CPRI Design Example

Introduction

In wireless applications, a fundamental path is the Remote Radio Head (RRH) to Base Station (BTS) path. In the downlink, an analog radio signal is translated into a digital format in which it can then be processed and manipulated. In the uplink direction, the opposite processing is applied. This example design will showcase three of the functions that are part of these data paths: compression, mapping of IQ samples into a CPRI payload, and a CPRI link that carries Control and IQ Payload between the RRH and the BTS. The modules/functions showcased in this design example are part of Altera's solution for wireless applications

Requirements

- Arria 10 PCle Development Kit
- FMC Loopback Card
- Quartus II 16.0
- Arria 10 PCIe Development Kit ClockControl
- Altera_CPRI_IQ_Mapper Tool
- CPRI v6 IP License
- ModelSim 10.1b or newer version

High-Level Description

A high-level block diagram of the design is shown in Figure 1.0. This design example connects Compression/DeCompression, IQ Mapper/DeMapper, and CPRI IP modules. IQ samples are generated by Linear Feedback Shift Registers (LFSR) and are driven into the Compression Modules. After compression the IQ samples are mapped by the IQ Mapper module and are then driven into the CPRI IP. The CPRI module implements the CPRI protocol. It loads the IQ samples unto the CPRI IQ Data Plane. In this example the CPRI transmit serial link is routed back to the receive serial link, implementing an electrical serial loopback. In the receive direction (uplink), the IQ samples are extracted from the CPRI Frame by the CPRI module and are sent to the IQ DeMapper. From the DeMapper, the IQ samples go to the DeCompression modules.

To show the integrity of the IQ data and the impact of compression on the IQ data, this design example uses an Error Vector Magnitude module. The uncompressed IQ Data, generated by the LFSRs, and the De-Compressed IQ Data received in the uplink direction (output of the DeCompression modules) are sent into the EVM module which calculates a difference in magnitude between the two.

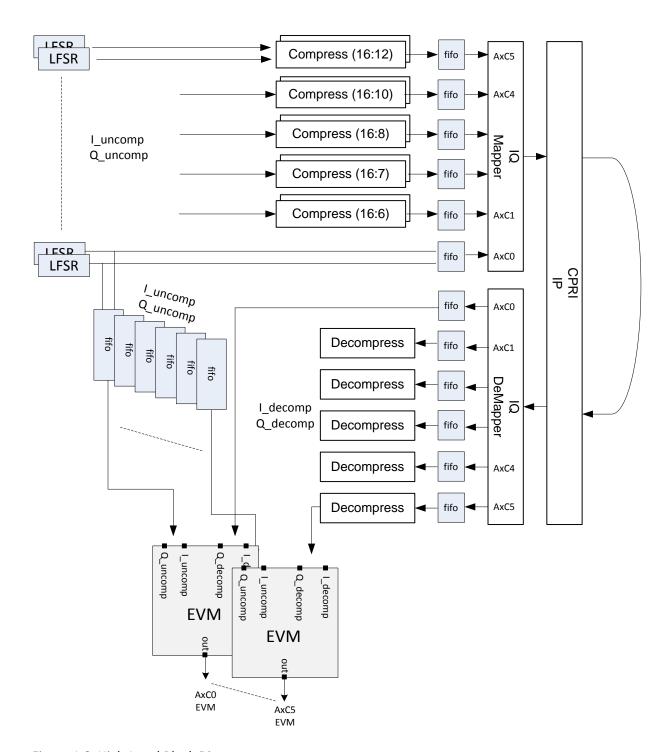


Figure 1.0 High-Level Block Diagram

Features

Table 1.0 shows the features corresponding to each of the IP modules demonstrated in this design example.

