Examples of "Instantiating WRPC in your own HDL design"

W. Hennig, September 2019 www.xia.com whennig@xia.com

This document describes the steps for implementing the White Rabbit core in a custom firmware design … actually two by now. The implementation follows the same principles in both designs, so they are treated here together [with changes for the second design in square brackets and purple font]. The document is intended as a commentary to section 8 of the WRPC user manual (v 4.2) based on personal experience; it is not an official manual of any kind.

Approach

The WR core here is implemented in a Kintex 7 FPGA [or a Spartan 6 FPGA] that is part of a desktop [or PXI(e)] module intended to digitize and process signals from radiation detectors, the Pixie-Net XL [or Pixie Hybrid] (see www.xia.com for details). The "user firmware" that the WR core is integrated into consists of logic to capture digital data streams from multiple ADCs and process them for pulse height and other information. It makes use of the WR core by a) tagging detector pulses with WR derived time stamps, b) running the ADCs with a clock synchronized by the WR core (using external PLLs), and c) sending data out via the WR Ethernet link. The "user firmware" plays no role in the instantiation of the WR core and is not discussed here further.

Hardware

The FPGA is a Kintex 7 XC7K70T-1FBG676C (K7) [or Spartan 6 XC6SLX45T-3FGG484 (S6), as on CUTE, CRIO, or SPEC reference designs]. It is attached to the following standard WR peripherals:

- I2C PROM (24AA64-I/MS)
- 1-wire temperature sensor (DS18B20U+T&R)
- Oscillators and PLLs (LF VCXO026156, VM53S3-25.000-2.5/-30+75, CDCM61004RHBT)
- Dual DAC (AD5663BRMZ from CUTE design)
- SFP cage (with WR recommended modules)

The CDCM PLL is driven by the WR controlled oscillator and has multiple outputs of 125 MHz clocks. One output is connected directly to the K7's GTX [or S6's GTP] clock reference input. Another is connected to a clock fanout PLL (LMK03003) that provides LVDS clocks of programmable frequency to up to 5 ADC chips and to a generic clock capable pin pair of the K7. [A third output of the CDCM PLL is connected directly to a generic clock capable pin pair of the S6. S6 GTP pins locations are identical to the CUTE and CRIO reference designs.]

The K7 is connected with a slow general purpose I/O bus to a Zynq processor board (MicroZed) to configure the FPGA, set processing parameters, and monitor data acquisition runs. The Zynq is thus used as a "controller host PC" for the Kintex. A Zynq UART port is connected to the WR UART interface pins on the K7. [The S6 is connected to a shared PXI local bus from PLX9054]

The closest official WR reference designs are CLBv2 and FASEC [or CUTE, CRIO and SPEC].

HDL Project Setup

This work uses Xilinx Vivado 2018.1 [or ISE 14.5] for the HDL design. Vivado projects are generally created by a tcl script. Unfortunately the CLBv2 uses ISE and does not include a script, and the FASEC design is based on a Zynq and the script is overly complex. Therefore the tcl script of an earlier Pixie processor board was adapted to include the source files of the WR core. A file "WR_files.list" extracted from the FASEC

reference design (wr-cores\syn\wrc_board_fasec_ip.tcl) specifies the WR files so that the script can add it to the Vivado project in one command

```
vivado_procs::add_file_list     $proj_dir/WR_files.list     $::env(PROJ_RTL)/WR   sources_1
```

See below (appendix A) for the contents of the list file. You can contact me for the full script. A similar file.list file specifies the files for the "user firmware" pulse processing logic.

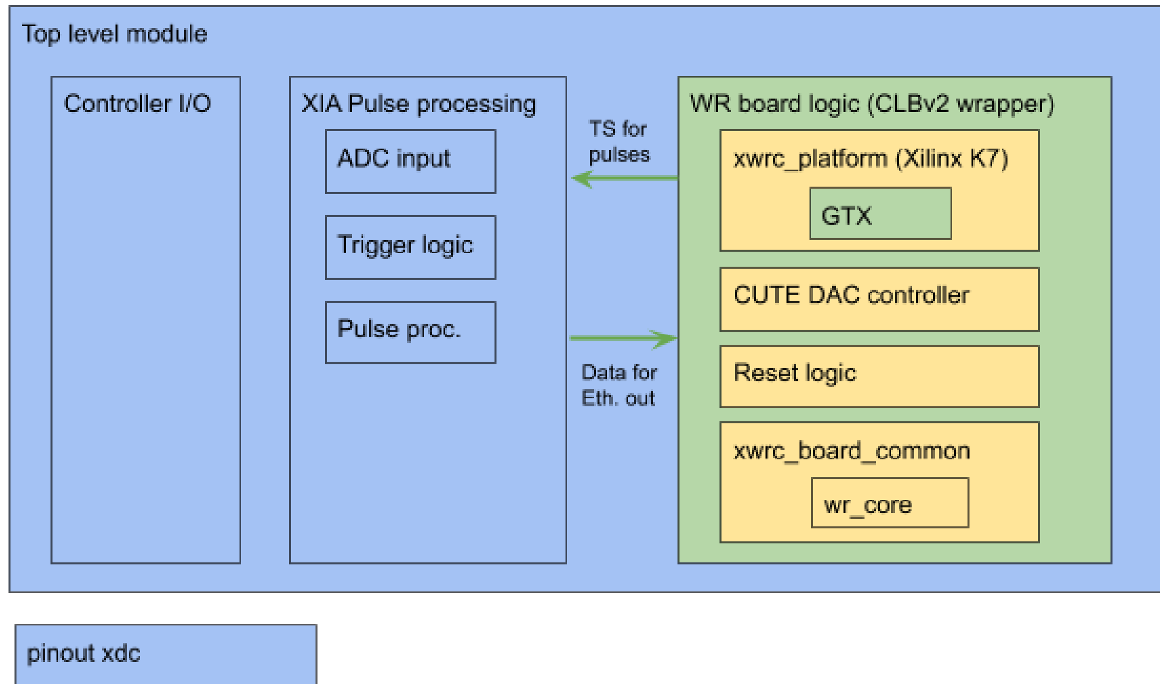

Fig. 1: Block diagram of firmware modules. Blue: original XIA design "user firmware". Yellow: original WR core modules (unchanged from v4.2). Green: WR modules modified by XIA for specific HW implementation. Compare to WR manual section 8, Fig. 1. [For S6, the WR board logic is based on the SPEC board wrapper and there is no XIA pulse processing, only PLX PCI(e) local bus I/O.]

Four significant board-specific changes have been made in the project compared to a WR reference design:

1) The top level module contains the K7 [or S6] input/output ports and instantiates a) the pulse processing logic in a Verilog module with various submodule, b) the controller IO to the Zynq, and c) the WR logic in a Verilog wrapper for the "WR board". [The S6 design uses a separate Artix 7 FPGA for pulse processing and only implements PLX PCI(e) local bus I/O]
2) The instantiation of the "WR board" is based on the CLBv2 [or SPEC] but modified for the actual I/O ports used in this design. Thankfully the WR distribution includes Verilog wrappers for the top level reference VHDL modules (wrc_board_clbv2 for xwrc_board_clbv2 [or wrc_board_spec for xwrc_board_spec]), so that it is easy to call the VHDL module(s) from a Verilog top module like mine.
3) Instead of the standard two-DAC controller module, the dual DAC module from the CUTE reference design is instantiated
4) The GTX interface has been replaced with a custom GTX core to accommodate this board's [poor?] pinout choice.

