```
-------------------------------------------------------------------------
------
-- Title      : ODILE Top Controller
-- Project    :
-------------------------------------------------------------------------
------
-- File       : odile_controller.vhd
-- Author     : Ryan Thomas  <ryant@uchicago.edu>
-- Company    : University of Chicago
-- Created    : 2020-03-12
-- Last update: 2021-05-27
-- Platform   :
-- Standard   : VHDL'93/02
-------------------------------------------------------------------------
------
-- Description: Top Controller for the ODILE board. Produces several
different
-- reset signals and handles start/stop of sequencer
-------------------------------------------------------------------------
------
--!\file odile_controller.vhd

library ieee;
use ieee.std_logic_1164.all;

--!Helper package for blocks that interface with the commands from the
DAQ.
package ODILE_command_list is
  subtype command is std_logic_vector(23 downto 0);
  -------------------------------------------------------------------------
------
  -- List of supported commands. Note if you add a command, add it to the
  -- is_valid_command function below so the response generator knows it
is
  -- valid.
  -------------------------------------------------------------------------
------
  -- ! SIP  Allows to set the IP and MAC address from the board ID
    constant CMD_SET_IP      : command := X"53_49_50";
  -- ! PNG, toggle on or off PONG messages in data flow
  constant CMD_TOGGLE_PONG      : command := X"50_4E_47";
  --! GID get FPGA chip unique ID
  constant CMD_GET_CHIP_ID         : command := X"47_49_44";
  --!MST, define your ODILE board as the master board.
  constant CMD_SET_MST          : command := X"4D_53_54";
  --!SLV, define your ODILE board as a slave board.
  constant CMD_SET_SLV          : command := X"53_4C_56";
   --!EXS, enable start_sequencer signal coming from outside (RJ45).
  constant CMD_ENABLE_EXT_START_SEQ          : command := X"45_58_53";
   --!DXS, enable start_sequencer signal coming from outside (RJ45).
  constant CMD_DISABLE_EXT_START_SEQ          : command := X"44_58_53";
  --!SEX, Start EXposure. Starts clocking to take an image
  constant CMD_START_SEQ        : command := X"53_45_58";
  --!AEX, Abort EXposure. Stops clocking to take an image.
  constant CMD_STOP_SEQ         : command := X"41_45_58";
  --!STS, STep Sequencer. Steps the sequencer
  constant CMD_STEP_SEQ         : command := X"53_54_53";
  --!GCT, Get Compile Time. Tells the ODILE to report the compile time of
the firmware, if it was compiled with that option.
  constant CMD_GET_TS           : command := X"47_43_54";
```

```
  --!GCL, Get Command List. Returns the list of currently supported
commands.
  constant CMD_GET_CMD_LIST     : command := X"47_43_4C";
  --!GUT, Get UpTime. Gets seconds board has been up since last reset.
Note: this is derived from counting clock cycles, so it will not be
accurate.
  constant CMD_GET_UPTIME       : command := X"47_55_54";
  --!INV, INValid. Indicates that the previous command was not
recognized.
  constant CMD_INVALID          : command := X"49_4E_56";
  --!DON, DONe. Sent by the ODILE to indicate that the previous command
has finished. Not supported for all commands.
  constant CMD_DONE             : command := X"44_4F_4E";
  --!ERR, ERRor. Sent by the ODILE to indicate an error with the previous
command.
  constant CMD_ERROR            : command := X"45_52_52";
  --!RDP, ReaD Program. Tells the ODILE to read back the sequencer
program memory
  constant CMD_READ_PROG        : command := X"52_44_50";
  --!RDT, ReaD Timing. Reads back the sequencer timing memory
  constant CMD_READ_TIME        : command := X"52_44_54";
  --!RDO, ReaD Output. Reads the sequencer output memory
  constant CMD_READ_OUT         : command := X"52_44_4F";
  --!GEC, Get Error Code. Tells the ODILE to report the current error
code (0x0 if no error)
  constant CMD_GET_ERROR        : command := X"47_45_43";
  --!CER, Clear ERror. Clears the current error code.
  constant CMD_CLEAR_ERROR      : command := X"43_45_43";
  --!RSC, ReSet Cabac. Triggers the "reset_cabac" signal.
  constant CMD_RESET_CABAC      : command := X"52_53_43";
  --!RDC, Read Cabac. Reads the current value of the
"Reg32b_cabacspi_ReadOnly" register.
  constant CMD_READ_CABAC       : command := X"52_44_43";

  --!RSD, ReSet DAC. Triggers the "reset_dac" signal.
  constant CMD_RESET_DAC      : command := X"52_53_44";
  --!RDD, Read DAC. Reads the current value of the
"Reg32b_dacspi_ReadOnly" register.