Table 1.0 Features

Module	Feature					
CPRI IP	REC Master					
	9.8 Gbps					
	6 AxCs					
	Direct IQ Mapper Interface					
Mapper	9.8 Gbps					
	6 AxCs					
	8X Sampling					
	20MHz LTE					
Compression	AxC0 : No Compression					
	AxC1: 16:12 Compression					
	AxC2: 16:10 Compression					
	AxC3: 16:8 Compression					
	AxC4: 16:7 Compression					
	AxC5: 16:6 Compression					

Running the Design

Setup and connect hardware

- Connect the power supply to the PCIe Development Board
- Connect the USB-Blaster cable to the PCIe Development Board and to a USB port on your PC/Laptop
- Insert an FMC Loopback Card on FMC Port A of the PCIe Development Board
- Power On the board

Program the Clock Source

- Bring up the Clock GUI, ClockControl.exe
- Select the Si5338 (U14) tab
- Enter 307.20 for CLK1 and click on "Set New Freq"
- Close the Clock GUI

Program the FPGA

- Download the design, cpri_de_a10.zip
- Unzip the cpri_de_a10.zip file
- Change directory to qdir
- Invoke Quartus and open the project, top_rec.qpf
- Open SignalTAP, Tools -> SignalTAP II Logic Analyzer
- Program the device with top_rec.sof

Enable the CPRI Transmitter

- Bring up the System Console, Tools -> System Debugging Tools -> System Console
- Change directory to ../system_console
- source main_run.tcl
- reg_write 0x02000008 0x1

Observe the activity on SignalTAP

og. rng @	2016/07/20 10:24:22 (0:0:0.1 elapsed) #2	click to insert time bar															
ype Alia	s Name	-128	()	128		256		384		512		640		768		89
•	±-state_l1_synch_reg[20]								6h								
*	axc0_evm_val_reg			┸	_	_Л_	_/∟	╜	╜╙	╜	╜╙	┚┖	__	╜╙	╜╙	⅃┖	
•									0000h								
*	axc1_evm_val_reg			\	\∟	_/∟	_/∟	╜	ᆜ┖	⅃┖	_/∟	╜╙	_/∟	╜╙	__	⅃┖	
•									0009h								
*	axc2_evm_val_reg			\	\∟	_/∟	_/∟	⅃┖	ᆜ┖	⅃┖	ᆜ┖	⅃Ĺ	_/∟	⅃Ĺ	ᆜ┖	⅃┖	
•									0037h								
*	axc3_evm_val_reg			\	_	_Л_	_/∟	╜	ᆜ┖	╜	_/∟	╜╙	_/∟	╜╙	╜╙	⅃┖	
	⊕ axc3_evm_reg[150]								00FEh								
	axc4_evm_val_reg				_	┸	__	⅃┖	┸	╌	┸	┚┖	┸	┚┖	┸	⅃┖	
•	± axc4_evm_reg[150]								0196h								Ξ
-	axc5_evm_val_reg				_	_Л_	_/∟	⅃┖	╜	⅃┖	┚┖	┚┖	┸	⅃L	╜╙	⅃┖	
	⊕ axc5_evm_reg[150]								0267h								
•	iq_mapper:iq_mapper axc0_read		Ш		_	_	_	_/L	_/L	_/_	_/L	_/∟	_/L	_/L	_L	_L	
•	±-iq_mapper:iq_mapper axc0_data[390]															\blacksquare	
-	iq_mapper:iq_mapper axc1_read		oxdot		\L	_	_	_/L	_/L	_/_	_	__	_/L	__	_/∟	_L	
•	±-iq_mapper:iq_mapper axc1_data[390]															\blacksquare	
	iq_mapper:iq_mapper axc2_read	\Box	\square	╙	_	_	_	_/L	_/_	_/_	_/L	_/_	__	__	_/_	⅃┖	
•	iq_mapper.iq_mapper axc2_data[390]															\Box	_
	iq_mapper:iq_mapper axc3_read		oxdot	Ш.	_	_	_	_/L	__	_/L	_	__	_/L	__	_	⅃┖	
5	⊞in mannerin mannerlaxc3 data[39 0]															_	_

Simulating the Design

You must have access to Modelsim 10.1b or newer.