[The S6's dual GTP channel core has been specified as ch.0 active, ch.1 inactive in
…/platform/xilinx/wr_xilinx_pkg.vhd: g_gtp_enable_ch0 : integer := 1;
g_gtp_enable_ch1 : integer := 0;]

In the WR core, a number of parameters (generics) specify various code options. Those modified in this project are (in wrc_board_clbv2):

- g_with_external_clock_input : integer := 0;
  (not using external clock reference)
- g_dpram_initf : string := "…\WR\bin\wrpc\wrc_phy16.bram"; [or \wrc_phy8.bram";]
  (this is the original BRAM file for the SoftPLL without modification, only the path is updated)
- g_fabric_iface : string := "PLAINFBRC"; (must be caps, despite comments in lowercase)

The K7 clbv2 reference board design differs from the S6 SPEC reference board design mainly in the unused top level ports (e.g. streamer interface). However, the K7 GTX vs S6 GTP transceiver difference also has effects on the clocking and BRAM data width, causing the following differences in the K7 design compared to the more standard (SPEC) S6 design:

- the "clk_125m_pllref_p/n" clock inputs are not required
- the output "clk_125m_ref_o" from "xwrc_platform_xilinx" is actually a 62.5 MHz clock
- the ports "phy16_o/i" on "xwrc_platform_xilinx" connect to "phy16_o" on "xwrc_board_common" instead of "phy8_i/o".

Modifications of WR code files

Ports

The top level Verilog module calls out the Verilog compatible wrapper module "wrc_board_clbv2.vhd" which in turn calls out "xwrc_board_clbv2.vhd" [or "wrc_board_spec.vhd" which in turn calls out "xwrc_board_spec.vhd"]. The top level module connects only to the following ports of wrc_board_clbv2.vhd [or wrc_board_spec.vhd] (see also section 8.1.2 of the WR manual). See appendix B for the actual Verilog code sections.

name	dir	connection	Notes
areset_n_i	in	"Locked" signal from local DCM	
clk_20m_vcxo_i	in	IO pin	
clk_125m_pllref_p_i	In	IO pin	Only on S6 board
clk_125m_pllref_n_i	in	IO pin	Only on S6 board
clk_125m_gtp_n_i	in	IO pin	
clk_125m_gtp_p_i	in	IO pin	
clk_sys_62m5_o	out	Optional user firmware	62.5 MHz always
clk_ref_125m_o	out	Optional user firmware	62.5 MHz in K7 design due to 16bit GTX, 125 MHz in S6
clk_ref_62m5_o	out	Optional user firmware	
plldac_sync_n_o	out	IO pin	Replaces Standard DAC ports
plldac_ldac_n_o	out	IO pin	
plldac_clr_n_o	out	IO pin	
plldac_sclk_o	out	IO pin	
plldac_din_o	out	IO pin	

sfp_txp_o	out	IO pin	
sfp_txn_o	out	IO pin	
sfp_rxn_i	in	IO pin	
sfp_rxn_i	in	IO pin	
sfp_det_i	in	IO pin	WR_SFP_MOD_DEF0_in
sfp_sda_i	in	IO pin	WR_SFP_MOD_DEF2_in
sfp_sda_o	out	IO pin	WR_SFP_MOD_DEF2_out
sfp_scl_i	in	IO pin	WR_SFP_MOD_DEF1_in
sfp_scl_o	out	IO pin	WR_SFP_MOD_DEF1_out
sfp_rate_select_o	out	IO pin	
sfp_tx_fault_i	in	IO pin	
sfp_tx_disable_o	out	IO pin	
sfp_los_i	in	IO pin	
eeprom_sda_i	in	IO pin	Connected as e.g. IOBUF wrio0 (.O(SDA_i), .I(1'b0), .IO(SDA), .T(SDA_o));
eeprom_sda_o	out	IO pin	
eeprom_scl_i	In	IO pin	
eeprom_scl_o	out	IO pin	
onewire_i	in	IO pin	
onewire_oen_o	out	IO pin	
uart_rxd_i	in	IO pin	
uart_txd_o	out	IO pin	
tm_link_up_o	out	Optional user firmware	
tm_time_valid_o	out	Optional user firmware	
tm_tai_o	out	Optional user firmware	
tm_cycles_o	out	Optional user firmware	
pps_p_o	out	IO pin	
wrf_src_sel_o	out	Tied to wrf_src_ack_i	
wrf_src_ack_i	in	Tied to wrf_src_sel_o[0]	
wrf_src_stall_i	in	Tied to 0	
wrf_src_err_i	in	Tied to 0	
wrf_src_rty_i	in	Tied to 0	
wrf_snk_adr_i	in	Optional user firmware	
wrf_snk_dat_i	in	Optional user firmware	
wrf_snk_cyc_i	in	Optional user firmware	
wrf_snk_stb_i	in	Optional user firmware	
wrf_snk_we_i	in	Tied to 1	
wrf_snk_sel_i	in	Optional user firmware	
wrf_snk_ack_o	out	Optional user firmware	
wrf_snk_stall_o	out	Optional user firmware	

Clock Connections

The “clk_125m_pllref” clock connections are not required for the K7. The WR core (in the Kintex variant) does not need the fabric clock signal from the DAC controlled PLL; the GTX reference clock and the VCXO 20MHz clock are sufficient.

DAC Connections

The standard WR design has 2 DAC chips and the WR core outputs a common data signal and two chip select signals to program these DACs. In this design, following the CUTE and CRIO reference designs, a dual DAC is used that has different connectivity (1 data, one sync, more control).

Further down in the hierarchy, in module xwrc_board_clbv2.vhd, the CLBv2 design is therefore modified to instantiate the DAC module "cutewr_serial_dac_arb" instead of the usual "spec_serial_dac_arb".

K7 GTX

In this design, the RX and TX lines are split over two GTX transceivers. This appears to be legal per Xilinx manuals and simplifies my PCB layout, but is not foreseen in the WR core (and may affect performance?). Most likely this is not done in other boards, but I list here the steps taken to update the WR GTX core as an example for other GTX customization.
TX = MGTXTXP/N0_116 = GTX_X0Y4 = pin F2/F1
RX = MGTXRXP/N1_116 = GTX_X0Y5 = pin E4/E3
clk = MGTREFCLK0P/N_116 = pin D6/D5

<u>Implementation Steps</u>

1) **Start with single transceiver generic 1G Ethernet with Vivado GTX Wizard**
   Use VHDL project for better comparison to WR files
   Options
   - Include Shared Logic in example design
   - gigabit ethernet CC [maybe no CC needed]
   - In transceiver selection, uncheck Use GTX X0Y0 and check a more appropriate one (here: GTX_X0Y4)
   - select appropriate TX/RX clock source (here: REFCLK0_Q1; where CLK0 = MGTREFCLK0, _Q1 = _116)

2) **Update Wizard options to best match WR wrapper**
   (compare Wizard *gtwizard…wrapper_gt.vhd* with WR wrapper *whiterabbit_gtxe2_channel_wrapper_gt.vhd*)
   Remove optional ports and options
   - Encoding Tab: RXBUFRESET, RXBUFSTATUS, RXPCSRESET, TXCHARDISPMODE, TXCHARDISPVAL, TXPCSRESET, TXPMARESET
   - Comma Tab: RXBYTEREALIGN, RXDFEAGCOVRDEN, TXPOSTCURSOR, TXPRECURSOR, RX/TXPOLARITY
   - PCIe/SATA tab: TX/RXPOWERDOWN, TXELECIDLE, TXPRBSFORCEERR
   - CB and CC tab: uncheck "use clock Correction"
   Add optional ports and options
   - Comma Tab: RXSLIDE, RXBYTEISALIGN, RXCOMMADET, Decode valid comma only

3) **Keeping options the same, copy and recustomize Wizard setup for 2 transceivers** (gtwizard_dual). Select and check box to "use GTX_X0Y5" This creates a 1-transceiver GTX wrapper *gtwizard_dual_gt.vhd* equivalent to the WR wrapper and a mult" GTX wrapper wrapper instantiating 2 of the wrappers, gtwizard_dual_multi_gt.vhd.