  constant CMD_READ_DAC       : command := X"52_44_44";


  --!RSV, ReSet DIP. Triggers the "reset_dip" signal.
  constant CMD_RESET_DIP      : command := X"52_53_56";
  --!RDV, Read DIP. Reads the current value of the
"Reg32b_dipspi_ReadOnly" register.
  constant CMD_READ_DIP       : command := X"52_44_56";

  --!CBS, Clear MAX14802. Triggers the "reset_MAX14802" signal.
  constant CMD_RESET_MAX14802      : command := X"43_42_53";
  --!RDM, Read MAX14802. Reads the current value of the
"Reg32b_MAX14802serial_ReadOnly" register.
  constant CMD_READ_MAX14802       : command := X"52_44_4D";

  -- Enables
  -- B3018_16 : VSUB Generator Block;
  -- B3018_17_1, B3018_17_2, B3018_17_3, B3018_17_4 : Bias Generator
Block (There are four identical channels);
  -- B3018_18_1, B3018_18_2: Bias Generator with Offset Block (There are
two identical channels);
```

```
 --!EHV, Enable the 100volts VSUB supply (High voltage).
constant CMD_ENABLE_HV      : command := X"45_48_56";

 --!VR0, Enable the "TPS7A330" signal.
constant CMD_ENABLE_VR0      : command := X"56_52_30";

 --!VR1, Enable the "TPS7A330" signal.
constant CMD_ENABLE_VR1      : command := X"56_52_31";

 --!VR2, Enable the "TPS7A330" signal.
constant CMD_ENABLE_VR2      : command := X"56_52_32";

 --!VR3, Enable the "TPS7A330" signal.
constant CMD_ENABLE_VR3      : command := X"56_52_33";

 --!VR4, Enable the "TPS7A330" signal.
constant CMD_ENABLE_VR4      : command := X"56_52_34";

 --!VR5, Enable the "TPS7A330" signal.
constant CMD_ENABLE_VR5      : command := X"56_52_35";

 --!VR6, Enable the "TPS7A330" signal.
constant CMD_ENABLE_VR6      : command := X"56_52_36";

 --!DHV, Disable the 100volts VSUB supply (High voltage).
constant CMD_DISABLE_HV      : command := X"44_48_56";

 --!0RV, DISABLE the "TPS7A330" signal.
constant CMD_DISABLE_VR0      : command := X"30_52_56";

 --!1RV, DISABLE the "TPS7A330" signal.
constant CMD_DISABLE_VR1      : command := X"31_52_56";

 --!2RV, DISABLE the "TPS7A330" signal.
constant CMD_DISABLE_VR2      : command := X"32_52_56";

 --!3RV, DISABLE the "TPS7A330" signal.
constant CMD_DISABLE_VR3      : command := X"33_52_56";

 --!4RV, DISABLE the "TPS7A330" signal.
constant CMD_DISABLE_VR4      : command := X"34_52_56";

 --!5RV, DISABLE the "TPS7A330" signal.
constant CMD_DISABLE_VR5      : command := X"35_52_56";

 --!6RV, DISABLE the "TPS7A330" signal.
constant CMD_DISABLE_VR6      : command := X"36_52_56";

 --!CON, Enable the "TPS7A330" signal.
constant CMD_ENABLE_CLEAR      : command := X"43_4F_4E";



 --!LON, Enable the "TPS7A330" signal.
constant CMD_ENABLE_LATCH      : command := X"4C_4F_4E";

--!EPS, Enable Front-end positive supply.
constant CMD_ENABLE_POS      : command := X"45_50_53";
--!ENS, Enable Front-end negative supply.
constant CMD_ENABLE_NEG      : command := X"45_4E_53";
--!DPS, Disable Front-end positive supply.
```

```
  constant CMD_DISABLE_POS      : command := X"44_50_53";
  --!DNS, Disable Front-end negative supply.
  constant CMD_DISABLE_NEG      : command := X"44_4E_53";

  --!RCR, Read CRoc. Initiates a request to the croc interface to read
the CROC register. To read the result, use the GCR command.
  constant CMD_READ_CROC        : command := X"52_43_52";
  --!GCR, Get CRoc. Reads the 96-bit croc register to the PC.
  constant CMD_GET_CROC         : command := X"47_43_52";
  --!RDB, ReaD configuration Blocks. Reads the all the configuration
register memories.
  constant CMD_READ_CONFIG      : command := X"52_44_42";
  --!ERS, ERase Sequencer. Erases all the sequencer memories (sets them
to 0x0).
  constant CMD_ERASE_SEQ        : command := X"45_52_53";
  --!RDF, ReaD indirect Function. Reads the indirect function sequencer
memory.
