```
--Signal naming conventions
-- ON- chip signals are not preficed
-- Upper part of the on-board bus signale preficed with             iu_
-- Lower part of the on-board bus signals preficed with             il_
-- Upper part of pc bus                                             pu_
-- Lower part of pc bus                                             pl_
-- Backplane bus                                                    bp_
-- fiber output                                                     fo_
-- fiber input                                                      fi_
--
--
-- take care of reading data from the onboard bus
-- direction: ibus_upper (BP data read) to FO_action
--
library ieee;
use ieee.std_logic_1164.all;
use work.numeric_std.all;

entity read_from_ibus is

   port (
    clk   : in  std_logic;
    reset : in  std_logic;
    iu_req   : in std_logic;
    iu_ack   : buffer  std_logic;
    iu_io    : in std_logic_vector(15 downto 0);
    data  : out  std_logic_vector(15 downto 0);
    req  : out  std_logic;
    ack  : in std_logic);

end read_from_ibus;

architecture read_from_ibus_arch of read_from_ibus is
type iu_read_state is (idle,hs,iu_hs1,iu_hs2);
signal iu_read : iu_read_state;
signal timeout : unsigned(3 downto 0);
begin  -- write_to_ibus_arch

  iu_reader: process (clk, reset)
  begin  -- process iu_writer
    if reset = '1' then                 -- asynchronous reset (active low)

    elsif clk'event and clk = '1' then  -- rising clock edge
      req <= '0';
      iu_ack <= '0';
      case iu_read is
        when idle =>
          if (iu_req = '1') then
            data <= iu_io;
            iu_ack <= '1';
            iu_read <= hs;
            timeout <= "0000";
          else
            iu_read <= idle;
          end if;
        when hs =>
          iu_ack <= '1';
          req <= '1';
          if (iu_req = '0') then
            iu_read <= iu_hs1;
            timeout <= "0000";
          else
            if (timeout = "1111") then
              iu_read <= iu_hs1;
              timeout <= "0000";
            else
              timeout <= timeout+1;
            end if;
          end if;
        when iu_hs1 =>
          req <= '1';
          if (ack = '1') then
            req <= '0';
            iu_read <= iu_hs2;
          else
            if (timeout = "1111") then
              iu_read <= idle;
            else
              timeout <= timeout+1;
            end if;
          end if;
        when iu_hs2 =>                  -- no timeout here because by this
                                        -- stage we know there is something
                                        -- at the other end
          if (ack = '0') then
            iu_read <= idle;
          end if;
        when others =>
          iu_read <= idle;
      end case;
    end if;
  end process iu_reader;
end read_from_ibus_arch;

-- This block reads data from the FI when addressed. It captures and
-- acts on command data and places other data or address info in the on
-- chip buffers.
-- Each transaction typically consists of two transfers, an address transfer
-- followed by a data transfer. There are a few exceptions where only an
-- address transfer is required, namelly a soft reset and switching to
-- loopback mode.
-- When the code detects an address transfer it check to see if it is being
-- addressed. If it is then enables itself. Thie chip stays enabled untill
-- another address transfer occurs. In this way the chip can accept follow
-- up data transfers.