- Change directory to sim
- Make the file run_sim an executable i.e. chmod 777 run_sim
- Execute run_sim. i.e. ./run_sim

The run_sim script will compile the necessary libraries and components as well as the testbench. It will bring up ModelSim with predefined nodes to trace in the Wave panel. You can detach the Wave panel and monitor the activity on the signals being traced. The simulation will wait until Frame Synchronization has been acquired and then it will run for 20,000 clock cycles.

IP Module Details

Compression

Obtaining the compression/decompression modules

The compression and decompression modules are included in the package, cpri_pkg.sv. The names of the modules are ccam (compression) and ceam (decompression). When you generate a CPRI IP instance, the cpri_pkg.sv is generated under the following folders.

```
For synthesis it is located under,
<your_cpri_instance_name>/altera_cpri_ii_instance_160/synth/src_hdl.

For simulation it is located under,
<your_cpri_instance_name>/altera_cpri_ii_instance_160/sim/<your simulator>/.

The "cpri_pkg.sv" is encrypted for both synthesis and simulation.
```

Using the compression/decompression modules

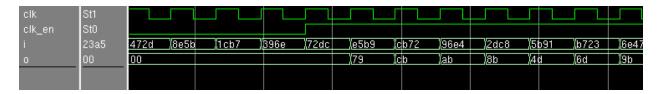
The following shows the instantiation of the modules in Verilog HDL.

```
ccam #(
 .EXPANDED_WIDTH
                             (EXPANDED_WIDTH),
 .COMPRESSED_WIDTH
                             (COMPRESSED_WIDTH)
) i ccam (
 .clk
                                            // IQ Sampling Clock
              (clk_s
                                     ),
              (clk_s_rst
                                     ),
                                            // Reset synchronous to Sampling Clock
 .rst
              (uncomp_dat_val
                                            // uncompressed data input valid
 .clk en
                                     ),
 i.
              (uncomp_data
                                     ),
                                            // uncompressed data input [EXPANDED_WIDTH-1:0]
              (comp data
                                     )
                                            // compressed data output [COMPRESSED WIDTH-1:0]
 o.
);
ceam #(
 .EXPANDED_WIDTH
                             (EXPANDED_WIDTH),
 .COMPRESSED WIDTH
                             (COMPRESSED_WIDTH)
) i ceam (
 .clk
              (cpri_clk
                                     ),
                                            // In this example, the CPRI clkout
              (cpri clk rst
                                     ),
                                            // Reset synchronous to the CPRI clkout
 .rst
 .clk_en
              (comp_data_val
                                     ),
                                            // compressed data input valid
 i.
              (comp_data
                                     ),
                                            // compressed data input [COMPRESSED WIDTH-1:0]
              (decomp data
                                            // decompressed data output [EXPANDED WIDTH-1:0]
                                     )
 .0
);
```

To use the embedded compression modules, include the "cpri_pkg.sv" in your project.

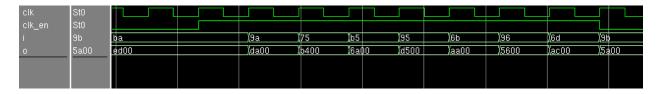
Compression Input/Output Interface

The following is a capture of a simulation waveform for the 16:8 compression instance used in the design.



DeCompression Input/Output Interface

The following is a capture of a simulation waveform for the 8:16 de-compression instance used in the design.