  constant CMD_READ_INDF        : command := X"52_44_46";
  --!RDR, ReaD indirect Reps. Reads the indirect function repitition
sequencer memory.
  constant CMD_READ_INDR        : command := X"52_44_52";
  --!RDA, ReaD indirect Address. Reads the indirect function address
sequencer memory.
  constant CMD_READ_INDSA       : command := X"52_44_41";
  --!RDS, Read indirect Sub reps. Reads the indirect subfunction
repitition sequencer memory.
  constant CMD_READ_INDSR       : command := X"52_44_53";
  ------------------------------------------------------------------------
------
  -- EPCQIO commands
  ------------------------------------------------------------------------
------
  --!EWR, EPCQ WRite. Writes the current EPCQ FIFO buffer contents to the
EPCQ flash, starting at the currently set address. Requires an 8-bit
prefix to specify the number of 32-bit words to write.
  constant CMD_EPCQ_WRITE       : command := X"45_57_52";
  --!ERD, EPCQ ReaD. Reads the EPCQ flash memory, starting at the
currently set address.
  constant CMD_EPCQ_READ        : command := X"45_52_44";
  --!ERB, EPCQ Reset Buffers. Clears the EPCQIO read/write buffers
  constant CMD_EPCQ_CLEAR       : command := X"45_52_42";
  --!ESA, EPCQ Set Address. Sets the start address for read/write
commands. Two-word command, the address should be sent in the second
word.
  constant CMD_EPCQ_SETA        : command := X"45_53_41";
  --!E4B, EPCQ enable 4 Byte. Debug command to enable 4 byte addressing
in the Altera EPCQIO block.
  constant CMD_EPCQ_EN4B        : command := X"45_34_42";
  --!ESE, EPCQ Sector Erase. Erase the sector containing the currently
set address.
  constant CMD_EPCQ_ERASE_SEC   : command := X"45_53_45";
  ------------------------------------------------------------------------
------
  -- Remote update commands
  ------------------------------------------------------------------------
------
  --!RUA, Remote Update Address. Sets the start address to load firmware
from. Two word command with the second word being the start address.
  constant CMD_RU_ADDRESS       : command := X"52_55_41";
  --!\brief RUR, Remote Update Reconfig. Starts the process of loading
the firmware from the address sset with RUA.
```

```
  --!Note that if this address does not
  --!contain a valid firmware image, the FPGA will automatically try to
load the factory firmware from 0x0, if that also does not contain a valid
  --!firmware the FPGA will need to be manually reflashed.
  constant CMD_RU_RECONFIG      : command := X"52_55_52";
  --!RUL, Remote Update Reload. Debug command that rereads the Altera
remote update block parameters.
  constant CMD_RU_REREAD        : command := X"52_55_4C";
  ------------------------------------------------------------------------
------
  -- Configuration loading commands
  ------------------------------------------------------------------------
------
  --!\brief LDC, LoaD Config. Loads configuration register settings from
one of the ten pages stored on the flash memory. The page should be
specified in the
  --!8 bit prefix. Valid pages range from 0x0 to 0xA.
  constant CMD_CONF_LOAD        : command := X"4C_44_43";
  ------------------------------------------------------------------------
------
  -- Monitoring/misc commands
  ------------------------------------------------------------------------
------
  --!SCM, Start CCD Monitoring. Activates the start_monitoring signal.
  constant CMD_START_MONITORING : command := X"53_43_4d";
  --!GCM, Get CCD Monitoring. Reads the 18x32 bit register of the
top_monitoring results.
  constant CMD_GET_MONITORING   : command := X"47_43_4d";
  --!SSW, Set SWitches. Sets the hardware switches. The 8 switch values
should be specified in binary in the 8-bit prefix.
  constant CMD_SET_SWITCHES     : command := X"53_53_57";
  --!GSW, Get SWitches. Gets the hardware switches.