4) **In *gtwizard_dual_gt.vhd, the* single transceiver wrapper, update all ports and attributes to match WR wrapper.** This can be done doing a line-by-line diff. Capitalization is a bit cumbersome. Also my Vivado uses version 3.6 of the Wizard, WR 4.2 an older version.
   List of ports created by Wizard, but not used in WR (these need to be tied to GND further up in the hierarchy or manually removed as ports)
   - txpolatity_in
   - txbufstatus_out
   - txdiffctrl_in, txinhibit_in

- rxdfelpmreset_in, rxmonitorout_out, rxmonitorsel_in
- rxprbserr_out, rxprbssel_in, rxprbscntreset_in
- eyescantrigger_in, eyescanreset_in
- dmonitorout_out
- qpllrefclk_in, cpllpd_in, cpllrefclksel_in

List of ports used in WR but not brought out in Wizard (these need to be added manually to the port list)
- rxcdrlock_out
- rxcdrreset_in

5) **Modify WR source file to call the 2-transceiver wrapper and connect WR core signals to wrapper ports.**
   Original: *wr_gtx_phy_family7.vhd*
   |-- *whiterabbit_gtxe2_channel_wrapper_gt.vhd*
   Modified*:* wr_gtx_phy_family7.vhd
   |-- *gtwizard_dual_multi_gt.vhd*
   |-- *gtwizard_dual_gt.vhd* (gtx0)
   |-- *gtwizard_dual_gt.vhd* (gtx1)
   |-- *gtwizard_dual_cpll_railing.vhd* (as generated by wizard)
   Here: TX uses GTX transceiver 0, RX uses GTX transceiver 1
   Signals from WR core to GTX inputs are sent to both transceivers
   TX outputs from GTX0 are connected to WR core, TX outputs from GTX1 are not connected
   RX outputs from GTX1 are connected to WR core, RX outputs from GTX0 are not connected

6) **Verify TX/RXPOLARITY matches PCB**
   These can be brought out as ports or edited in *gtwizard_dual_gt.vhd*

Addressing Compile Errors/Warnings

While the WR compiles ok "out of the box" (with a number of warnings that can be ignored), I encountered real errors if the diagnostic ports are connected to the top level Verilog module. The error is "error synth 8-26, instantiation from verilog of vhdl entity with complex port types not implemented" even though these ports are type "std_logic_vector". Therefore the following port definitions were removed in wrc_board_clbv2.vhd:

- aux_diag_i
- aux_diag_o
- tm_dac_value_o
- tm_dac_wr_o
- tm_clk_aux_lock_en_i
- tm_clk_aux_locked_o

Operation

For testing the operation of the custom WR implementation, the steps in section 4 of the WR user manual provide the initial guidance. In this work, the WR UART pins are connected to the Zynq PS UART port, and with the Zynq running Linux, communication is established by running the minicom utility on the Zynq Linux terminal interface.

UART settings for WR are the minicom default (115200 baud, 8N1, HW flow yes, SW flow no), /dev/ttyPS1

Connection to User Firmware

WR clock

For the “user firmware”, the WR clock outputs clk_sys_62m5_o and clk_ref_62m5_o [or clk_ref_125_o] could be used. In this work, however, we use a different path: The WR controls the VCXO oscillators via the DACs. The 25 MHz VCXO clock is buffered and fanned out in a first PLL chip that provides the clock to the K7 GTX clock reference and a second PLL fanout chip. Outputs from the second PLL chip are connected to the ADCs and the FPGA. This FPGA clock input is then used for ADC pulse processing. [The S6 design currently makes no use of this clock.]

WR time stamps

In our design, we want to record the WR time with each captured ADC pulse. To do so, we connect our “user firmware” to the ports “tm_tai_o” and “tm”cycles_o”, with contain the seconds and sub-second clock cycles of the WR time. Note that a cycle time here is 16ns (62.5 MHz clock).

User Data Output to WR Ethernet port

The WR core provides 3 options for data I/O through the Ethernet link: WR fabric, Streamers, and Etherbone. In this project, we plan to send data from the pulse processing logic out as UDP packages, and chose the WR fabric option. Etherbone was not explored. Streamer are described in a wiki page: https://ohwr.org/project/wr-cores/wikis/wr-streamers-module

The WR fabric interface is described in the WR user manual section 8.1.6. We developed test logic to generate a sample UDP packet as shown in Appendix C. Here the controller logic provides the destination MAC address (MAC_ADDR) and a pulse (u_senddata) to trigger the generation of the packet. Source and destination IP and other UDP parameters are currently simply hard coded, and the “data” is a series of “0x1234”.

Other Notes and Suggestions

1. In WR manual section 4.1 it is not clear that the I2C EEPROM needs to be initialized. (This may be specific to designs like this, where no SPI PROM is present). To initialize, type in the WR UART interface

    sdb fs 1 0 80

    where

    "1" = I2C EEPROM
    "0" = base address
    "80" = the *decimal* representation of 1010_000, the PROM I2C address and ID

2. Generics for the data interface are described in all lower case, e.g. in wrc_board_clbv2.vhd
    -- "plainfbrc" = expose WRC fabric interface
    -- "streamers" = attach WRC streamers to fabric interface
    -- "etherbone" = attach Etherbone slave to fabric interface
    but the string it must be PLAINFBRC, not PLAIN or plainfbrc. There is an error check function defined in wr_board_pkg.vhd, but it does not create an error (in Vivado?) for the lowercase string; Vivado simply instantiates nothing.

3. In the Spartan 6 design, choosing the GTP channel was tricky. The code has several places where one can choose between ch.0 and ch.1; and wrong/mismatched settings can cause ISE placer errors. For me, switching to ch.0 (which seems to be the pinout in CUTE) required
    in xwrc_platform_xilinx.vhd:

    ```
    g_gtp_enable_ch0        : integer := 1;
    g_gtp_enable_ch1        : integer := 0;
    ```

    in wr_xilinx_pkg.vhd:

    ```
    g_gtp_enable_ch0        : integer := 1;          (line 46)
    g_gtp_enable_ch1        : integer := 0;
    ```

    and

    ```
    g_enable_ch0 : integer := 1;                     (line 92)
    g_enable_ch1 : integer := 0;
    ```

    in wr_gtp_phy_spartan6.vhd:

    ```
    g_enable_ch0     : integer := 1;
    g_enable_ch1     : integer := 0
    ```

    (It did not seem to propagate down from the highest level module)