Mapper

Obtaining the IQ Mapper/DeMapper Code GenerationTool

You can download the tool using the following link.

http://alterawiki.com/wiki/File:Altera CPRI IQ Mapper.zip

The user guide to the IQ Mapper can be found in the following link.

http://alterawiki.com/wiki/File:Altera CPRI IQ Mapper User Guide.pdf

Using the spreadsheet 9.8G 8xS varying sample widths

The tool is an Excel spreadsheet that allows you to configure the IQ Data Plane for the CPRI Radio Frame supporting 0.6144 Gbps to 10.1276 Gbps data rates. In this example, the Line Rate selected is 9.8304 Gbps and supports Symmetrical DL/UL CPRI Frames. The following captures show the configuration of the User Plane for 6 AxCs using 8x sampling with each AxC having a different sample width, supporting the different compression ratios.

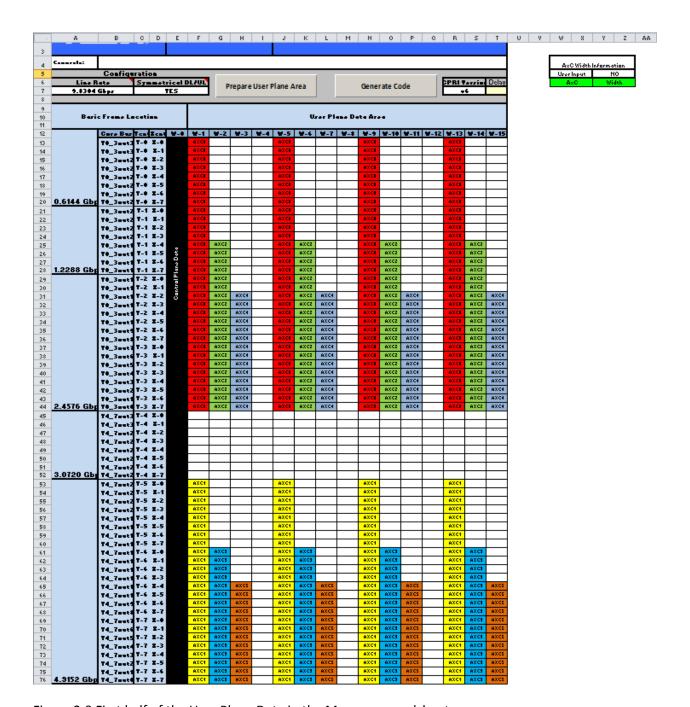


Figure 2.0 First half of the User Plane Data in the Mapper spreadsheet

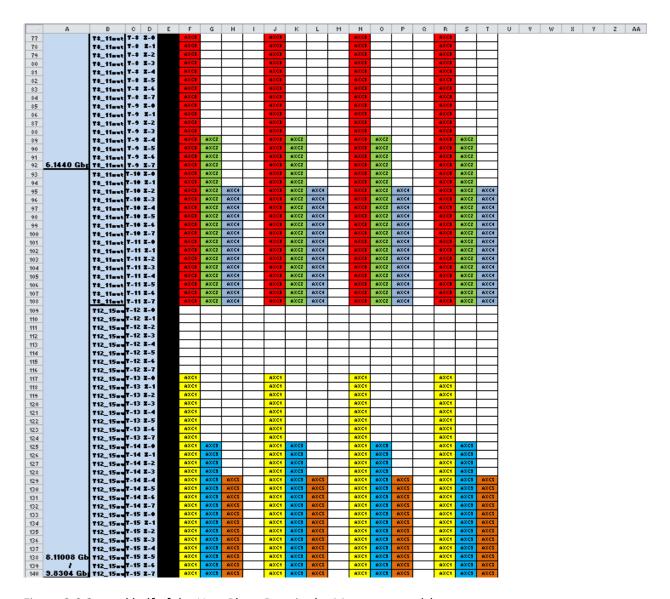


Figure 3.0 Second half of the User Plane Data in the Mapper spreadsheet

Generated RTL

The Mapper/DeMapper register transfer logic, RTL, is generated in the following files.

designs/lib/iq_demapper.v
designs/lib/iq_mapper.v

In this design the iq_demapper.v and iq_mapper.v files were moved and are contained in the "rtl" design subfolder.