  constant CMD_GET_SWITCHES     : command := X"47_53_57";
  --!Returns true if command is valid
  function is_valid_command(cmd : command) return boolean;

  type command_array is array(natural range <>) of command;
  --!List of valid commands. Update this if you add a command above
  constant VALID_COMMANDS : command_array := (CMD_SET_MST,
CMD_SET_SLV,CMD_SET_IP, CMD_TOGGLE_PONG, CMD_GET_CHIP_ID,
                                              CMD_ENABLE_EXT_START_SEQ,
CMD_DISABLE_EXT_START_SEQ, CMD_START_SEQ, CMD_STOP_SEQ, CMD_STEP_SEQ,
CMD_GET_TS, CMD_GET_CMD_LIST,
                                              CMD_READ_PROG,
CMD_READ_TIME, CMD_READ_OUT, CMD_GET_ERROR, CMD_CLEAR_ERROR,
                                              CMD_RESET_DAC,
CMD_READ_DAC, CMD_RESET_DIP, CMD_READ_DIP, CMD_RESET_MAX14802,
CMD_READ_MAX14802,

CMD_ENABLE_HV, CMD_ENABLE_VR0, CMD_ENABLE_VR1, CMD_ENABLE_VR2,
CMD_ENABLE_VR3, CMD_ENABLE_VR4, CMD_ENABLE_VR5, CMD_ENABLE_VR6,

CMD_DISABLE_HV, CMD_DISABLE_VR0, CMD_DISABLE_VR1, CMD_DISABLE_VR2,
CMD_DISABLE_VR3, CMD_DISABLE_VR4, CMD_DISABLE_VR5, CMD_DISABLE_VR6,

CMD_ENABLE_CLEAR, CMD_ENABLE_LATCH,
                                              CMD_ENABLE_POS,
CMD_ENABLE_NEG, CMD_DISABLE_POS, CMD_DISABLE_NEG,
                                    CMD_READ_CONFIG, CMD_ERASE_SEQ,
CMD_READ_INDF,
```

```
                                                CMD_READ_INDR,
CMD_READ_INDSA, CMD_READ_INDSR, CMD_EPCQ_WRITE,
                                                CMD_EPCQ_READ,
CMD_EPCQ_CLEAR, CMD_EPCQ_SETA, CMD_EPCQ_EN4B,
                                                CMD_EPCQ_ERASE_SEC,
CMD_RU_ADDRESS, CMD_RU_RECONFIG, CMD_RU_REREAD,
                                                CMD_CONF_LOAD,
CMD_START_MONITORING, CMD_GET_MONITORING,
                                                CMD_SET_SWITCHES,
CMD_GET_SWITCHES, CMD_GET_UPTIME);

end package ODILE_command_list;

package body ODILE_command_list is
  --!Scans over list of valid commands and returns true if cmd is in that
list.
  function is_valid_command (cmd : command) return boolean is
  begin
    for I in VALID_COMMANDS'range loop
      if cmd = VALID_COMMANDS(I) then
        return true;
      end if;
    end loop;
    return false;
  end function is_valid_command;

end package body ODILE_command_list;

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.math_real.all;
use work.eth_common.all;
use work.ODILE_command_list.all;

--!\brief Controller that parses data from the Ethernet interface and
--!interprets any commands from the DAQ.

entity odile_controller is
  port (
    clock                 : in  std_logic;
    reset                 : in  std_logic;
    ------------------------------------------------------------------------
------
    --!\name Inputs from ethernet
    --!\{
    ------------------------------------------------------------------------
------
    data_in               : in  std_logic_vector(31 downto 0);
    data_valid            : in  std_logic;
    data_port             : in  std_logic_vector(15 downto 0);
    data_addr             : in  std_logic_vector(79 downto 0);
    source_iface          : in  std_logic_vector(3 downto 0);
    --!\}
    ------------------------------------------------------------------------
------
    --!\name Outputs to PLLs (set board as master or slave)
    --!\{
    ------------------------------------------------------------------------
------
    set_master            : out std_logic;
```

```
    set_slave              : out std_logic;
    enable_ext_start_seq   : out std_logic;
    disable_ext_start_seq  : out std_logic;
    --!\}
    ------------------------------------------------------------------------
------
    --!\name Outputs to sequencer
    --!\{
    ------------------------------------------------------------------------
------
    start_sequence         : out std_logic;
    step_sequence          : out std_logic;
    stop_sequence          : out std_logic;
    --!\}
    ------------------------------------------------------------------------
------
    --!\name Outputs to ethernet_ccdcontrol_interface
    --!\{
    ------------------------------------------------------------------------
------
    read_triggers          : out std_logic_vector(15 downto 0);
    clear_error            : out std_logic;
    reset_cabac            : out std_logic;
      reset_dac                : out std_logic;
      reset_dip                : out std_logic;
      reset_MAX14802           : out std_logic;
    erase_sequencer        : out std_logic;
      enable_100VP           : out std_logic;
      enable_VR0             : out std_logic;
      enable_VR1             : out std_logic;
      enable_VR2             : out std_logic;
      enable_VR3             : out std_logic;
      enable_VR4             : out std_logic;
      enable_VR5             : out std_logic;
      enable_VR6             : out std_logic;
      enable_CLEAR           : out std_logic;
      enable_LATCH           : out std_logic;
    --!\}

    ------------------------------------------------------------------------
------
    --!\name Outputs to Front-end supply regulators
    --!\{
    ------------------------------------------------------------------------
------
        enable_pos             : out std_logic;
        enable_neg             : out std_logic;
    --!\}

    ------------------------------------------------------------------------
------
    --!\name Outputs to EPCQIO/remote update block
    --!\{
    ------------------------------------------------------------------------
------
    epcqio_read_data       : out std_logic;
    epcqio_write_data      : out std_logic;
    epcqio_enable_4byte    : out std_logic;
    epcqio_erase_sector    : out std_logic;
    epcqio_clear_buffers   : out std_logic;
    epcqio_address         : out std_logic_vector(31 downto 0);
```

```
    epcqio_num_words       : out std_logic_vector(6 downto 0);
    --!\}
    ------------------------------------------------------------------------
------
    --!\name Outputs to remote update block
    --!\{
    ------------------------------------------------------------------------
------
    ru_do_reconfig         : out std_logic;
    ru_application_address : out std_logic_vector(23 downto 0);
    ru_reread_params       : out std_logic;
    --!\}
    ------------------------------------------------------------------------
------
    --!\name Miscellaneous outputs
    --!\{
    ------------------------------------------------------------------------
------
    --!Signals the top_monitoring block to sample it's ADCs
    start_monitoring       : out std_logic;
    --!Tells the Ethernet interface to read the 18x32 bit monitoring
register
    read_monitoring        : out std_logic;
    --!Signals the CROC controller to read the current CROC status.