Appendix A: Vivado tcl script file list for WR files

board/clbv2/wr_clbv2_pkg.vhd
board/clbv2/wrc_board_clbv2.vhd
board/clbv2/xwrc_board_clbv2.vhd
board/common/wr_board_pkg.vhd
board/common/xwrc_board_common.vhd
ip_cores/etherbone-core/hdl/eb_slave_core/eb_cfg_fifo.vhd
ip_cores/etherbone-core/hdl/eb_slave_core/eb_checksum.vhd
ip_cores/etherbone-core/hdl/eb_slave_core/eb_commit_fifo.vhd
ip_cores/etherbone-core/hdl/eb_slave_core/eb_eth_rx.vhd
ip_cores/etherbone-core/hdl/eb_slave_core/eb_eth_tx.vhd
ip_cores/etherbone-core/hdl/eb_slave_core/eb_ethernet_slave.vhd
ip_cores/etherbone-core/hdl/eb_slave_core/eb_fifo.vhd
ip_cores/etherbone-core/hdl/eb_slave_core/eb_hdr_pkg.vhd
ip_cores/etherbone-core/hdl/eb_slave_core/eb_internals_pkg.vhd
ip_cores/etherbone-core/hdl/eb_slave_core/eb_pass_fifo.vhd
ip_cores/etherbone-core/hdl/eb_slave_core/eb_slave_fsm.vhd
ip_cores/etherbone-core/hdl/eb_slave_core/eb_slave_top.vhd
ip_cores/etherbone-core/hdl/eb_slave_core/eb_stream_narrow.vhd
ip_cores/etherbone-core/hdl/eb_slave_core/eb_stream_widen.vhd
ip_cores/etherbone-core/hdl/eb_slave_core/eb_tag_fifo.vhd
ip_cores/etherbone-core/hdl/eb_slave_core/eb_tx_mux.vhd
ip_cores/etherbone-core/hdl/eb_slave_core/eb_wbm_fifo.vhd
ip_cores/etherbone-core/hdl/eb_slave_core/etherbone_pkg.vhd
ip_cores/general-cores/modules/common/gc_crc_gen.vhd
ip_cores/general-cores/modules/common/gc_extend_pulse.vhd
ip_cores/general-cores/modules/common/gc_frequency_meter.vhd
ip_cores/general-cores/modules/common/gc_pulse_synchronizer.vhd
ip_cores/general-cores/modules/common/gc_pulse_synchronizer2.vhd
ip_cores/general-cores/modules/common/gc_reset.vhd
ip_cores/general-cores/modules/common/gc_sync_ffs.vhd
ip_cores/general-cores/modules/common/gc_sync_register.vhd
ip_cores/general-cores/modules/common/gencores_pkg.vhd
ip_cores/general-cores/modules/genrams/common/generic_shiftreg_fifo.vhd
ip_cores/general-cores/modules/genrams/common/inferred_async_fifo.vhd
ip_cores/general-cores/modules/genrams/common/inferred_sync_fifo.vhd
ip_cores/general-cores/modules/genrams/generic/generic_async_fifo.vhd
ip_cores/general-cores/modules/genrams/generic/generic_sync_fifo.vhd
ip_cores/general-cores/modules/genrams/genram_pkg.vhd
ip_cores/general-cores/modules/genrams/memory_loader_pkg.vhd
ip_cores/general-cores/modules/genrams/xilinx/gc_shiftreg.vhd
ip_cores/general-cores/modules/genrams/xilinx/generic_dpram.vhd
ip_cores/general-cores/modules/genrams/xilinx/generic_dpram_dualclock.vhd
ip_cores/general-cores/modules/genrams/xilinx/generic_dpram_sameclock.vhd
ip_cores/general-cores/modules/genrams/xilinx/generic_dpram_split.vhd
ip_cores/general-cores/modules/genrams/xilinx/generic_simple_dpram.vhd
ip_cores/general-cores/modules/wishbone/wb_crossbar/sdb_rom.vhd
ip_cores/general-cores/modules/wishbone/wb_crossbar/xwb_crossbar.vhd

ip_cores/general-cores/modules/wishbone/wb_crossbar/xwb_sdb_crossbar.vhd
ip_cores/general-cores/modules/wishbone/wb_dpram/xwb_dpram.vhd
ip_cores/general-cores/modules/wishbone/wb_lm32/generated/lm32_allprofiles.v
ip_cores/general-cores/modules/wishbone/wb_lm32/generated/xwb_lm32.vhd
ip_cores/general-cores/modules/wishbone/wb_lm32/platform/generic/jtag_tap.v
ip_cores/general-cores/modules/wishbone/wb_lm32/platform/generic/lm32_multiplier.v
ip_cores/general-cores/modules/wishbone/wb_lm32/src/jtag_cores.v
ip_cores/general-cores/modules/wishbone/wb_lm32/src/lm32_adder.v
ip_cores/general-cores/modules/wishbone/wb_lm32/src/lm32_addsub.v
ip_cores/general-cores/modules/wishbone/wb_lm32/src/lm32_dp_ram.vhd
ip_cores/general-cores/modules/wishbone/wb_lm32/src/lm32_logic_op.v
ip_cores/general-cores/modules/wishbone/wb_lm32/src/lm32_ram.vhd
ip_cores/general-cores/modules/wishbone/wb_lm32/src/lm32_shifter.v
ip_cores/general-cores/modules/wishbone/wb_onewire_master/sockit_owm.v
ip_cores/general-cores/modules/wishbone/wb_onewire_master/wb_onewire_master.vhd
ip_cores/general-cores/modules/wishbone/wb_onewire_master/xwb_onewire_master.vhd
ip_cores/general-cores/modules/wishbone/wb_slave_adapter/wb_slave_adapter.vhd
ip_cores/general-cores/modules/wishbone/wb_uart/simple_uart_pkg.vhd
ip_cores/general-cores/modules/wishbone/wb_uart/simple_uart_wb.vhd
ip_cores/general-cores/modules/wishbone/wb_uart/uart_async_rx.vhd
ip_cores/general-cores/modules/wishbone/wb_uart/uart_async_tx.vhd
ip_cores/general-cores/modules/wishbone/wb_uart/uart_baud_gen.vhd
ip_cores/general-cores/modules/wishbone/wb_uart/wb_simple_uart.vhd
ip_cores/general-cores/modules/wishbone/wb_uart/xwb_simple_uart.vhd
ip_cores/general-cores/modules/wishbone/wbgen2/wbgen2_eic.vhd
ip_cores/general-cores/modules/wishbone/wbgen2/wbgen2_fifo_sync.vhd
ip_cores/general-cores/modules/wishbone/wbgen2/wbgen2_pkg.vhd
ip_cores/general-cores/modules/wishbone/wishbone_pkg.vhd
modules/fabric/xwrf_mux.vhd
modules/fabric/wr_fabric_pkg.vhd
modules/fabric/xwb_fabric_sink.vhd
modules/fabric/xwb_fabric_source.vhd
modules/timing/dmtd_phase_meas.vhd
modules/timing/dmtd_with_deglitcher.vhd
modules/timing/pulse_stamper.vhd
modules/wr_dacs/cutewr_serial_dac.vhd
modules/wr_dacs/cutewr_serial_dac_arb.vhd
modules/wr_endpoint/endpoint_pkg.vhd
modules/wr_endpoint/endpoint_private_pkg.vhd
modules/wr_endpoint/ep_1000basex_pcs.vhd
modules/wr_endpoint/ep_autonegotiation.vhd
modules/wr_endpoint/ep_clock_alignment_fifo.vhd
modules/wr_endpoint/ep_crc32_pkg.vhd
modules/wr_endpoint/ep_leds_controller.vhd
modules/wr_endpoint/ep_packet_filter.vhd
modules/wr_endpoint/ep_pcs_tbi_mdio_wb.vhd
modules/wr_endpoint/ep_registers_pkg.vhd
modules/wr_endpoint/ep_rtu_header_extract.vhd
modules/wr_endpoint/ep_rx_buffer.vhd