Using the Mapper/DeMapper modules

The top-level design module, top_rec.v, has the instantiation of the IQ Mapper and DeMapper module. Please use that as an example of how to instantiate and connect the ports of the modules.

```
iq_mapper iq_mapper
                                                // input
 .cpri_clk
                        (cpri_clk),
 .rst_n
                        (~cpri_clk_rst),
                                                // input
                        (aux_tx_seq[6:0]),
                                                // input
 .aux_tx_seq
 .map_ena
                        (1'b1),
                                                // input
 .axc0_data
                        (axc0_iq),
                                                // input 40bits wide but only X are used for this design
 .axc0 read
                        (axc0_read),
                                                // output
 .axc1_data
                                                // input
                        (axc1_iq),
 .axc1 read
                        (axc1_read),
                                                // output
 .axc2 data
                        (axc2 iq),
                                                // input
 .axc2_read
                        (axc2_read),
                                                // output
 .axc3_data
                        (axc3_iq),
                                                // input
 .axc3_read
                        (axc3_read),
                                                // output
 .axc4 data
                        (axc4_iq),
                                                // input
 .axc4 read
                        (axc4 read),
                                                // output
 .axc5_data
                        (axc5_iq),
                                                // input
 .axc5_read
                        (axc5_read),
                                                // output
 .cpri_tx_data
                        (iq_tx_data[31:0]),
                                                // output
 .cpri_tx_ready
                        (iq_tx_valid[3:0])
                                                // output
);
```

IQ Framer Input/Output Interface

Figure 4.0 shows the content of IQ samples for two CPRI Frames at the Mapper inputs and two full CPRI Frames at the Mapper outputs. There is a delay of one CPRI Frame between the presence of the IQ samples and its corresponding mapped CPRI Frame at the output.

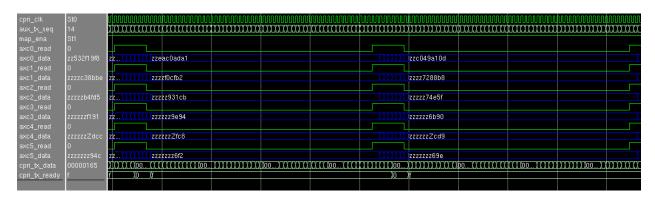


Figure 4.0 Frame N and N+1 Mapper inputs and Frame N-1 and N Mapper Outputs

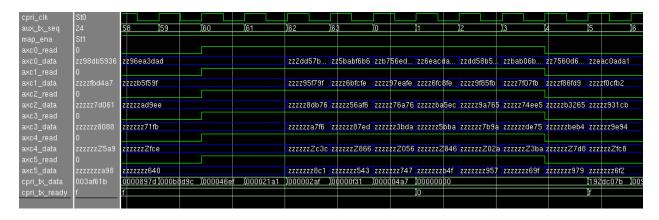


Figure 5.0 Frame N Mapper Input (Detailed)