    read_croc              : out std_logic;
    --!Whether to send an acknowledgment of the command to the DAQ
    send_cmd_ack           : out std_logic;
    --!Command to acknowledge
    cmd_to_ack             : out std_logic_vector(31 downto 0);
    --!Read configuration from flash
    cm_load_config         : out std_logic;
    --!Config page to load (from 0-9)
    cm_config_page         : out std_logic_vector(3 downto 0);
    --!Read our current configuration registers
    read_config            : out std_logic;
    --!Interface that generated the command
    reply_iface            : out std_logic_vector(3 downto 0);
    reply_addr             : out std_logic_vector(79 downto 0);
    --!Hardware switches
    switches               : out std_logic_vector(7 downto 0);
    enable_pong            : out std_logic;
    board_daq_id : out std_logic_vector(7 downto 0);
    chip_id_lsb : in std_logic_vector(31 downto 0)
    );
--!\}
end entity odile_controller;

architecture vhdl_rtl of odile_controller is

  signal data_in_reg                              : std_logic_vector(31
downto 0) := (others => '0');
  signal data_port_reg                            : std_logic_vector(15
downto 0) := (others => '0');
  signal data_valid_reg                           : std_logic
:= '0';
  signal data_addr_reg                            : std_logic_vector(79
downto 0) := (others => '0');
  signal source_iface_reg                         : std_logic_vector(3
downto 0)  := (others => '0');
  signal read_program, read_timing, read_output : std_logic
:= '0';
```

```
  signal read_cabac_reg                          : std_logic
:= '0';
  signal read_dac_reg                            : std_logic
:= '0';
  signal read_dip_reg                            : std_logic
:= '0';
  signal read_MAX14802_reg                        : std_logic
:= '0';
  signal read_ind_func, read_ind_rep             : std_logic
:= '0';
  signal read_ind_sub_add, read_ind_sub_rep      : std_logic
:= '0';
  signal epcqio_address_reg                      : std_logic_vector(31
downto 0) := (others => '0');
  signal second_word                             : boolean
:= false;
  signal epcqio_numwords_reg                     : std_logic_vector(6
downto 0)  := (others => '0');
  signal ru_reconfig_reg                         : std_logic
:= '0';
  signal ru_address_reg                          : std_logic_vector(23
downto 0) := (others => '1');
  signal ru_reread_params_reg                    : std_logic
:= '0';
  signal cm_load_config_reg                      : std_logic
:= '0';
  signal cm_config_page_reg                      : std_logic_vector(3
downto 0)  := (others => '0');
  signal start_monitoring_reg                    : std_logic
:= '0';
  signal read_monitoring_reg                     : std_logic
:= '0';
  signal switches_reg                            : std_logic_vector(7
downto 0)  := (others => '0');
  signal read_croc_reg                           : std_logic
:= '0';
  signal get_croc_reg                            : std_logic
:= '0';

begin

  --Uses a single read_triggers bus to our ccdcontrol interface because
we have many
  --signals and I don't want to code every single one by hand.