modules/wr_endpoint/ep_rx_crc_size_check.vhd
modules/wr_endpoint/ep_rx_early_address_match.vhd
modules/wr_endpoint/ep_rx_oob_insert.vhd
modules/wr_endpoint/ep_rx_path.vhd
modules/wr_endpoint/ep_rx_pcs_16bit.vhd
modules/wr_endpoint/ep_rx_pcs_8bit.vhd
modules/wr_endpoint/ep_rx_status_reg_insert.vhd
modules/wr_endpoint/ep_rx_vlan_unit.vhd
modules/wr_endpoint/ep_rx_wb_master.vhd
modules/wr_endpoint/ep_sync_detect.vhd
modules/wr_endpoint/ep_sync_detect_16bit.vhd
modules/wr_endpoint/ep_timestamping_unit.vhd
modules/wr_endpoint/ep_ts_counter.vhd
modules/wr_endpoint/ep_tx_crc_inserter.vhd
modules/wr_endpoint/ep_tx_header_processor.vhd
modules/wr_endpoint/ep_tx_inject_ctrl.vhd
modules/wr_endpoint/ep_tx_packet_injection.vhd
modules/wr_endpoint/ep_tx_path.vhd
modules/wr_endpoint/ep_tx_pcs_16bit.vhd
modules/wr_endpoint/ep_tx_pcs_8bit.vhd
modules/wr_endpoint/ep_tx_vlan_unit.vhd
modules/wr_endpoint/ep_wishbone_controller.vhd
modules/wr_endpoint/wr_endpoint.vhd
modules/wr_endpoint/xwr_endpoint.vhd
modules/wr_mini_nic/minic_wb_slave.vhd
modules/wr_mini_nic/minic_wbgen2_pkg.vhd
modules/wr_mini_nic/wr_mini_nic.vhd
modules/wr_mini_nic/xwr_mini_nic.vhd
modules/wr_pps_gen/pps_gen_wb.vhd
modules/wr_pps_gen/wr_pps_gen.vhd
modules/wr_pps_gen/xwr_pps_gen.vhd
modules/wr_softpll_ng/softpll_pkg.vhd
modules/wr_softpll_ng/spll_aligner.vhd
modules/wr_softpll_ng/spll_wb_slave.vhd
modules/wr_softpll_ng/spll_wbgen2_pkg.vhd
modules/wr_softpll_ng/wr_softpll_ng.vhd
modules/wr_softpll_ng/xwr_softpll_ng.vhd
modules/wr_streamers/dropping_buffer.vhd
modules/wr_streamers/escape_detector.vhd
modules/wr_streamers/escape_inserter.vhd
modules/wr_streamers/rx_streamer.vhd
modules/wr_streamers/streamers_pkg.vhd
modules/wr_streamers/streamers_priv_pkg.vhd
modules/wr_streamers/tx_streamer.vhd
modules/wr_streamers/wr_streamers_wb.vhd
modules/wr_streamers/wr_streamers_wbgen2_pkg.vhd
modules/wr_streamers/xrtx_streamers_stats.vhd
modules/wr_streamers/xrx_streamer.vhd
modules/wr_streamers/xrx_streamers_stats.vhd

modules/wr_streamers/xtx_streamer.vhd
modules/wr_streamers/xtx_streamers_stats.vhd
modules/wr_streamers/xwr_streamers.vhd
modules/wr_tbi_phy/disparity_gen_pkg.vhd
modules/wrc_core/wr_core.vhd
modules/wrc_core/wrc_diags_pkg.vhd
modules/wrc_core/wrc_diags_wb.vhd
modules/wrc_core/wrc_periph.vhd
modules/wrc_core/wrc_syscon_pkg.vhd
modules/wrc_core/wrc_syscon_wb.vhd
modules/wrc_core/wrcore_pkg.vhd
modules/wrc_core/xwr_core.vhd
modules/wrc_core/xwrc_diags_wb.vhd
platform/xilinx/wr_gtp_phy/family7-gtp/whiterabbit_gtpe2_channel_wrapper_gtrxreset_seq.vhd
platform/xilinx/wr_gtp_phy/family7-gtp/whiterabbit_gtpe2_channel_wrapper_gt.vhd
platform/xilinx/wr_gtp_phy/family7-gtp/whiterabbit_gtpe2_channel_wrapper.vhd
platform/xilinx/wr_gtp_phy/family7-gtp/wr_gtp_phy_family7.vhd
platform/xilinx/wr_gtp_phy/family7-gtx/wr_gtx_phy_family7.vhd
platform/xilinx/wr_gtp_phy/family7-gtx/gtwizard_dual_cpll_railing.vhd
platform/xilinx/wr_gtp_phy/family7-gtx/gtwizard_dual_gt.vhd
platform/xilinx/wr_gtp_phy/family7-gtx/gtwizard_dual_multi_gt.vhd
platform/xilinx/wr_gtp_phy/gtp_bitslide.vhd
platform/xilinx/wr_gtp_phy/spartan6/gtp_phase_align.vhd
platform/xilinx/wr_gtp_phy/spartan6/whiterabbitgtp_wrapper_tile_spartan6.vhd
platform/xilinx/wr_gtp_phy/spartan6/wr_gtp_phy_spartan6.vhd
platform/xilinx/wr_xilinx_pkg.vhd
platform/xilinx/xwrc_platform_xilinx.vhd