--
library ieee;
use ieee.std_logic_1164.all;
use work.numeric_std.all;
entity read_from_FI is  -- to BP direction
  -- note internal upper bus is used - from FI=PC=upper
  port (
    clk       : in     std_logic;
    reset     : in     std_logic;
    req        : in     std_logic;
    ack        : out    std_logic;
    data     : in     std_logic_vector(15 downto 0);
    il_io      : out std_logic_vector(15 downto 0);
    address   : buffer std_logic_vector(7 downto 0);
    il_req       : buffer    std_logic;
    il_ack       : in     std_logic;
    i_am_remote:in     std_logic;
    my_data_address:in    std_logic_vector(3 downto 0);
    my_ctrl_address:in    std_logic_vector(3 downto 0);
 loopback_state:buffer std_logic;
    request_reset:out     std_logic;
    FI_i_am_addressed: buffer std_logic;
    receive_enabled   : buffer std_logic
    );

end read_from_FI;

architecture read_from_FI_arch of read_from_FI is
type FI_read_type is (idle,am_i_addressed,FI_hs_for_address,FI_hs_for_data,
                      FI_use_ctrl_data,internal_ack1,internal_ack2);
signal FI_read_state : FI_read_type;
signal timeout : unsigned(3 downto 0);
signal this_is_a_ctrl_transaction : std_logic;
begin  -- read_from_FI_arch

```

```
  -- purpose: read from the fiber receiver
  -- type   : sequential
  -- inputs : clk, reset
  -- outputs:
  read_FI: process (clk, reset)
  begin  -- process read_ibus
    if reset = '1' then                 -- asynchronous reset (active low)
      FI_i_am_addressed <= '0';
      this_is_a_ctrl_transaction <= '0';
    elsif clk'event and clk = '1' then  -- rising clock edge
      ack <= '0';
      il_req <= '0';
      request_reset <= '0';
      receive_enabled <= '1';   -- this side is always enabled
      loopback_state <= '0';  -- disable loopback
      case (FI_read_state) is
        when idle =>
          ack <= '0';
          if (req = '1') then
            il_io <= data;
            FI_read_state <= am_i_addressed;
            timeout <= "0000";
          end if;
        when am_i_addressed =>
          FI_read_state <= idle;
          if (data(15) = '1') then -- this is an address
            FI_i_am_addressed <= '0';   -- any addtress transaction should
                                        -- reset select bits on all chips
            this_is_a_ctrl_transaction <= '0';
            if (data(12) = i_am_remote) then
              if (data(11 downto 8) = my_ctrl_address) then
                this_is_a_ctrl_transaction <= '1';
                --                FI_i_am_addressed <= '1';
                -- don't loopback                 loopback_state <= data(4);
                request_reset <= data(5);
                -- this side is always enabled -- receive_enabled <= data(0);
                address <= data(7 downto 0);
                FI_read_state <= FI_hs_for_address;
              else
                if (data(11 downto 8) = my_data_address) then
                --                FI_i_am_addressed <= '1';
                address <= data(7 downto 0);
                FI_read_state <= FI_hs_for_address;
                end if;
              end if;
            --
            -- This is not an address, but if this chip is already addressed
            -- then this will be a data transaction to complete the previous
            -- address transaction
            --
            --            if (FI_i_am_addressed = '1') then
              FI_read_state <= FI_hs_for_data;
            --          else
            --            FI_read_state <= internal_ack1; --
            --          end if;

          end if;
        else
        FI_read_state <= FI_hs_for_data;
        end if;
        when FI_hs_for_address =>
          ack <= '1';
          if (req = '0') then
              FI_read_state <= idle;
          else
            if (timeout = "1111") then
              -- Hmm this timeout will lead to a spurious transaction, but not
              -- to worry, there must be a serious problem somewhere else in
              -- this case
              FI_read_state <= idle;
            else
              timeout <= timeout+1;
            end if;
          end if;
        when FI_hs_for_data =>
          ack <= '1';
          if (req = '0') then
            --
            -- things to do once the external transaction has completed
            --
            if (this_is_a_ctrl_transaction = '1') then
              FI_read_state <= FI_use_ctrl_data;
            else
              FI_read_state <= internal_ack1;
              timeout <= "0000";
            end if;
          else
            if (timeout = "1111")  then
              --
              -- If this timeout is triggered we are likely to keep looping.
              -- By gong to an idle state we ensure that only 1 data transaction
              -- can take place anyway, thought there is a possibility of
              -- losing a transaction as well.