  read_triggers(0)           <= read_program;
  read_triggers(1)           <= read_timing;
  read_triggers(2)           <= read_output;
  read_triggers(3)           <= read_ind_func;
  read_triggers(4)           <= read_ind_rep;
  read_triggers(5)           <= read_ind_sub_add;
  read_triggers(6)           <= read_ind_sub_rep;
  read_triggers(7)           <= read_cabac_reg;
  read_triggers(8)           <= get_croc_reg;
  read_triggers(9)           <= read_dac_reg;
  read_triggers(10)          <= read_dip_reg;
  read_triggers(11)          <= read_MAX14802_reg;
  read_triggers(15 downto 12) <= (others => '0');
  epcqio_address             <= epcqio_address_reg;
  epcqio_num_words           <= epcqio_numwords_reg;
  ru_do_reconfig             <= ru_reconfig_reg;
  ru_application_address     <= ru_address_reg;
```

```
  ru_reread_params           <= ru_reread_params_reg;
  cm_load_config             <= cm_load_config_reg;
  cm_config_page             <= cm_config_page_reg;
  start_monitoring           <= start_monitoring_reg;
  read_monitoring            <= read_monitoring_reg;
  switches                   <= switches_reg;
  read_croc                  <= read_croc_reg;
  ------------------------------------------------------------------------
------
  --!Interprets commands from our DAQ into triggers for other components.
  ------------------------------------------------------------------------
------
  command_parser : process (clock, reset)
    variable prev_cmd : command := (others => '0');
  begin
    if reset = '1' then
      --Default states
      board_daq_id         <= X"05";
      enable_pong          <= '1';
      set_master           <= '0';
      set_slave            <= '0';
      enable_ext_start_seq <= '0';
      disable_ext_start_seq<= '0';
      start_sequence       <= '0';
      stop_sequence        <= '0';
      step_sequence        <= '0';
      read_program         <= '0';
      read_timing          <= '0';
      read_output          <= '0';
      read_ind_rep         <= '0';
      read_ind_func        <= '0';
      read_ind_sub_add     <= '0';
      read_ind_sub_rep     <= '0';
      reset_cabac          <= '0';
        reset_dac            <= '0';
        reset_dip            <= '0';
        reset_MAX14802       <= '0';
          enable_100VP         <= '0';
        enable_VR0           <= '0';
        enable_VR1           <= '0';
        enable_VR2           <= '0';
        enable_VR3           <= '0';
        enable_VR4           <= '0';
        enable_VR5           <= '0';
        enable_VR6           <= '0';
        enable_CLEAR         <= '0';
        enable_LATCH         <= '1';
      enable_pos           <= '0';
      enable_neg           <= '0';
      data_in_reg          <= (others => '0');
      data_port_reg        <= (others => '0');
      data_valid_reg       <= '0';
      clear_error          <= '0';
      erase_sequencer      <= '0';
      epcqio_clear_buffers <= '0';
      epcqio_write_data    <= '0';
      epcqio_read_data     <= '0';
      epcqio_erase_sector  <= '0';
      epcqio_enable_4byte  <= '0';
      epcqio_address_reg   <= (others => '0');
      second_word          <= false;
```

```
    epcqio_numwords_reg  <= (others => '0');
    ru_address_reg       <= (others => '1');
    ru_reconfig_reg      <= '0';
    ru_reread_params_reg <= '0';
    cm_load_config_reg   <= '0';
    start_monitoring_reg <= '0';
    read_monitoring_reg  <= '0';
    read_cabac_reg       <= '0';
      read_dac_reg         <= '0';
      read_dip_reg         <= '0';
      read_MAX14802_reg     <= '0';
    read_croc_reg        <= '0';
    switches_reg         <= (others => '0');
    get_croc_reg         <= '0';

  elsif rising_edge(clock) then
    --Register data in to reduce timing requirements
    data_in_reg          <= data_in;
    data_port_reg        <= data_port;
    data_valid_reg       <= data_valid;
    data_addr_reg        <= data_addr;
    source_iface_reg     <= source_iface;
    --Default states
    set_master           <= '0';
    set_slave            <= '0';
    enable_ext_start_seq <= '0';
    disable_ext_start_seq<= '0';
    start_sequence       <= '0';
    stop_sequence        <= '0';
    step_sequence        <= '0';
    read_program         <= '0';
    read_timing          <= '0';
    read_output          <= '0';
    read_ind_rep         <= '0';
    read_ind_func        <= '0';
    read_ind_sub_add     <= '0';
    read_ind_sub_rep     <= '0';
    reset_cabac          <= '0';
      reset_dac            <= '0';
      reset_dip            <= '0';
      reset_MAX14802        <= '0';
      -- enable_VR0           <= '0';
      -- enable_VR1           <= '0';
      -- enable_VR2           <= '0';
      -- enable_VR3           <= '0';
      -- enable_VR4           <= '0';
      -- enable_VR5           <= '0';
      -- enable_VR6           <= '0';
      enable_CLEAR         <= '0';
      enable_LATCH         <= '1';

    send_cmd_ack         <= '0';
    clear_error          <= '0';
    read_config          <= '0';
    erase_sequencer      <= '0';
    epcqio_clear_buffers <= '0';
    epcqio_write_data    <= '0';
    epcqio_read_data     <= '0';
    epcqio_erase_sector  <= '0';
    epcqio_enable_4byte  <= '0';
    cmd_to_ack           <= (others => '0');
```

```
      ru_reconfig_reg      <= '0';
      ru_reread_params_reg <= '0';
      cm_load_config_reg   <= '0';
      start_monitoring_reg <= '0';
      read_monitoring_reg  <= '0';
      read_cabac_reg       <= '0';
        read_dac_reg         <= '0';
        read_dip_reg         <= '0';
        read_MAX14802_reg     <= '0';
      read_croc_reg        <= '0';
      get_croc_reg         <= '0';
      switches_reg         <= switches_reg;  --Explicitly register the
switches
      --We can set this to false because it is updated next clock cycle,
will
      --be set to true if we have a two-word command.