Appendix B: WR instantiation in top level Verilog module (Kintex)

```
//*******************************************************
// WR core instantiation
//*******************************************************

  wire pps_out, wr_ref_clk;
  wire WR_CLK20_in;
  wire WR_SW_OPTION_in;
  wire WR_UART_TXD_out;
  wire WR_UART_RXD_in;
  wire WR_PLLDAC_SYNCn_out;
  wire WR_PLLDAC_LADCn_out;
  wire WR_PLLDAC_CLRn_out;
  wire WR_PLLDAC_SCLK_out;
  wire WR_PLLDAC_DIN_out;
  wire WR_SFP_TX_FAULT_in;
  wire WR_SFP_TX_DISABLE_out;
  wire WR_SFP_MOD_DEF1_in,WR_SFP_MOD_DEF1_out; //,WR_SFP_MOD_DEF1_t;
  wire WR_SFP_MOD_DEF2_in,WR_SFP_MOD_DEF2_out; //,WR_SFP_MOD_DEF2_t;
  wire WR_SFP_MOD_DEF0_in;
  wire WR_SFP_RATE_SELECT0_out;
  wire WR_SFP_RATE_SELECT1_out;
  wire WR_SFP_LOS_in;
  wire WR_EEPROM_SCL_in,WR_EEPROM_SCL_out; // WR_EEPROM_SCL_t;
  wire WR_EEPROM_SDA_in,WR_EEPROM_SDA_out; // WR_EEPROM_SDA_t;
  wire WR_ONE_WIRE_data,WR_ONE_WIRE_oe;

  assign WR_SFP_RATE_SELECT1_out = 0;

  IBUFG  wri0 (.I(WR_CLK20), .O(WR_CLK20_in));
  IBUF   wri1 (.I(WR_SW_OPTION), .O(WR_SW_OPTION_in));
  IBUF   wri2 (.I(WR_UART_RXD), .O(WR_UART_RXD_in));
  IBUF wri3   (.I(WR_SFP_TX_FAULT), .O(WR_SFP_TX_FAULT_in));
  IBUF wri4   (.I(WR_SFP_LOS), .O(WR_SFP_LOS_in));
  IBUF wri5   (.I(WR_SFP_MOD_DEF0), .O(WR_SFP_MOD_DEF0_in));

  OBUF wro0   (.I(WR_UART_TXD_out), .O(WR_UART_TXD));
  OBUF wro1   (.I(WR_PLLDAC_LADCn_out), .O(WR_PLLDAC_LADCn));
  OBUF wro2   (.I(WR_PLLDAC_SYNCn_out), .O(WR_PLLDAC_SYNCn));
  OBUF wro3   (.I(WR_PLLDAC_CLRn_out), .O(WR_PLLDAC_CLRn));
  OBUF wro4   (.I(WR_PLLDAC_SCLK_out), .O(WR_PLLDAC_SCLK));
  OBUF wro5   (.I(WR_PLLDAC_DIN_out), .O(WR_PLLDAC_DIN));
  assign WR_PLLDAC_LADCn_out = 1'b0;
  assign WR_PLLDAC_CLRn_out =  1'b1;

  OBUF wro7   (.I(WR_SFP_TX_DISABLE_out), .O(WR_SFP_TX_DISABLE));
  OBUF wroc   (.I(WR_SFP_RATE_SELECT0_out), .O(WR_SFP_RATE_SELECT0));
  OBUF wrod   (.I(WR_SFP_RATE_SELECT1_out), .O(WR_SFP_RATE_SELECT1));

  IOBUF wrio0 (.O(WR_EEPROM_SDA_in), .I(1'b0), .IO(WR_EEPROM_SDA), .T(WR_EEPROM_SDA_out));
  IOBUF wrio1 (.O(WR_EEPROM_SCL_in), .I(1'b0), .IO(WR_EEPROM_SCL), .T(WR_EEPROM_SCL_out));
  IOBUF wrio2 (.O(WR_ONE_WIRE_data), .I(1'b0), .IO(WR_ONE_WIRE), .T(!WR_ONE_WIRE_oe));
  IOBUF wrio3 (.O(WR_SFP_MOD_DEF1_in), .I(1'b0), .IO(WR_SFP_MOD_DEF1), .T(WR_SFP_MOD_DEF1_out));
  IOBUF wrio4 (.O(WR_SFP_MOD_DEF2_in), .I(1'b0), .IO(WR_SFP_MOD_DEF2), .T(WR_SFP_MOD_DEF2_out));

  wire flash_sclk_o;
```

```
wrc_board_clbv2 WRlogic (

        //----------------------------------------------------------------------------
        //Clocks/resets
        //----------------------------------------------------------------------------
        //Reset input (active low, can be async)
        .areset_n_i         (h_locked), // : in  std_logic;
        //Optional reset input active low with rising edge detection. Does not
        //reset PLLs.
        //.areset_edge_n_i     ( ), // : in  std_logic ( ), // := '1';
        //Clock inputs from the board
        .clk_20m_vcxo_i      ( WR_CLK20_in), // : in  std_logic;
        //.clk_125m_pllref_p_i ( F0CLK_P), // : in  std_logic;                // replaces DCM_DAQ
        //.clk_125m_pllref_n_i ( F0CLK_N), // : in  std_logic;
        .clk_125m_gtp_n_i    ( GTX_WR_CLKN), // : in  std_logic;
        .clk_125m_gtp_p_i    ( GTX_WR_CLKP), // : in  std_logic;
         //10MHz ext ref clock input (g_with_external_clock_input = TRUE)
        .clk_10m_ext_i       ( ), // : in  std_logic                               ( ), // := '0';
        //External PPS input (g_with_external_clock_input = TRUE)
        .pps_ext_i           ( ), // : in  std_logic                               ( ), // := '0';
        //62.5MHz sys clock output
        .clk_sys_62m5_o      ( wr_sys_clk), // : out std_logic;
        //125MHz ref clock output
        .clk_ref_62m5_o      ( wr_ref_clk),  // : out std_logic;                //  for DCM_DAQ
        //active low reset outputs, synchronous to 62m5 and 125m clocks
        .rst_sys_62m5_n_o    ( ), // : out std_logic;
        .rst_ref_62m5_n_o    ( ), // : out std_logic;

        //----------------------------------------------------------------------------
        //Shared SPI interface to DACs
        //  PN XL uses a dual DAC as in CRIO/CUTE S6 ref design.
        //----------------------------------------------------------------------------
        //.plldac_sclk_o   ( WR_PLLDAC_SCLK_out),  // : out std_logic;
        //.plldac_din_o    ( WR_PLLDAC_DIN_out),   // : out std_logic;
        //.pll25dac_cs_n_o ( WR_PLLDAC_SYNCn_out), // : out std_logic;
        //.pll20dac_cs_n_o ( ), // : out std_logic;

        .plldac_sync_n_o   ( WR_PLLDAC_SYNCn_out),  //   : out std_logic;
        .plldac_ldac_n_o   ( WR_PLLDAC_LADCn_out),   //   : out std_logic;
        .plldac_clr_n_o    ( WR_PLLDAC_CLRn_out), //   : out std_logic;
        .plldac_sclk_o     ( WR_PLLDAC_SCLK_out), //    : out std_logic;
        .plldac_din_o      ( WR_PLLDAC_DIN_out), //    : out std_logic;

        //----------------------------------------------------------------------------
        //SFP I/O for transceiver and SFP management info
        //----------------------------------------------------------------------------
        .sfp_txp_o        ( GTX_WR_TXP), // : out std_logic;
        .sfp_txn_o        ( GTX_WR_TXN), // : out std_logic;
        .sfp_rxp_i        ( GTX_WR_RXP), // : in  std_logic;
        .sfp_rxn_i        ( GTX_WR_RXN), // : in  std_logic;
        .sfp_det_i         ( WR_SFP_MOD_DEF0_in), // : in  std_logic ( ), // := '1';
        .sfp_sda_i         ( WR_SFP_MOD_DEF2_in), // : in  std_logic;
        .sfp_sda_o         ( WR_SFP_MOD_DEF2_out), // : out std_logic;
        //.sfp_sda_t         ( WR_SFP_MOD_DEF2_t), // : out std_logic;
        .sfp_scl_i         ( WR_SFP_MOD_DEF1_in), // : in  std_logic;
        .sfp_scl_o         ( WR_SFP_MOD_DEF1_out), // : out std_logic;
        //.sfp_scl_t         ( WR_SFP_MOD_DEF1_t), // : out std_logic;
```

```
.sfp_rate_select_o ( WR_SFP_RATE_SELECT0_out), // : out std_logic;
.sfp_tx_fault_i    ( WR_SFP_TX_FAULT_in), // : in  std_logic ( ), // := '0';
.sfp_tx_disable_o  ( WR_SFP_TX_DISABLE_out), // : out std_logic;
.sfp_los_i         ( WR_SFP_LOS_in), // : in  std_logic ( ), // := '0';

//-----------------------------------------------------------------------------
//I2C EEPROM
//-----------------------------------------------------------------------------
.eeprom_sda_i ( WR_EEPROM_SDA_in), // : in  std_logic;
.eeprom_sda_o ( WR_EEPROM_SDA_out), // : out std_logic;
//.eeprom_sda_t ( WR_EEPROM_SDA_t), // : out std_logic;
.eeprom_scl_i ( WR_EEPROM_SCL_in), // : in  std_logic;
.eeprom_scl_o ( WR_EEPROM_SCL_out), // : out std_logic;
//.eeprom_scl_t ( WR_EEPROM_SCL_t), // : out std_logic;

  //-----------------------------------------------------------------------------
//Onewire interface
//-----------------------------------------------------------------------------
//.thermo_id_i ( WR_ONE_WIRE_out), // : in  std_logic;
//.thermo_id_o ( WR_ONE_WIRE_in), // : out std_logic;
//.thermo_id_t ( WR_ONE_WIRE_t), // : out std_logic;
.onewire_i     ( WR_ONE_WIRE_data), //: in  std_logic;
