crc_reg = CRC_RESET_VALUE;
189       for (i = 0 ; i < (RB_WORDS-2) ; i++) {
190         doCRCTable ( data_pong[i] );
191       }
192
193       data_pong [RB_WORDS - 2] = (int) (crc_reg);
194       data_pong [RB_WORDS - 1] = (int) (crc_reg >> 16);
195
196       // Unpack and send a resource block. One bit per word.
197       Unpack_Send (&data_pong[0], RB_WORDS);
198
199       // Receive ACK/NACK signal. Re-transmit if NACK. Send new RB if ACK.
200       CLX_Drecv (&crc_pass[0], 5, CLX_INT, ROUTE_ACK_NACK);
201       while ( crc_pass[0] != 1 ) {
202         Unpack_Send (&data_pong[0], RB_WORDS);
203         CLX_Drecv (&crc_pass[0], 5, CLX_INT, ROUTE_ACK_NACK);
```

Cell Instantiation

Coherent Logix™

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

```
303  /*********************************************************
304  * CELL INSTANTIATIONS
305  *********************************************************
306
307  if (CLX_RANK == CLX_INSTS) {
308
309    // Transmitter
310    encoder_tx:   encoder(  ROUTE_CONTROL_TO_ENCODER,
311        ROUTE_SOURCE_TO_ENCODER,
312        ROUTE_ENCODER_TO_MODULATOR);
313
314    modulator_tx: modulator(  ROUTE_CONTROL_TO_MODULATOR,
315        ROUTE_ENCODER_TO_MODULATOR,
316        ROUTE_MODULATOR_TO_IFFT_REAL,
317        ROUTE_MODULATOR_TO_IFFT_IMAG);
318
319    ifft_tx: ifft(  ROUTE_CONTROL_TO_IFFT,
320        ROUTE_MODULATOR_TO_IFFT_REAL,
321        ROUTE_MODULATOR_TO_IFFT_IMAG,
322        ROUTE_IFFT_REAL_TO_CHANNEL,
323        ROUTE_IFFT_IMAG_TO_CHANNEL);
324  }
325
326  return 0;
327 }
328
329
```

Example cell: Modulator

Coherent Logix™

Cell declaration with communication routes interface

Call to modulation processing subroutine