              --
              FI_read_state <= idle;
            else
              timeout <= timeout+1;
            end if;
          end if;
          if (this_is_a_ctrl_transaction = '0') then
            --
            -- Get a jump start on completing internal transactions
            --
            il_req <= '1';  -- attempt to transfer data out
          end if;
        when FI_use_ctrl_data =>
          --
          -- What we do with the control data is completely dependent on the
          -- chip all of the rest of this code should be fairly generic, if
          -- a littl over elaborate in most cases
          --
          --if (address(1 downto 0) = "01") then
            --c1: for i in 0 to 7 loop
--* put reset fifo here            --counter_start(i) <= data(i);
              --counter_end(i) <= data(i+8);
            --end loop c1;
          --end if;
            FI_read_state <= idle;
      when internal_ack1 =>
          il_req <= '1';
          if (il_ack = '1') then
            timeout <= "0000";
            il_req <= '0';
            FI_read_state <= internal_ack2;
          else
            if (timeout = "1111") then
              FI_read_state <= idle;
            else
              timeout <= timeout+1;
            end if;
          end if;
       when internal_ack2 =>
          il_req <= '0';
          if (il_ack = '0') then
            FI_read_state <= idle;
          else
            if (timeout = "1111") then
              FI_read_state <= idle;
            else
              timeout <= timeout+1;
```

```
281             end if;
282           end if;
283         when others =>
284           FI_read_state <= idle;
285       end case;
286     end if;
287   end process read_FI;
288 end read_from_FI_arch;
289
290
291 library ieee;
292 use ieee.std_logic_1164.all;
293 use work.numeric_std.all;
294
295 entity ibus_FO_action is
296     port (
297       clk : in std_logic;
298       reset : in std_logic;
299       il_i_am_addressed : in std_logic;
300       fo_req :  in std_logic;
301       fo_ack : buffer std_logic;
302       data :    in std_logic_vector(15 downto 0);
303       fiber_clk: in std_logic;
304       fo_d: out std_logic_vector(9 downto 0);
305       fo_ENA_l:  buffer std_logic
306     );
307
308 end ibus_FO_action;
309
310 architecture arch_ibus_FO_action of ibus_FO_action is
311   -- purpose: <description>
312    type states is (fo_idle,fo_byte1,fo_wait1,fo_byte2,fo_wait2);
313    signal state : states;
314    signal fiber_out_buf : std_logic_vector(15 downto 0);
315    signal byte_buf : std_logic_vector(7 downto 0);
316 --   signal fo_send : std_logic;    -- <comment>
317    alias sc : std_logic is fo_d(8);
318    alias svs : std_logic is fo_d(9);
319
320   -- purpose: performs the actual monitor action
321   -- type:    memorizing
322   -- inputs:  clk, reset
323   -- outputs:
324 begin
325   FO_action : process (clk, reset)
326   begin  -- process FO_action
327     if reset = '1' then
328 --    fo_ENA_l <= '1';
329     --  if reset_l = '0' then
330       -- activities triggered by rising edge of clock
331       elsif clk'event and clk = '1' then
332 --    fo_send <= '0';   -- enable needs to stay low for both bytes
333 --    fo_ENA_l <= '0';
334       case state is
335           when fo_idle =>
336 --        fo_send <= '0';
337           fo_ENA_l <= '1';
338           fo_d(8) <= '0';   -- command channel flag
339           sc <= '0';
340           svs <= '0';
341           fo_ack  <= '0';
342           if (fo_req = '1') then
343               fo_d(7 downto 0) <= data(7 downto 0);
344 state <= fo_byte1;
345 --            if (fiber_clk = '1') then
346               -- if fiber clock is high it is now
347               -- transitioning to low, which
348               -- means data will be tranmitted
349               -- on next cycle. since that
350               -- will present a rising edge
351              --fo_not_strobe
352 --             state <= fo_wait1;
353              --
354              -- Hmm appears to be a bug in the component desciption
355              -- it appears that a low on ena any time that the
356              -- clock is high sends the data, rather than
357              -- the rising edge as written.
358              -- fo_send <= '1';
359 --             fo_send <= '1';
360 --          fo_ENA_l <= '0';
361 --           else
362 --             state <= fo_byte1;
363 --           end if;
364            --state <= fo_idle;
365         end if;
366           when fo_byte1 =>
367 --          fo_send <= '1';
368            if (fiber_clk = '1') then
369              fo_ENA_l <= '0';
370              state <= fo_wait1;
371              fo_ack  <= '1';    -- so other side has time to see it
372            else
373              state <= fo_byte1;
374            end if;
375           when fo_wait1 =>
376 --          fo_send <= '1';
377            fo_ENA_l <= '0';
378            state <= fo_byte2;