.onewire_oen_o ( WR_ONE_WIRE_oe), //: out std_logic;

//-----------------------------------------------------------------------------
//UART
//-----------------------------------------------------------------------------
.uart_rxd_i ( WR_UART_RXD_in), // : in  std_logic;
.uart_txd_o ( WR_UART_TXD_out), // : out std_logic;

//-----------------------------------------------------------------------------
//External WB interface
//-----------------------------------------------------------------------------
//.aux_master_o ( ), // : out t_wishbone_master_out;
//.aux_master_i ( ), // : in  t_wishbone_master_in ( ), // := cc_dummy_master_in;

//-----------------------------------------------------------------------------
//WR fabric interface (when g_fabric_iface = "plain")
//-----------------------------------------------------------------------------
.wrf_src_adr_o   ( wrf_src_adr_o ), //  out std_logic_vector(1 downto 0);
.wrf_src_dat_o   ( wrf_src_dat_o ), //  out std_logic_vector(15 downto 0);
.wrf_src_cyc_o   ( wrf_src_cyc_o ), //  out std_logic;
.wrf_src_stb_o   ( wrf_src_stb_o ), //  out std_logic;
.wrf_src_we_o    ( wrf_src_we_o ), //   out std_logic;
.wrf_src_sel_o   ( wrf_src_sel_o ), //  out std_logic_vector(1 downto 0);
.wrf_src_ack_i   (wrf_src_sel_o[0]), // wrf_src_ack_i ), //  in  std_logic;
.wrf_src_stall_i (1'b0), // wrf_src_stall_i ), //in  std_logic;
.wrf_src_err_i   (1'b0), // wrf_src_err_i  ), // in  std_logic;
.wrf_src_rty_i   (1'b0), // wrf_src_rty_i  ), // in  std_logic;
.wrf_snk_adr_i   ( wrf_snk_adr_i  ), // in  std_logic_vector(1 downto 0);
.wrf_snk_dat_i   ( wrf_snk_dat_i  ), // in  std_logic_vector(15 downto 0);
.wrf_snk_cyc_i   ( wrf_snk_cyc_i  ), // in  std_logic;
.wrf_snk_stb_i   ( wrf_snk_stb_i  ), // in  std_logic;
.wrf_snk_we_i    (1'b1), // wrf_snk_we_i   ), // in  std_logic;
.wrf_snk_sel_i   ( wrf_snk_sel_i  ), // in  std_logic_vector(1 downto 0);
.wrf_snk_ack_o   ( wrf_snk_ack_o  ), // out std_logic;
.wrf_snk_stall_o ( wrf_snk_stall_o), // out std_logic;
```

```
    .wrf_snk_err_o   ( wrf_snk_err_o  ), // out std_logic;
    .wrf_snk_rty_o   ( wrf_snk_rty_o  ), // out std_logic;

    //---------------------------------------------------------------------------
    //WR streamers (when g_fabric_iface = "streamers")
    //---------------------------------------------------------------------------
  /*
    .wrs_tx_data_i  (wrs_tx_data_i  ), //  : in  std_logic_vector(g_tx_streamer_params.data_width-1 downto 0) := (others => '0');
    .wrs_tx_valid_i       (wrs_tx_valid_i      ), // : in  std_logic                              := '0';
    .wrs_tx_dreq_o        (wrs_tx_dreq_o       ), // : out std_logic;
    .wrs_tx_last_i        (wrs_tx_last_i       ), // : in  std_logic                              := '1';
    .wrs_tx_flush_i       (wrs_tx_flush_i      ), // : in  std_logic                              := '0';
    .wrs_tx_cfg_mac_l_i   (48'd0               ), // : in  std_logic_vector(47 downto 0)          := x"000000000000";
    .wrs_tx_cfg_mac_t_i   (wrs_tx_cfg_mac_t_i  ), // : in  std_logic_vector(47 downto 0)          := x"ffffffffffff";
    .wrs_tx_cfg_etype_i   (16'hdbff            ), // : in  std_logic_vector(15 downto 0)          := x"dbff";
    .wrs_rx_first_o       (wrs_rx_first_o      ), // : out std_logic;
    .wrs_rx_last_o        (wrs_rx_last_o       ), // : out std_logic;
    .wrs_rx_data_o        (wrs_rx_data_o       ), // : out std_logic_vector(g_rx_streamer_params.data_width-1 downto 0);
    .wrs_rx_valid_o       (wrs_rx_valid_o      ), // : out std_logic;
    .wrs_rx_dreq_i        (wrs_rx_dreq_i       ), // : in  std_logic                              := '0';
    .wrs_rx_cfg_mac_l_i   (48'd0               ), // : in std_logic_vector(47 downto 0)           := x"000000000000";
    .wrs_rx_cfg_mac_r_i   (wrs_rx_cfg_mac_r_i  ), // : in std_logic_vector(47 downto 0)           := x"000000000000";
    .wrs_rx_cfg_etype_i   (16'hdbff            ), // : in std_logic_vector(15 downto 0)           := x"dbff";
    .wrs_rx_cfg_acc_b_i   (1'b1                ), // : in std_logic                               := '1';
    .wrs_rx_cfg_flt_r_i   (1'b0                ), // : in std_logic                               := '0';
    .wrs_rx_cfg_fix_l_i   (28'd0               ), // : in std_logic_vector(27 downto 0)           := x"0000000";
  */

    //.wrs_tx_data_i  ( ), // : in  std_logic_vector(g_tx_streamer_params.data_width-1 downto 0) ( ), // := (others => '0');
    //.wrs_tx_valid_i ( ), // : in  std_logic                              ( ), // := '0';
    //.wrs_tx_dreq_o  ( ), // : out std_logic;
    //.wrs_tx_last_i  ( ), // : in  std_logic                              ( ), // := '1';
    //.wrs_tx_flush_i ( ), // : in  std_logic                              ( ), // := '0';
    //.wrs_tx_cfg_i   ( ), // : in  t_tx_streamer_cfg                      ( ), // := c_tx_streamer_cfg_default;
    //.wrs_rx_first_o ( ), // : out std_logic;
    //.wrs_rx_last_o  ( ), // : out std_logic;
    //.wrs_rx_data_o  ( ), // : out std_logic_vector(g_rx_streamer_params.data_width-1 downto 0);
    //.wrs_rx_valid_o ( ), // : out std_logic;
    //.wrs_rx_dreq_i  ( ), // : in  std_logic                              ( ), // := '0';
    //.wrs_rx_cfg_i   ( ), // : in t_rx_streamer_cfg                       ( ), // := c_rx_streamer_cfg_default;
    // if used, update above structs to standard ports as defined in wrc_board_pnxl

    //---------------------------------------------------------------------------
    //Etherbone WB master interface (when g_fabric_iface = "etherbone")
    //---------------------------------------------------------------------------
    //.wb_eth_master_o ( ), // : out t_wishbone_master_out;
    //.wb_eth_master_i ( ), // : in  t_wishbone_master_in ( ), // := cc_dummy_master_in;
    // if used, update above structs to standard ports as defined in wrc_board_pnxl

    //---------------------------------------------------------------------------
    //Timecode I/F
    //---------------------------------------------------------------------------
    .tm_link_up_o    (tm_link_up ), // : out std_logic;               // state of Ethernet link (up/down), 1 means Ethernet link is up
    .tm_time_valid_o (tm_time_valid ), // : out std_logic;            // if 1, the timecode generated by the WRPC is valid
    .tm_tai_o        (tm_tai ), // : out std_logic_vector(39 downto 0);            // TAI part of the timecode (full seconds)
    .tm_cycles_o     (tm_cycles ), // : out std_logic_vector(27 downto 0);        // fractional part of each second represented by  the
state of 62.5 MHz counter (each count is 16 ns)
```

```
//---------------------------------------------------------------------------
//Buttons, LEDs and PPS output
//---------------------------------------------------------------------------
.led_act_o  ( ), // : out std_logic;
.led_link_o ( ), // : out std_logic;
//.btn1_i     ( 1'b0 ), // : in  std_logic ( ), // := '1';
//.btn2_i     ( 1'b0 ), // : in  std_logic ( ), // := '1';
//1PPS output
.pps_p_o    ( pps_out), // : out std_logic;
.pps_led_o  ( ) // : out std_logic;
//Link ok indication
//.link_ok_o  ( ) // : out std_logic
);
```