      second_word          <= false;

      --If the data is coming into the command UDP port and is valid,
parse it
      if (data_port_reg = UDP_PORT_COMMAND and
          data_valid_reg = '1') then
        case data_in_reg(23 downto 0) is
          ------------------------------------------------------------------
------
          -- Board ID control
          ------------------------------------------------------------------
------
          when CMD_SET_IP =>
            second_word <= true;
            prev_cmd    := CMD_SET_IP;
            board_daq_id <= data_in_reg(31 downto 24);
          when CMD_TOGGLE_PONG =>
            enable_pong <= not enable_pong;
          ------------------------------------------------------------------
------
          -- Board state control
          ------------------------------------------------------------------
------
          when CMD_SET_MST =>
            set_master <= '1';
          when CMD_SET_SLV =>
            set_slave <= '1';
          when CMD_ENABLE_EXT_START_SEQ =>
            enable_ext_start_seq <= '1';
          when CMD_DISABLE_EXT_START_SEQ =>
            disable_ext_start_seq <= '1';
          ------------------------------------------------------------------
------
          -- Sequence controls
          ------------------------------------------------------------------
------
          when CMD_START_SEQ =>
            start_sequence <= '1';
          when CMD_STOP_SEQ =>
            stop_sequence <= '1';
          when CMD_STEP_SEQ =>
            step_sequence <= '1';
          when CMD_CLEAR_ERROR =>
            clear_error <= '1';
```

```
          ------------------------------------------------------------------
------
          -- Reads for various sequencer memories
          ------------------------------------------------------------------
------
          when CMD_READ_PROG =>
            read_program <= '1';
          when CMD_READ_TIME =>
            read_timing <= '1';
          when CMD_READ_OUT =>
            read_output <= '1';
          when CMD_READ_INDF =>
            read_ind_func <= '1';
          when CMD_READ_INDR =>
            read_ind_rep <= '1';
          when CMD_READ_INDSA =>
            read_ind_sub_add <= '1';
          when CMD_READ_INDSR =>
            read_ind_sub_rep <= '1';
          --Read all configuration blocks
          when CMD_READ_CONFIG =>
            read_config <= '1';
          -- Erase our sequencer
          when CMD_ERASE_SEQ =>
            erase_sequencer <= '1';
          -- Reset CABAC
          when CMD_RESET_CABAC =>
            reset_cabac <= '1';
          -- Read CABAC status register
          when CMD_READ_CABAC =>
            read_cabac_reg <= '1';
             -- Reset DAC
            when CMD_RESET_DAC =>
            reset_dac <= '1';
          -- Read DAC status register
          when CMD_READ_DAC =>
            read_dac_reg <= '1';

              -- Reset DIP
            when CMD_RESET_DIP =>
            reset_dip <= '1';

          -- Read DIP status register
            when CMD_READ_DIP =>
            read_dip_reg <= '1';

             -- Reset MAX14802
            when CMD_RESET_MAX14802 =>
            reset_MAX14802 <= '1';

             -- Read MAX14802 status register
            when CMD_READ_MAX14802 =>
            read_MAX14802_reg <= '1';

                -- Enable 100VP VSUB SUPPLY
                 when CMD_ENABLE_HV =>
            enable_100VP <= '1';

           -- Enable VR0
            when CMD_ENABLE_VR0 =>
            enable_VR0 <= '1';
```

```
-- Enable VR1
 when CMD_ENABLE_VR1 =>
enable_VR1 <= '1';

-- Enable VR2
 when CMD_ENABLE_VR2 =>
enable_VR2 <= '1';

-- Enable VR3
 when CMD_ENABLE_VR3 =>
enable_VR3 <= '1';

-- Enable VR4
 when CMD_ENABLE_VR4 =>
enable_VR4 <= '1';

-- Enable VR5
 when CMD_ENABLE_VR5 =>
enable_VR5 <= '1';

-- Enable VR6
 when CMD_ENABLE_VR6 =>
enable_VR6 <= '1';

-- Enable CLEAR
 when CMD_ENABLE_CLEAR =>
enable_CLEAR <= '1';

-- Enable LATCH
 when CMD_ENABLE_LATCH =>
enable_LATCH <= '0';

     -- Disable 100VP VSUB supply
      when CMD_DISABLE_HV =>