```
310 /*****************************************************************************
311  * Modulator cell
312  *****************************************************************************/
313
314 clx_cell modulator( clx_input  routeControl,
315                     clx_input  routeBitstreamIn,
316                     clx_output routeDataOutputReal,
317                     clx_output routeDataOutputImag )
318 {
319   int i, carrierIndex=0, carrierStride, symbolCount=0;
320
321   // Clear Symbol Data
322   for(i= 0; i < 2048; i++) {
323     dataReal[i] = dataImag[i] = 0;
324   }
325
326   // Receive and apply the control and configuration parameters (i.e. type of
327   // modulation and symbol size).
328   CLX_Qrecv(&controlWord, routeControl);
329   decodeControlWord();
330
331   while(1) {
332
333     // Receive a block of data bits
334     CLX_Drecv(bitsIn, bitTransferSize, CLX_UNSIGNED, routeBitstreamIn);
335
336     // Modulate the block of data bits
337     modulate(carrierIndex);
338
339     // Output the modulated symbols over two routes (I and Q) simultaneously.
340     CLX_DIsend( dataReal, symbolSize, CLX_INT, routeDataOutputReal);
341     CLX_Dsend(  dataImag, symbolSize, CLX_INT, routeDataOutputImag);
342     CLX_Swait(routeDataOutputReal); CLX_Endroute(routeDataOutputReal);
343
344     // Print statement (for debugging)
345     clx_print_cycle("%T Modulated Symbol=", symbolCount++);
346
347   } // while(1)
348 }   // end of modulator() cell
349
```

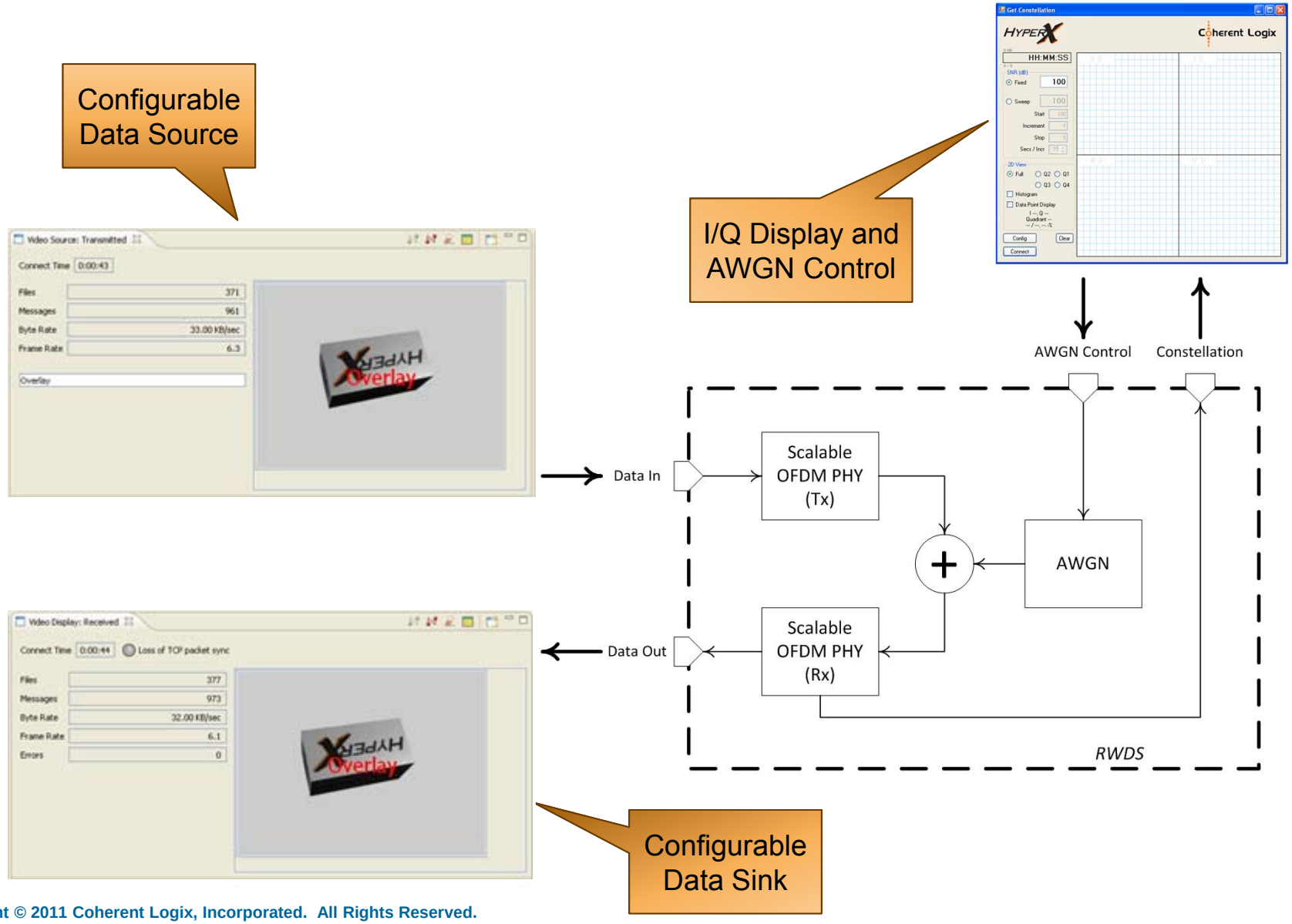
Modulation processing subroutine

ANSI C version (hardware-independent)

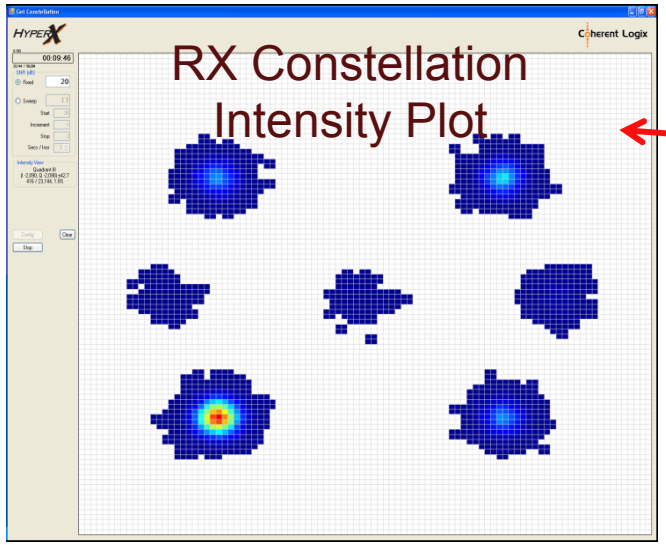
HyperX Assembly version

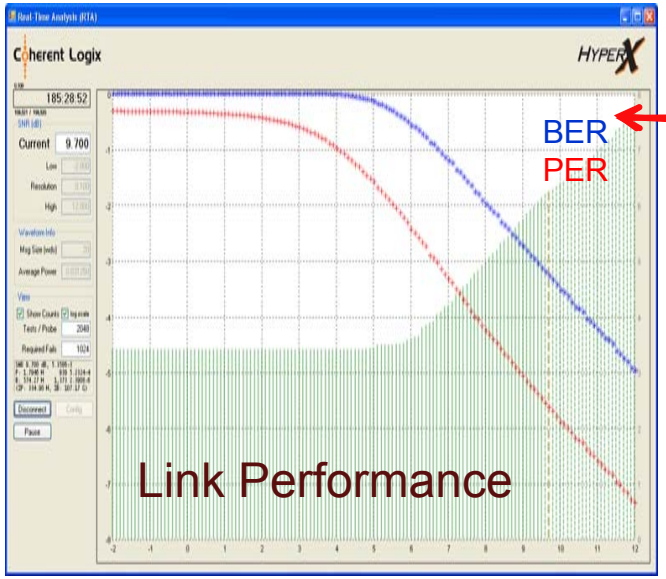
```
void modulate(int carrierIndex) {