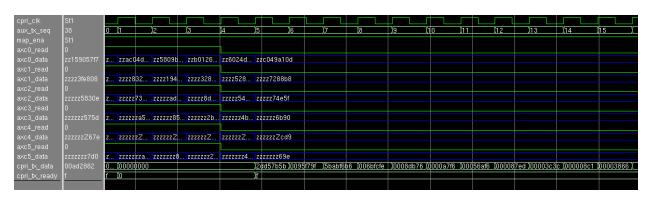


Figure 6.0 Frame N Mapper Output (Detailed)

```
iq_demapper iq_demapper
 .cpri_clk
                        (cpri_clk),
                                                // input
                        (~cpri_clk_rst),
                                                // input
 .rst_n
                        (aux_rx_seq[6:0]),
                                                // input
 .aux_rx_seq
 .demap_ena
                        (1'b1),
                                                // input
 .axc0_data
                        (axc0_data),
                                                // output
 .axc0_valid
                        (axc0_valid),
                                                // output
 .axc1_data
                        (axc1_data),
                                                // output
                                                // output
 .axc1_valid
                        (axc1_valid),
 .axc2_data
                        (axc2_data),
                                                // output
 . axc 2\_valid
                        (axc2_valid),
                                                // output
 .axc3_data
                        (axc3_data),
                                                // output
 .axc3_valid
                        (axc3_valid),
                                                // output
 .axc4_data
                        (axc4_data),
                                                // output
 .axc4_valid
                        (axc4_valid),
                                                // output
 .axc5_data
                        (axc5_data),
                                                // output
 .axc5_valid
                        (axc5_valid),
                                                // ouput
                                                // input
 .cpri_rx_data
                        (iq_rx_data),
 .cpri_rx_ready
                        (iq_rx_valid)
                                                // input
);
```

cpri_clk	St0	TUUUUU	\mathbf{m}	mmm	mm	mm	mm	سسسا	mm	mm	m	mm	m	m	m	mm
aux_rx_seq	44															
demap_ena	St1															
cpri_rx_ready	f	f												χo)f	
cpri_rx_data	000007e6) X X00000			X X X X00	000 X X			(00000)			(00000		
axc0_valid	0															
axc0_data	002a912518		() (009d4	1f4301												X X00
axc1_valid	0															
axc1_data	00007869a3		(00000	d06d7												X X00
axc2_valid	0															
axc2_data	000009ba11		((((((((((((((((((((a4809												X X00
axc3_valid	0															
axc3_data	000000af7f) X X00000	07433												X X X00
axc4_valid	0															
axc4_data	0000001d23		((((((((((((((((((((024b9												((((((((((((((((((((
axc5_valid	0															
axc5_data	00000005df		((((((((((((((((((((00fce												X X00

Figure 7.0 Frame N DeMapper Input and Output

cpri_clk	St1															
aux_rx_seq	32	3	/4	(5	(6),8	()9	10	(11	(12	(13	14	15	16 (1	7
demap_ena	St1															
cpri_rx_ready	f	0)f													
cpri_rx_data	00000000	00	3a9e8602	005439a	f (753)	c0c05 (00743	5f (0008481:	(00006353	00024026	(000	09373 (00001	9d6 (00000fd6	00002346	00000fde	000000	
axc0_valid	0															
axc0_data	0029ff722c	00	(007f3a5	00fe75a.	(00fc	ea4 00f9d4	9 (00f3a92	(00e7535)	00cea7a	(009	ld4f4301					
axc1_valid	0															
axc1_data	000090d5d0	00	,00001e6	(00003cd	(000)	05cd (00007	d (0000e68.	. (0000cd0)	0000ad0	(000	08d06d7					
axc2_valid	0															
axc2_data	00000833ee	00	(0000078	<u> </u>	(000	00a6 (00000)	6 (0000031.	. (0000051)	0000071	(000	00a4809					
axc3_valid	0															
axc3_data	0000004de6	00	(000000d	<u> Дофооооь</u>	(000	00000) (00000)	6 (0000009.	. (0000006)	0000009	(000	0007433					
axc4_valid	0															
axc4_data	0000001ec7	00	(0000003	<u> Хофооооз</u>	(000	ф <u>ооз Хооооо</u>	2 X0000002.	. (0000001)	0000002	(000	00024b9					
axc5_valid	0															
axc5_data	0000000458	00	(00000000	<u> </u>	(000	0000 (00000	o <u>X</u> 0000000.	0000000	0000000	(000	0000fce					