enable_100VP <= '0';

-- Disable VR0
 when CMD_DISABLE_VR0 =>
enable_VR0 <= '0';

-- Disable VR1
 when CMD_DISABLE_VR1 =>
enable_VR1 <= '0';

-- Disable VR2
 when CMD_DISABLE_VR2 =>
enable_VR2 <= '0';

-- Disable VR3
 when CMD_DISABLE_VR3 =>
enable_VR3 <= '0';

-- Disable VR4
 when CMD_DISABLE_VR4 =>
enable_VR4 <= '0';

-- Disable VR5
 when CMD_DISABLE_VR5 =>
enable_VR5 <= '0';
```

```
            -- Disable VR6
             when CMD_DISABLE_VR6 =>
             enable_VR6 <= '0';


                -- Enable POS FRONT-END SUPPLY
                 when CMD_ENABLE_POS =>
             enable_pos <= '1';

                -- Enable NEG FRONT-END SUPPLY
                 when CMD_ENABLE_NEG =>
             enable_neg <= '1';

                -- Disable POS FRONT-END SUPPLY
                 when CMD_DISABLE_POS =>
             enable_pos <= '0';

                -- Disable NEG FRONT-END SUPPLY
                 when CMD_DISABLE_NEG =>
             enable_neg <= '0';



          ------------------------------------------------------------
------
          -- Commands to EPCQIO/remote update
          ------------------------------------------------------------
------
          when CMD_EPCQ_READ =>
            epcqio_read_data    <= '1';
            epcqio_numwords_reg <= data_in_reg(30 downto 24);
          when CMD_EPCQ_WRITE =>
            epcqio_write_data   <= '1';
            epcqio_numwords_reg <= data_in_reg(30 downto 24);
          when CMD_EPCQ_CLEAR =>
            epcqio_clear_buffers <= '1';
          when CMD_EPCQ_SETA =>
            second_word <= true;
            prev_cmd    := CMD_EPCQ_SETA;
          when CMD_EPCQ_EN4B =>
            epcqio_enable_4byte <= '1';
          when CMD_EPCQ_ERASE_SEC =>
            epcqio_erase_sector <= '1';
          when CMD_RU_RECONFIG =>
            ru_reconfig_reg <= '1';
          when CMD_RU_ADDRESS =>
            second_word <= true;
            prev_cmd    := CMD_RU_ADDRESS;
            --Presets for 'F' (factory) and 'A' (application) addresses
            if (data_in_reg(31 downto 24) = X"46") then
              second_word    <= false;
              ru_address_reg <= X"000000";
            elsif (data_in_reg(31 downto 24) = X"41") then
              second_word    <= false;
              ru_address_reg <= X"100000";
            end if;
          when CMD_RU_REREAD =>
            ru_reread_params_reg <= '1';
          when CMD_CONF_LOAD =>
            cm_config_page_reg <= data_in_reg(27 downto 24);
            cm_load_config_reg <= '1';
```

```
          --Start reading monitoring status
          when CMD_START_MONITORING =>
            start_monitoring_reg <= '1';
          when CMD_GET_MONITORING =>
            read_monitoring_reg <= '1';
          when CMD_SET_SWITCHES =>
            switches_reg <= data_in_reg(31 downto 24);
          when CMD_READ_CROC =>
            read_croc_reg <= '1';
          when CMD_GET_CROC =>
            get_croc_reg <= '1';
          when others =>
        --Insert error generating here, maybe
        end case;
        --Send an acknowledgement of the command
        send_cmd_ack <= '1';
        cmd_to_ack   <= data_in_reg;
        reply_iface  <= source_iface_reg;
        reply_addr   <= data_addr_reg;

        -- Reads the values from the Set switches command
        if data_in_reg(23 downto 0) = CMD_GET_SWITCHES then
          cmd_to_ack(31 downto 24) <= switches_reg;
        end if;

        --For two word commands
        if second_word then
          if prev_cmd = CMD_EPCQ_SETA then
            send_cmd_ack       <= '0';
            epcqio_address_reg <= data_in_reg;
          elsif prev_cmd = CMD_RU_ADDRESS then
            send_cmd_ack   <= '0';
            ru_address_reg <= data_in_reg(23 downto 0);
          elsif prev_cmd = CMD_SET_IP then
            send_cmd_ack       <= '0';
            if (data_in_reg /= chip_id_lsb) then
              board_daq_id <= X"00";
            end if;
          end if;
          second_word <= false;
        end if;

      end if;
    end if;
  end process;

end architecture vhdl_rtl;
```