379           when fo_byte2 =>
380           --
381           -- here fiber clock transitioning from high to low
382           --
383 --        fo_send <= '1';
384         fo_ENA_l <= '0';
385           fo_d(7 downto 0) <=  data(15 downto 8);
386           state <= fo_wait2;
387           when fo_wait2 =>
388           -- final low to high
389           --
390           --
391           -- Give this fiber strobe time tp get back to high
392           --
393 --        fo_send <= '1';
394         fo_ENA_l <= '1';
395           state <= fo_idle;
396           when others =>
397           state <= fo_idle;
398       end case;
399       end if;
400   end process FO_action;
401 --    fo_ENA_l <= '0' when (fo_send = '1') else '1';
402 end arch_ibus_FO_action;
403
404
405
406 library ieee;
407 use ieee.std_logic_1164.all;
408 use work.numeric_std.all;
409
410 entity ibus_FI_port is
411
412   port (
413     clk:        in     std_logic;
414     reset :     in std_logic;
415     data :      buffer std_logic_vector(15 downto 0);
416     req   : buffer std_logic;
417     ack   : in  std_logic;
418     fiber_to_ibus_buf :    in std_logic_vector(15 downto 0);
419     data_pump_word_ready   : in std_logic;
420     refill_ibus_output_buf  : out std_logic
```

```
421     );
422
423 end ibus_FI_port;
424
425 architecture arch_ibus_FI_port of ibus_FI_port is
426   type read_states  is (idle,read_req,end_cycle);
427   signal read_state: read_states;
428   signal timeout : unsigned(3 downto 0);
429 begin
430     ibus_FI_read : process (clk, reset)
431     begin
432     if reset = '1' then
433 --      refill_ibus_output_buf <= '1';
434 refill_ibus_output_buf <= '0';
435           -- asynchronous reset (active low)
436     elsif clk'event and clk = '1' then
437             req <= '0';
438 refill_ibus_output_buf <= '1';
439         case read_state is
440         when  idle =>
441          req <= '0';
442          timeout <= "0000";
443             if (data_pump_word_ready = '1') then
444                 if (ack = '0') then
445                     data(15 downto 0) <= fiber_to_ibus_buf(15 downto 0);
446                     req <= '1';
447 --                  refill_ibus_output_buf <= '0';
448                     read_state <= read_req;
449                 else
450                 read_state <= idle;
451                 end if;
452             end if;
453         when read_req =>
454             if (ack = '1') then
455             req <= '0';
456             read_state <= end_cycle;
457 else
458   if (timeout = "1111") then
459     read_state <= idle;
460   else
461     timeout <= timeout+1;
462     read_state <= read_req;
463   end if;
464 end if;
465         when end_cycle =>
466 refill_ibus_output_buf <= '1';
467             if (ack = '0') then
468 --          refill_ibus_output_buf <= '1';
469 refill_ibus_output_buf <= '0';
470                 read_state <= idle;
471             else
472                 if (timeout = "1111") then
473                 read_state <= idle;
474                 else
475                 timeout <= timeout+1;
476                 read_state <= end_cycle;
477                 end if;
478             end if;
479         when others =>
480            read_state <= idle;
481         end case;
482     end if;
483     end process ibus_FI_read;
484 end arch_ibus_FI_port;
485
486
487
488 library ieee;
489 use ieee.std_logic_1164.all;
490 use work.numeric_std.all;
491
492 entity fiber_rec is
493
494     port (clk : in std_logic;
495     reset :     in std_logic;
496       fr_d:  in std_logic_vector(11 downto 0);  -- data bus from fiber rec
497       fr_RDY_l: in std_logic;                   -- from fiber rec RDY
498       fr_status: in std_logic;                  -- from fiber rec SO
499       receive_enabled: in std_logic;
500       increment_fifo_count: buffer std_logic;
501       violation_count: buffer unsigned(7 downto 0);
502       fifo_reset_l: buffer std_logic;           -- output to FIFO reset
503       fifo_D: out std_logic_vector(8 downto 0); -- output to FIFO d inputs
504       fifo_WRITE_l: out std_logic;              -- FIFO write strobe
505       reset_fifo_request: in std_logic
506       );
507 end fiber_rec;
508
509 -- purpose: manages recept of fiber data and its storage in the fifo
510 architecture arch_fiber_rec of fiber_rec is
511   type fr_states  is (idle,fifo_latch);-- handshake states
512   signal fr_state : fr_states;  -- handshake
513   alias code_violation : std_logic is fr_d(9);
514   alias control_byte : std_logic is fr_d(8);
515 begin  -- fiber_rec_arch
516     -- type:     memorizing
517     -- inputs:  clk, reset
518     -- outputs: <signal names>
519 -- had to just pass thru receiver data to FIFO input and RDY to write strobe
520 -- to meet FIFO WR strobe min of 30nsec. Receiver has 60/40% duty cycle.
521 -- Otherwise we would have a 20nsec write strobe.  Code could be cleaned
522 -- up later, but leaving state machine in for now....