Figure 8.0 Frame N DeMapper Input (Detailed)

cpri_clk	St1																			
aux_rx_seq	35	3	/4	X5		\\6		7		(8		(9		(10		(11		(12		(18
demap_ena	St1																			
cpri_rx_ready	f	0	\f																	
cpri_rx_data	00000000	00	(9e5402e7	0052	9fdd	3ca8	05cf	(007	9fba	(000	175d1	(000	08f9e	(0002	ec7	(000	05f7c	(00	00384d	X_
axc0_valid	0																			
axc0_data	00c297ca15	00	(003a9e8602	(0075	3c0c05	(00ea	79180b	(00d4	f23017	(00a)	9e5602e	(005	3cac05c	(00a7	9580b9	(004	f2a0173			
axc1_valid	0																			
axc1_data	000040379b	00	00005439af	(0000	74375f	(0000	b0cb7f	(000)	90c97f	(000)	06196ff	(000	08329fe	(0000	1947fd	(000	0329fee			
axc2_valid	0																			
axc2_data	00000497e2	00	0000084813	(0000	024026	(0000	04404d	(000)	06409a	(000	008851a	(000	004159a	(0000)	061634	(000	0082d51			
axc3_valid	0																			
axc3_data	0000002a48	00	(0000ф0ь353	(0000	009373	(0000	0067ae	(000)	008e8e	(000	000595b	(000	000797b	(0000)	00cfde	(000	000afbe			
axc4_valid	0	ш																		
axc4_data	0000001e3a	00	X00000019d6	(0000	002346	(0000	0015ab	(000)	001dbb	(000	0002fdd	(000	00027cd	(0000)	001fbb	(000	0003c5d			
axc5_valid	0	ш																		
axc5_data	00000005a5	00	X0000000fd6	(0000	D00fde	(0000	000fb4	(000)	000f6c	(000	0000ea4	(000	0000d59	(00001	000b62	(000	0000950			

Figure 9.0 Frame N DeMapper Output (Detailed)

CPRI

Obtaining the IP

The CPRI IP is available from the Self-Service Licensing Center. You will need to have a myAltera account to access the Self Service Licensing Center. The IP product is CPRI Version 6.0 IP Core. Log on to myAltera using your credentials and then proceed to the Self Service Licensing Center. Follow the guides for accessing the CPRI IP and for creating a license. After installing the IP and the license, you can proceed to generating a variation of the CPRI IP.

Configuring and Generating a variant

This design example uses a variant of the CPRI IP with the following parameters. Options not listed below are disabled for this variant.

General

Selected device family:

Line bit rate (Mbit/s):

Synchronization mode:

Arria 10

9830.4

Master

Operation mode: TX/RX Duplex

Core clock source input: PCS
Transmitter local clock division factor: 1

Number of receiver CDR reference clock(s): 1
Receiver CDR reference clock frequency (MHz): 307.2
Recovered clock source: PCS
Receiver soft buffer depth: 6

Interfaces

Magement (CSR) interface standard: AvalonMM
Avalon-MM Interface addressing type: Word
Auxiliary and direct interface write latency cycle(s): 0

Enable auxiliary interface

Enable direct I/Q mapping interface

Direct IQ Interfaces

The main data path interfaces exercised in this example are the Direct IQ Interfaces.

Figure 10.0 shows the activity at the TX IQ Interface for one CPRI Radio Frame. The interface is driven by the IQ Mapper module. Notice the cycles in which the iq_tx_data bus carries 0x000000000 payload. These cycles correspond to the Control Word W-0, and the empty words W-4, W-7, and W-12 in the Data Plane. See Figure 2.0 and 3.0.

Figure 11.0 shows the activity at the RX IQ Interface for one CPRI Radio Frame. This interface drives the input port of the IQ DeMapper module.

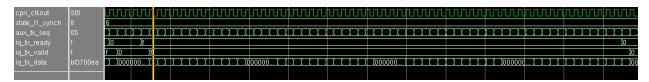


Figure 10.0 TX IQ Interface

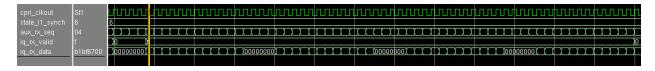


Figure 11.0 RX IQ Interface