 int i, k, currentCarrier,  bitCount, bitPtr, bitWordPtr;
 int currentBits ;

#ifdef CCODE

 // Populate Carriers with current Bits
 for (i = 0; i < bitTransferSize; i+=bitsPerCarrier, carrierIndex++) {
   currentCarrier = carrierLocations[carrierIndex] ;
   currentBits = 0;
   for (k = i; k < i+bitsPerCarrier; k++) {
     currentBits = (bitsIn[k] | currentBits) << 1;
   }
   currentBits >>= 1;
   dataReal[currentCarrier] = modulationR_lut[currentBits];
   dataImag[currentCarrier] = modulationI_lut[currentBits];
 }

 // Populate Pilots with Training
 for (i = 0; i < pilotsPerSymbol; i++) {
   currentCarrier = pilotLocations[i];
   dataReal[currentCarrier] = -1414;
   dataImag[currentCarrier] = 0;
 }

#else

 asm(
     "mov %0, %idx_stridex_i \n"              // bitsPerCarrier
     "mov %2, %idx_end_i \n"                  // bitTransferSize
     "sub %0, $1, %r2 \n"                     // bitsPerCarrier-1 (for loop)
     "mov %1, %rp0 \n"                        // dataR
     "mov %5, %rp1 \n"                        // dataI
     "repeat $0, %null, %null, %i,  MODULATE%= \n"

     " mov %3, %idx_current_j \n"             //  currentCarrier = carrierLocations[carrierIndex++]
     " mov %idx_current_i, %idx_current_k \n" // k = i
     " mov $0, %r0 \n"                        // currentBits = 0;
     " add %idx_current_i, %r2, %r1 \n"       // i+bitsPerCarrier-1
     " mov %r1, %idx_end_k \n"                // k < i+bitsPerCarrier-1
     " mov carrierLocations[%j], %idx_current_j \n"  //  currentCarrier = carrierLocations[carrierIndex++]

     " repeat1 %null, %null, $1, %k\n"        //for(k = i; k < i+bitsPerCarrier; k++){
     "   or<<1 bitsIn[%k], %r0, %r0\n"        // currentBits = (bitsIn[k] | currentBits) << 1;

     " shra %r0, $1, %r0\n"                   // currentBits >>= 1;
     " mov %r0, %idx_current_k\n"             // currentBits >>= 1;
     " add %3, $1, %3\n"                      // carrierIndex++