523 fifo_D <= fr_d(8 downto 0);
524 fifo_WRITE_l <= fr_RDY_l;
525     fr : process (clk, fr_RDY_l,reset_fifo_request,reset)
526     begin  -- process fr
527      if (reset_fifo_request = '1' or reset = '1') then
528         fifo_reset_l <= '0';
529         -- activities triggered by asynchronous reset (active low)
530     elsif clk'event and clk = '1' then
531         case fr_state is
532         when idle =>
533             if (reset_fifo_request = '1') then
534             fifo_reset_l <= '0';
535             violation_count <= (others => '0');
536             else
537             if (fr_status = '1' and receive_enabled = '1') then
538                 if ( fr_RDY_l = '0') then
539                 fr_state <= fifo_latch;
540                 increment_fifo_count <= '1';
541                 fifo_reset_l <= '1';
542                 end if;
543             end if;
544             end if;
545         when fifo_latch =>
546             if (reset_fifo_request = '1') then
547             fifo_reset_l <= '0';
548             end if;
549             fr_state <= idle;
550         when others =>
551             fr_state <= idle;
552         end case;
553     end if;
554     end process fr;
555 end arch_fiber_rec;
556
557
558 library ieee;
559 use ieee.std_logic_1164.all;
560 use work.numeric_std.all;
```

```
561
562 entity fifo_data_pump is
563
564     port (clk : in std_logic;
565       reset : in std_logic;
566       violation_count: in unsigned(7 downto 0);
567       fiber_to_ibus_buf: buffer std_logic_vector(15 downto 0);
568       fifo_OUT: in std_logic_vector(8 downto 0);
569       fifo_READ_l: out std_logic;
570       data_pump_word_ready: buffer std_logic;
571       refill_ibus_output_buf: in std_logic;
572       fifo_reset_l: in std_logic;
573       fifo_EMPTY_l: in std_logic
574       );
575 end fifo_data_pump;
576
577 -- purpose: pump data out of the fifo
578 architecture arch_fifo_data_pump of fifo_data_pump is
579   type pump_states  is (idle,fifo_strobe,fifo_read_data,fifo_wait_on_empty);-- handshake states
580   signal pump_state : pump_states;  -- handshake
581 signal count : unsigned(1 downto 0);
582 begin  -- fifo_data_pump_arch
583     -- purpose: <description>
584
585     pump : process (clk,reset)
586
587     begin  -- process pump
588     -- activities triggered by asynchronous reset (active low)
589     if reset = '1' then
590         fifo_READ_l <= '1';
591     elsif clk'event and clk = '1' then
592         data_pump_word_ready <= '0';
593         fifo_READ_l <= '1';
594         case pump_state is
595         when idle =>
596             count <= "00";
597             if (fifo_reset_l = '1' or refill_ibus_output_buf = '1') then
598                 data_pump_word_ready <= '0';
599                 if (fifo_empty_l = '0') then
600                 -- This is an inverted signal so here there is
601                 -- no data in the fifo, must now wait for data
602                 -- to become available
603                     pump_state <= fifo_wait_on_empty;
604                 else
605 --                  pump_state <= fifo_strobe;
606                     pump_state <= fifo_wait_on_empty;   -- add a delay to make sure FIFO is ready
607                 end if;
608             else
609             pump_state <= idle;
610             end if;
611         when fifo_wait_on_empty =>
612             data_pump_word_ready <= '0';  --- may need to chsnge to 1
613             if (fifo_EMPTY_l = '0') then
614             pump_state <= fifo_wait_on_empty;
615             else
616             pump_state <= fifo_strobe;
617             end if;
618         when fifo_strobe =>
619 if (fifo_empty_l = '0') then
620 pump_state <= fifo_wait_on_empty;
621 else
622             data_pump_word_ready <= '0';
623             fifo_READ_l <= '0';
624             pump_state <= fifo_read_data;
625 end if;
626         when fifo_read_data =>
627             if (fifo_out(8) = '1') then
628                 count <= "00";
629                 fifo_READ_l <= '1';
630                 pump_state <= fifo_strobe;
631             else
632                 fiber_to_ibus_buf(15 downto 8) <= fifo_out(7 downto 0);