Appendix C: Generating a sample UDP packet for WR fabric interface

```
//---------------------------------------------------------------------------
//test code for plain fabric (wishbone)
//---------------------------------------------------------------------------
   wire [1:0] wrf_src_adr_o   ;
   wire [15:0] wrf_src_dat_o   ;
   wire wrf_src_cyc_o   ;
   wire wrf_src_stb_o   ;
   wire wrf_src_we_o    ;
   wire [1:0] wrf_src_sel_o   ;
   wire wrf_src_ack_i   ;
   wire wrf_src_stall_i ;
   wire wrf_src_err_i   ;
   wire wrf_src_rty_i   ;

    reg [1:0] wrf_snk_adr_i   ;
    reg [15:0] wrf_snk_dat_i   ;
    reg wrf_snk_cyc_i   ;
    reg wrf_snk_stb_i   ;
    wire wrf_snk_we_i    ;
    reg [1:0] wrf_snk_sel_i   ;
    wire wrf_snk_ack_o   ;
    wire wrf_snk_stall_o ;
    wire wrf_snk_err_o   ;
    wire wrf_snk_rty_o ;

   // constants for headers
   wire[15:0] wrf_snk_status = 16'h0200;   // last 4 bits  0 - isHP (high priority)
                                           //              1 - err (contains error)
                                           //              2 - vSMAC (valid source MAC else use from WR core)
                                           //              3 - vCRC (valid check sum)
   wire [15:0] EtherType = 16'h0800;       // IPv4
   wire [15:0] ipv4_w0   = 16'h4500;       // IPV4 header: version, header length, DSCP, ECN
   wire [15:0] ipv4_w1   = 16'd236;        // length total transmission to WR: 256bytes = 128 words.
                                           //      2 bytes WR status
                                           //      18 bytes Eth frame
                                           //      20 bytes IPv4 header
                                           //    =>  IPv4 total length = 256-20 = 236 bytes
   wire [15:0] ipv4_w2   = 16'h0000;       // identification
   wire [15:0] ipv4_w3   = 16'h0000;       // flags, fragment offset
   wire [15:0] ipv4_w4   = 16'h3F11;       // time to live (63), protocol (11-UDP):
   wire [15:0] ipv4_w5   = 16'hF79A;       // checksum    per http://www.n-cg.net/hec.htm
   wire [15:0] ipv4_w6   = 16'hc0a8;       // source IP  192.168
   wire [15:0] ipv4_w7   = 16'h0105;       // source IP .1.5
   wire [15:0] ipv4_w8   = 16'hc0a8;       // dest IP   192.168
   wire [15:0] ipv4_w9   = 16'h0111;       // dest IP   .1.17
   wire [15:0] udp_w0   = 16'h0000;        // UDP header source port
   wire [15:0] udp_w1   = 16'h0000;        // dest port
   wire [15:0] udp_w2   = 16'd216;         // length (header+data) = 216
   wire [15:0] udp_w3   = 16'h0000;        // checksum (0=disable)

  // counter for sending data
     reg [6:0] blkcntr;
    wire cntron;
   always @(posedge wr_sys_clk)
    if(u_senddata)
        blkcntr <= 127;
```

```
   else if(cntron & !wrf_snk_stall_o)        // count down unless stalled
      blkcntr <= blkcntr - 1;
  assign cntron =  |blkcntr;

 // data assembly
 always @(posedge wr_sys_clk)
    case (blkcntr)
        7'd127  : wrf_snk_dat_i <= wrf_snk_status;
        7'd126  : wrf_snk_dat_i <= MAC_ADDR[47:32];
        7'd125  : wrf_snk_dat_i <= MAC_ADDR[31:16];
        7'd124  : wrf_snk_dat_i <= MAC_ADDR[15:00];
        7'd123  : wrf_snk_dat_i <= 0;                  // source mac insert by WR core
        7'd122  : wrf_snk_dat_i <= 0;
        7'd121  : wrf_snk_dat_i <= 0;
        7'd120  : wrf_snk_dat_i <= EtherType;
        7'd119  : wrf_snk_dat_i <= ipv4_w0;
        7'd118  : wrf_snk_dat_i <= ipv4_w1;
        7'd117  : wrf_snk_dat_i <= ipv4_w2;
        7'd116  : wrf_snk_dat_i <= ipv4_w3;
        7'd115  : wrf_snk_dat_i <= ipv4_w4;
        7'd114  : wrf_snk_dat_i <= ipv4_w5;
        7'd113  : wrf_snk_dat_i <= ipv4_w6;
        7'd112  : wrf_snk_dat_i <= ipv4_w7;
        7'd111  : wrf_snk_dat_i <= ipv4_w8;
        7'd110  : wrf_snk_dat_i <= ipv4_w9;
        7'd109  : wrf_snk_dat_i <= udp_w0;
        7'd108  : wrf_snk_dat_i <= udp_w1;
        7'd107  : wrf_snk_dat_i <= udp_w2;
        7'd106  : wrf_snk_dat_i <= udp_w3;
        default: wrf_snk_dat_i <= 16'h1234;    // payload data ... tie to fifo from pulse processing
      endcase

// address and control
 always @(posedge wr_sys_clk)
    case (blkcntr)
          7'd127  : wrf_snk_adr_i <= 2'b10; // first word is status, addr = 2
          default: wrf_snk_adr_i <= 2'b00; // all other data, addr = 0
      endcase

 always @(posedge wr_sys_clk)
    if(blkcntr==7'd127)
       begin
          wrf_snk_sel_i <= 2'b11;
          wrf_snk_stb_i <= 1'b1;
       end
    else if(blkcntr==7'd0)
       begin
          wrf_snk_sel_i <= 2'b00;
          wrf_snk_stb_i <= 1'b0;
       end

 always @(posedge wr_sys_clk)
    if(blkcntr>7'd0)
       wrf_snk_cyc_i <=1'b1;
    else if (wrf_snk_ack_o==0)
       wrf_snk_cyc_i <=1'b0;
```