     " mov modulationR_lut[%k],  [%rp0+%j]\n" // data[currentCarrier  ] = modulationR_lut[currentBits];
     "MODULATE%=:\n"
```

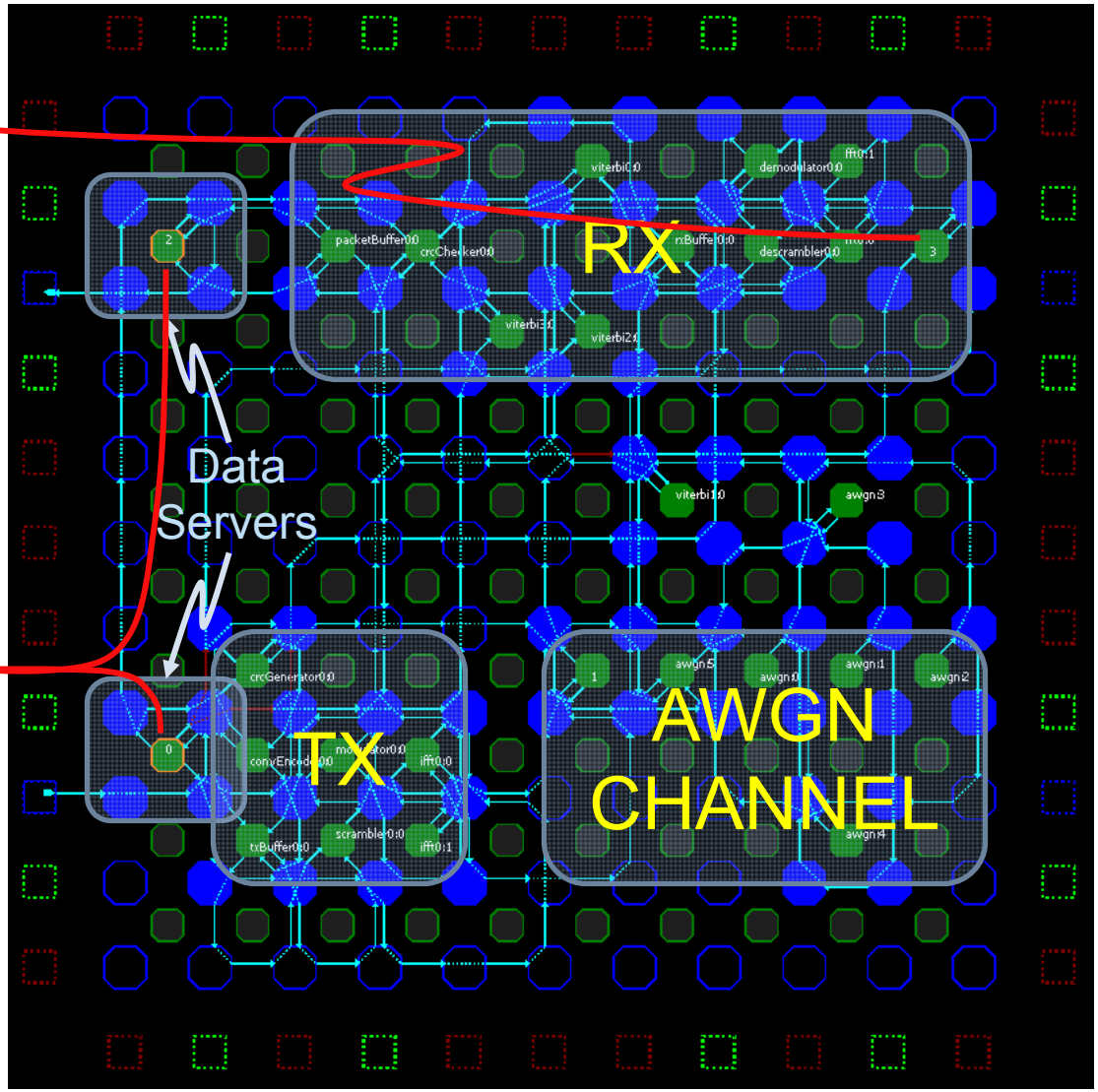
Real-Time Analysis (RTA) Example Framework



Real-Time Analysis: BER / PER Characterization







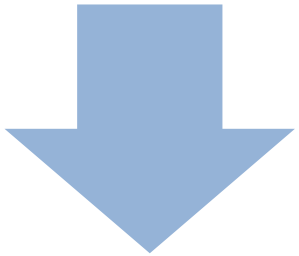
Demo Summary

- 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

Coherent Logix™

- 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

Coherent Logix™

Thank you for attending!

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