633                 fiber_to_ibus_buf(7 downto 0) <=
634                         fiber_to_ibus_buf(15 downto 8);
635                 if (count  = "01") then
636                     data_pump_word_ready <= '1';
637                     pump_state <= idle;
638                 else
639                     count <= count +1;
640                     fifo_READ_l <= '1';
641                     if (fifo_empty_l = '0') then
642                         pump_state <= fifo_wait_on_empty;
643                     else
644                         pump_state <= fifo_strobe;
645                     end if;
646                 end if;
647             end if;
648         when others =>
649             pump_state <= idle;
650         end case;
651 end if;
652     end process pump;
653 end arch_fifo_data_pump;
654
655
656
657 library ieee;
658 use ieee.std_logic_1164.all;
659 use work.numeric_std.all;
660
661 entity tout is
662   port (
663       clk : in std_logic;
664       fast : in std_logic;
665       slow : in std_logic;
666       DEBUG : buffer std_logic;
667       --
668       -- on board data bus and associated control signals
669       --
670       id: inout std_logic_vector(31 downto 0);
671       --
672       -- global
673       --
674       reset: in std_logic;
675       AO_FROM_PC_STROBE: buffer std_logic;
676       AO_FROM_PC_ACK: in std_logic;
677       AO_TO_PC_STROBE: in std_logic;
678       AO_TO_PC_ACK: buffer std_logic;
679       in_strobe: in std_logic;
680       -- fiber reciever
681       --
682       fr_d: in std_logic_vector(11 downto 0);
683       fr_ref_clk: out std_logic;
684       fr_rf:      buffer std_logic;
685       fr_mode:    out std_logic;
686       fr_status:  in std_logic;
687       fr_RDY_l: in std_logic;
688       fr_ckr:     in std_logic;
689       --
690       -- fiber output section
691       --
692       fo_d: out std_logic_vector(9 downto 0);
693       fo_ENN_l: out std_logic;
694       fo_ENA_l: buffer std_logic;
695       fo_CKW:   buffer std_logic;
696       fo_mode:    out std_logic;
697       fo_foto:    out std_logic;
698       fo_RP_l:  in  std_logic;
699       --
700       -- fifo stuff
```

```
701       --
702       fifo_reset_l: buffer std_logic;   -- Declare as a buffer since we want
703       fifo_WRITE_l: out std_logic;
704       fifo_D:    out  std_logic_vector(8 downto 0);
705       fifo_READ_l:  out std_logic;
706       fifo_FULL_l:  in std_logic;
707       fifo_HALF_l:  in std_logic;
708       fifo_EMPTY_l: in std_logic;
709       fifo_OUT:   in std_logic_vector(8 downto 0)
710     );
711 end tout;
712
713
714 -- purpose: Collect together preceeding modules
715 architecture tout_arch of tout is
716
717     signal write_oe : std_logic;
718     signal read_oe : std_logic;
719     signal fo_data_strobe : std_logic;
720     signal violation_count : unsigned(7 downto 0);
721     signal decriment_fifo_count : std_logic;
722     signal increment_fifo_count : std_logic;
723     signal data_pump_word_ready:    std_logic;
724     signal refill_ibus_output_buf: std_logic;
725     signal fiber_to_ibus_buf : std_logic_vector(15 downto 0);
726     signal reset_fifo : std_logic;
727     signal rec_enabled : std_logic;
728
729     type tristate_state_type is (idle,active);
730 signal tristate_state : tristate_state_type;
731 signal il_tristate : std_logic;
732 signal il_ack: std_logic;
733 signal i_am_remote : std_logic;        -- set to 1 for remote board 0 for local
734 signal my_data_address : std_logic_vector(3 downto 0);
735 signal my_ctrl_address : std_logic_vector(3 downto 0);
736 signal from_FI_ack : std_logic;
737 signal from_FI_req : std_logic;
738 signal data_from_pc : std_logic_vector(15 downto 0);
739 signal address_from_pc : std_logic_vector(7 downto 0);
740 signal power_up_reset: std_logic;
741 signal soft_request_reset: std_logic;
742 signal data_to_FO : std_logic_vector(15 downto 0);
743 signal this_chip_selected: std_logic;
744 signal FI_address : std_logic_vector(7 downto 0);
745 signal FI_to_ibus_req : std_logic;
746 signal FI_to_ibus_ack : std_logic;
747 signal FI_data: std_logic_vector(15 downto 0);
748 signal write_to_FO_req : std_logic;
749 signal write_to_FO_ack : std_logic;
750 -- alias reset_fifo : std_logic is request_reset;
751
752 begin
753     --    reset <= '1';             -- negative logic
754     DEBUG <= AO_FROM_PC_STROBE;
755     i_am_remote <= '1';
756     my_data_address <= "1111";
757     my_ctrl_address <= "0010";
758   --
759     -- Fiber output section
760     --
761     outstrobe : process (clk,reset)
762     begin  -- process outstrobe
763     -- activities triggered by asynchronous reset (active low)
764     if clk'event and clk = '1' then
765         fo_data_strobe <= not fo_data_strobe;
766     end if;
767     end process outstrobe;
768     fo_CKW <= fo_data_strobe;
769     fo_mode <= '0';
770     fo_foto <= '0';
771     fo_ENN_l <= '1';
772
773    ibus_reader : read_from_ibus port map (
774      clk       => clk,
775      reset     => reset,
776      iu_req    => AO_TO_PC_STROBE,  --  in
777      iu_ack    => AO_TO_PC_ACK,     --  out
778      iu_io     => id(31 downto 16),
779      data      => data_to_FO,
780      req       => write_to_FO_req,      -- out
781      ack       => write_to_FO_ack       -- in
782      );
783      -- direction: received FI=PC writes output to ibus_lower
784   FI_reader : read_from_FI port map (
785     clk             => clk,
786     reset           => reset,
787     req             => FI_to_ibus_req,  -- in
788     ack             => FI_to_ibus_ack,  -- out
789     data            => FI_data,
790     il_io           => id(15 downto 0),
791     address         => address_from_pc,
792     il_req          => AO_FROM_PC_STROBE,   -- out
793     il_ack          => AO_FROM_PC_ACK,      -- in
794     i_am_remote     => i_am_remote,
795     my_ctrl_address => my_ctrl_address,
796     my_data_address => my_data_address,
797     request_reset   => reset_fifo,
798     FI_i_am_addressed => this_chip_selected,
799     receive_enabled   => rec_enabled
800     );
801
802     --
803     ibus_FO: ibus_FO_action port map (
804         clk             => clk,
805         reset           => reset,
806         il_i_am_addressed => this_chip_selected,
807         fo_req          => write_to_FO_req, -- in
808         fo_ack          => write_to_FO_ack, -- out
809         data            => data_to_FO,
810         fiber_clk       => fo_CKW,
811         fo_d            => fo_d,
812         fo_ENA_l        => fo_ENA_l
813         );
814     --
815     -- fiber receiver stuff
816     --
817     fr_ref_clk <= fo_data_strobe;
818     fr_mode <= '0';
819     fr_rf <= '1';
820     ibus_FI: ibus_FI_port port map (
821         clk             => clk,
822         reset           => reset,
823         data            => FI_data,
824         req             => FI_to_ibus_req,  -- out
825         ack             => FI_to_ibus_ack,  -- in
826         fiber_to_ibus_buf       => fiber_to_ibus_buf,
827         data_pump_word_ready    => data_pump_word_ready,
828         refill_ibus_output_buf  => refill_ibus_output_buf
829         );
830
831     fr_imp: fiber_rec port map (
832         clk             => clk,
833         reset           => reset,
834         fr_d            => fr_d,
835         fr_RDY_l        => fr_RDY_l,
836         fr_status       => fr_status,
837         receive_enabled => rec_enabled,
838         increment_fifo_count    => increment_fifo_count,
839         violation_count         => violation_count,
840         fifo_reset_l            => fifo_reset_l,
```

```
841         fifo_D          => fifo_D,
842         fifo_WRITE_l    =>  fifo_WRITE_l,
843         reset_fifo_request      => reset_fifo
844                 );
845
846     pump: fifo_data_pump port map (
847         clk             => clk,
848         reset           => reset,
849         violation_count =>  violation_count,
850         fiber_to_ibus_buf =>  fiber_to_ibus_buf,
851         fifo_OUT        => fifo_OUT,
852         fifo_READ_l     => fifo_READ_l,
853         data_pump_word_ready => data_pump_word_ready,
854         refill_ibus_output_buf => refill_ibus_output_buf,
855         fifo_reset_l    =>  fifo_reset_l,
856         fifo_EMPTY_l => fifo_EMPTY_l
857                     );
858     --
859     end  tout_arch;
860
861
862
863
864
865
866
867
868
869
870
871
872
873
```